



Kivy Documentation

Release 2.0.0.dev0

www.kivy.org

Contents

I	Kivy Project	3
1	Philosophy	5
2	Contributing	7
3	FAQ	23
4	Contact Us	29
II	Programming Guide	31
5	Kivy Basics	33
6	Controlling the environment	39
7	Configure Kivy	43
8	Architectural Overview	45
9	Events and Properties	51
10	Input management	61
11	Widgets	69
12	Graphics	89
13	Kv language	93
14	Integrating with other Frameworks	103
15	Packaging your application	107
16	Package licensing	125

III Tutorials	129
17 Pong Game Tutorial	131
18 A Simple Paint App	145
19 Crash Course	155
IV API Reference	157
20 Kivy framework	159
21 Core Abstraction	259
22 Kivy module for binary dependencies.	293
23 Effects	295
24 Garden	301
25 Graphics	305
26 Input management	381
27 Kivy Language	403
28 External libraries	425
29 Modules	429
30 Network support	441
31 Storage	445
32 Tools	451
33 Widgets	453
V Appendix	695
34 License	697
Python Module Index	699
Index	701

Welcome to Kivy's documentation. Kivy is an open source software library for the rapid development of applications equipped with novel user interfaces, such as multi-touch apps.

We recommend that you get started with *Getting Started*. Then head over to the *Programming Guide*. We also have *Create an application* if you are impatient.

You are probably wondering why you should be interested in using Kivy. There is a document outlining our *Philosophy* that we encourage you to read, and a detailed *Architectural Overview*.

If you want to contribute to Kivy, make sure to read *Contributing*. If your concern isn't addressed in the documentation, feel free to *Contact Us*.

Part I

KIVY PROJECT

This part of the documentation explains the basic ideas behind Kivy's design and why you'd want to use it.

Philosophy

In case you are wondering what Kivy is all about and what sets it apart from other solutions, this document is for you.

1.1 Why bother?

Why would you want to use Kivy? After all, there are many great toolkits (or frameworks, or platforms) available out there – for free. You have Qt and Flash, to name just two good choices for application development. Many of these numerous solutions already support Multi-Touch, so what is it that makes Kivy special and worth using?

1.1.1 Fresh

Kivy is made for today and tomorrow. Novel input methods such as Multi-Touch have become increasingly important. We created Kivy from scratch, specifically for this kind of interaction. That means we were able to rethink many things in terms of human computer interaction, whereas older (not to mean ‘outdated’, rather ‘well-established’) toolkits carry their legacy, which is often a burden. We’re not trying to force this new approach to using a computer into the corset of existing models (say single-pointer mouse interaction). We want to let it flourish and let you explore the possibilities. *This* is what really sets Kivy apart.

1.1.2 Fast

Kivy is fast. This applies to both *application development* and *application execution* speeds. We have optimized Kivy in many ways. We implement time-critical functionality on the *C level* to leverage the power of existing compilers. More importantly, we also use *intelligent algorithms* to minimize costly operations. We also use the *GPU* wherever it makes sense in our context. The computational power of today’s graphics cards surpasses that of today’s CPUs by far for some tasks and algorithms, especially drawing. That’s why we try to let the GPU do as much of the work as possible, thus increasing performance considerably.

1.1.3 Flexible

Kivy is flexible. This means it can be run on *a variety of different devices*, including Android powered smartphones and tablets. We support *all major operating systems* (Windows, Linux, OS X). Being flexible also means that Kivy's fast-paced development allows it to *adapt to new technologies quickly*. More than once have we added support for new external devices and software protocols, sometimes even before they were released. Lastly, Kivy is also flexible in that it is possible to use it in combination with a great number of different third-party solutions. For example, on Windows we support WM_TOUCH, which means that any device that has Windows 7 Pen & Touch drivers will *just work* with Kivy. On OS X you can use Apple's Multi-Touch capable devices, such as trackpads and mice. On Linux, you can use HID kernel input events. In addition to that, we support TUIO (Tangible User Interface Objects) and a number of other input sources.

1.1.4 Focused

Kivy is focused. You can write a simple application with a few lines of code. Kivy programs are created using the *Python* programming language, which is incredibly versatile and powerful, yet easy to use. In addition, we created our own description language, the *Kivy Language*, for creating sophisticated user interfaces. This language allows you to set up, connect and arrange your application elements quickly. We feel that allowing you to focus on the essence of your application is more important than forcing you to fiddle with compiler settings. We took that burden off your shoulders.

1.1.5 Funded

Kivy is actively developed by professionals in their field. Kivy is a community-influenced, professionally developed and commercially backed solution. Some of our core developers develop Kivy for a living. Kivy is here to stay. It's not a small, vanishing student project.

1.1.6 Free

Kivy is free to use. You don't have to pay for it. You don't even have to pay for it if you're making money out of selling an application that uses Kivy.

Contributing

There are many ways in which you can contribute to Kivy. Code patches are just one thing amongst others that you can submit to help the project. We also welcome feedback, bug reports, feature requests, documentation improvements, advertisement & advocating, testing, graphics contributions and many other ideas. Just talk to us if you want to help, and we will help you help us.

2.1 Discussions

Discussions around Kivy development happens on Github's issues and pull requests for specific things. For things that don't fit in either, discussions happen on the [#dev Discord channel](#), and on the [kivy-dev google group](#). Please come ask for guidance if you are unsure about how to contribute, or you want confirmation about your ideas fitting in the project before working on them. If you want to ask for — or contribute — support, you can join the [#support Discord channel](#), and the [kivy-users google group](#).

2.2 Code of Conduct

In the interest of fostering an open and welcoming community, we as contributors and maintainers need to ensure participation in our project and our sister projects is a harassment-free and positive experience for everyone.

As such, it is vital that all interaction is conducted in a manner conveying respect, open-mindedness and gratitude. For a more comprehensive discussion of these guidelines, please refer to the [Contributor Covenant](#). This document provides an accurate description of what is expected of you, both as a core developer or a first time contributor.

2.3 Feedback

This is by far the easiest way to contribute something. If you're using Kivy for your own project, don't hesitate sharing. It doesn't have to be a high-class enterprise app, obviously. It's just incredibly

motivating to know that people use the things you develop and what it enables them to do. If you have something that you would like to tell us, please don't hesitate. Screenshots and videos are also very welcome! We're also interested in the problems you had when getting started. Please feel encouraged to report any obstacles you encountered such as missing documentation, misleading directions or similar. We are perfectionists, so even if it's just a typo, let us know.

2.4 Reporting an Issue

If you found anything wrong, a crash, segfault, missing documentation, invalid spelling or just weird examples, please take 2 minutes to report the issue.

1. Move your logging level to debug by editing `<user_directory>/kivy/config.ini`:

```
[kivy]
log_level = debug
```

2. Execute your code again, and copy/paste the complete output to <http://gist.github.com/>, including the log from Kivy and the python backtrace.
3. Open <https://github.com/kivy/kivy/issues/>
4. Set the title of your issue
5. Explain exactly what to do to reproduce the issue and paste the link of the output posted on <http://gist.github.com/>
6. Validate the issue and you're done!

If you are feeling up to it, you can also try to resolve the bug, and contribute by sending us the patch :) Read the next section to find out how to do this.

2.5 Code Contributions

Code contributions (patches, new features) are the most obvious way to help with the project's development. Since this is so common we ask you to follow our workflow to most efficiently work with us. Adhering to our workflow ensures that your contribution won't be forgotten or lost. Also, your name will always be associated with the change you made, which basically means eternal fame in our code history (you can opt-out if you don't want that).

2.5.1 Coding style

- If you haven't done it yet, read the [PEP8](#) about coding style in python.
- Activate the pep8 check on git commits like this:

```
make hook
```

This will pass the code added to the git staging zone (about to be committed) through a pep8 checker program when you do a commit, and ensure that you didn't introduce pep8 errors. If you did, the commit will be rejected: please correct the errors and try again.

2.5.2 Performance

- take care of performance issues: read [Python performance tips](#)

- cpu intensive parts of Kivy are written in cython: if you are doing a lot of computation, consider using it too.

2.5.3 Git & GitHub

We use git as our version control system for our code base. If you have never used git or a similar DVCS (or even any VCS) before, we strongly suggest you take a look at the great documentation that is available for git online. The [Git Community Book](#) or the [Git Videos](#) are both great ways to learn git. Trust us when we say that git is a great tool. It may seem daunting at first, but after a while you'll (hopefully) love it as much as we do. Teaching you git, however, is well beyond the scope of this document.

Also, we use [GitHub](#) to host our code. In the following we will assume that you have a (free) GitHub account. While this part is optional, it allows for a tight integration between your patches and our upstream code base. If you don't want to use GitHub, we assume you know what you are doing anyway.

2.5.4 Code Workflow

So here is the initial setup to begin with our workflow (you only need to do this once to install Kivy). Basically you follow the installation instructions from dev-install, but you don't clone our repository, you fork it. Here are the steps:

1. Log in to GitHub
2. Create a fork of the [Kivy repository](#) by clicking the *fork* button.
3. Clone your fork of our repository to your computer. Your fork will have the git remote name 'origin' and you will be on branch 'master':

```
git clone https://github.com/username/kivy.git
```

4. Compile and set up PYTHONPATH or install (see dev-install).
5. Install our pre-commit hook that ensures your code doesn't violate our styleguide by executing *make hook* from the root directory of your clone. This will run our styleguide check whenever you do a commit, and if there are violations in the parts that you changed, your commit will be aborted. Fix & retry.
6. Add the kivy repo as a remote source:

```
git remote add kivy https://github.com/kivy/kivy.git
```

Now, whenever you want to create a patch, you follow the following steps:

1. See if there is a ticket in our bug tracker for the fix or feature and announce that you'll be working on it if it doesn't yet have an assignee.
2. Create a new, appropriately named branch in your local repository for that specific feature or bugfix. (Keeping a new branch per feature makes sure we can easily pull in your changes without pulling any other stuff that is not supposed to be pulled.):

```
git checkout -b new_feature
```

3. Modify the code to do what you want (e.g. fix it).
4. Test the code. Try to do this even for small fixes. You never know whether you have introduced some weird bug without testing.

5. Do one or more minimal, atomic commits per fix or per feature. Minimal/Atomic means *keep the commit clean*. Don't commit other stuff that doesn't logically belong to this fix or feature. This is **not** about creating one commit per line changed. Use `git add -p` if necessary.
6. Give each commit an appropriate commit message, so that others who are not familiar with the matter get a good idea of what you changed.
7. Once you are satisfied with your changes, pull our upstream repository and merge it with you local repository. We can pull your stuff, but since you know exactly what's changed, you should do the merge:

```
git pull kivy master
```

8. Push your local branch into your remote repository on GitHub:

```
git push origin new_feature
```

9. Send a *Pull Request* with a description of what you changed via the button in the GitHub interface of your repository. (This is why we forked initially. Your repository is linked against ours.)

Warning: If you change parts of the code base that require compilation, you will have to recompile in order for your changes to take effect. The `make` command will do that for you (see the Makefile if you want to know what it does). If you need to clean your current directory from compiled files, execute `make clean`. If you want to get rid of **all** files that are not under version control, run `make distclean` (**Caution:** If your changes are not under version control, this command will delete them!)

Now we will receive your pull request. We will check whether your changes are clean and make sense (if you talked to us before doing all of this we will have told you whether it makes sense or not). If so, we will pull them and you will get instant karma. Congratulations, you're a hero!

2.6 Documentation Contributions

Documentation contributions generally follow the same workflow as code contributions, but are just a bit more lax.

1. Following the instructions above,
 - (a) Fork the repository.
 - (b) Clone your fork to your computer.
 - (c) Setup kivy repo as a remote source.
2. Install python-sphinx. (See `docs/README` for assistance.)
3. Use **ReStructuredText Markup** to make changes to the HTML documentation in `docs/sources`.

To submit a documentation update, use the following steps:

1. Create a new, appropriately named branch in your local repository:

```
git checkout -b my_docs_update
```

2. Modify the documentation with your correction or improvement.
3. Re-generate the HTML pages, and review your update:

```
make html
```

4. Give each commit an appropriate commit message, so that others who are not familiar with the matter get a good idea of what you changed.
5. Keep each commit focused on a single related theme. Don't commit other stuff that doesn't logically belong to this update.
6. Push to your remote repository on GitHub:

```
git push
```

7. Send a *Pull Request* with a description of what you changed via the button in the GitHub interface of your repository.

We don't ask you to go through all the hassle just to correct a single typo, but for more complex contributions, please follow the suggested workflow.

2.6.1 Docstrings

Every module/class/method/function needs a docstring, so use the following keywords when relevant:

- `.. versionadded::` to mark the version in which the feature was added.
- `.. versionchanged::` to mark the version in which the behaviour of the feature was changed.
- `.. note::` to add additional info about how to use the feature or related feature.
- `.. warning::` to indicate a potential issue the user might run into using the feature.

Examples:

```
def my_new_feature(self, arg):  
    """  
    New feature is awesome  
  
    .. versionadded:: 1.1.4  
  
    .. note:: This new feature will likely blow your mind  
  
    .. warning:: Please take a seat before trying this feature  
    """
```

Will result in:

```
def my_new_feature(self, arg): New feature is awesome  
    New in version 1.1.4.
```

Note: This new feature will likely blow your mind

Warning: Please take a seat before trying this feature

When referring to other parts of the api use:

- `:mod:`~kivy.module`` to refer to a module
- `:class:`~kivy.module.Class`` to refer to a class

- `:meth:~kivy.module.Class.method`` to refer to a method
- `:doc:~api-kivy.module`` to refer to the documentation of a module (same for a class and a method)

Obviously replacing *module* *Class* and *method* with their real name, and using using `'.'` to separate modules referring to imbricated modules, e.g:

```
:mod:~kivy.uix.floatlayout`
:class:~kivy.uix.floatlayout.FloatLayout`
:meth:~kivy.core.window.WindowBase.toggle_fullscreen`
:doc:~/api-kivy.core.window`
```

Will result in:

floatlayout FloatLayout toggle_fullscreen() Window

`:doc:` and `:mod:` are essentially the same, except for an anchor in the url which makes `:doc:` preferred for the cleaner url.

To build your documentation, run:

```
make html
```

If you updated your kivy install, and have some trouble compiling docs, run:

```
make clean force html
```

The docs will be generated in `docs/build/html`. For more information on docstring formatting, please refer to the official [Sphinx Documentation](#).

2.7 Unit tests contributions

For the testing team, we have the document *Unit tests* that explains how Kivy unit tests work and how you can create your own. Use the same approach as the *Code Workflow* to submit new tests.

2.7.1 Unit tests

Tests are located in the *kivy/tests* folder. If you find a bug in Kivy, a good thing to do can be to write a minimal case showing the issue and to ask core devs if the behaviour shown is intended or a real bug. If you write your code as a *unittest*, it will prevent the bug from coming back unnoticed in the future, and will make Kivy a better, stronger project. Writing a unittest may be a really good way to get familiar with Kivy while doing something useful.

Unit tests are separated into two cases:

- Non graphical unit tests: these are standard unit tests that can run in a console
- Graphical unit tests: these need a GL context, and if requested, work via image comparison

To be able to run unit tests, you need to install pytest (<https://pytest.org/>), and coverage (<http://nedbatchelder.com/code/coverage/>). You can use pip for that:

```
sudo pip install pytest coverage
```

Then, in the kivy directory:

```
make test
```


How it works

All the tests are located in *kivy/tests*, and the filename starts with *test_<name>.py*. Pytest will automatically gather all the files and classes inside this folder, and use them to generate test cases.

To write a test, create a file that respects the previous naming, then start with this template:

```
import unittest

class XXXTestCase(unittest.TestCase):

    def setUp(self):
        # import class and prepare everything here.
        pass

    def test_YYY(self):
        # place your test case here
        a = 1
        self.assertEqual(a, 1)
```

Replace XXX with an appropriate name that covers your tests cases, then replace 'YYY' with the name of your test. If you have any doubts, check how the other tests have been written.

Then, to execute them, just run:

```
make test
```

If you want to execute that file only, you can run:

```
pytest kivy/tests/test_yourtestcase.py
```

or include this simple *unittest.main()* call at the end of the file and run the test with *python test_yourtestcase.py*:

```
if __name__ == '__main__':
    unittest.main()
```

Graphical unit tests

While simple unit tests are fine and useful to keep things granular, in certain cases we need to test Kivy after the GL Window is created to interact with the graphics, widgets and to test more advanced stuff such as widget, modules, various cases of input and interaction with everything that becomes available only after the Window is created and Kivy properly initialized.

These tests are executed the same way like the ordinary unit tests i.e. either with *pytest* or via *unittest.main()*.

Here are two similar examples with different approaches of running the app. In the first one you are setting up the required stuff manually and the *tearDown()* of the *GraphicUnitTest* may only attempt to clean it after you:

```
from kivy.tests.common import GraphicUnitTest

class MyTestCase(GraphicUnitTest):

    def test_runtouchapp(self):
        # non-integrated approach
        from kivy.app import runTouchApp
        from kivy.uix.button import Button
```

(continues on next page)

```

button = Button()
runTouchApp(button)

# get your Window instance safely
from kivy.base import EventLoop
EventLoop.ensure_window()
window = EventLoop.window

# your asserts
self.assertEqual(window.children[0], button)
self.assertEqual(
    window.children[0].height,
    window.height
)

```

In the second test case both `setUp()` and `tearDown()` work together with `GraphicUnitTest.render()`. This is the basic setup it does automatically:

- Window is sized to 320 x 240 px
- Only the default Config is used during the test, it's restricted with the `KIVY_USE_DEFAULTCONFIG` environment variable
- Any input (mouse/touch/...) is *removed* and if you need to test it, either mock it or manually add it
- Window's canvas is cleared before displaying any widget tree

Warning: Do NOT use absolute numbers in your tests to preserve the functionality across the all resolutions. Instead, use e.g. relative position or size and multiply it by the `Window.size` in your test.

```

from kivy.tests.common import GraphicUnitTest, UnitTestTouch

class MyTestCase(GraphicUnitTest):

    def test_render(self):
        from kivy.uix.button import Button

        # with GraphicUnitTest.render() you basically do this:
        # runTouchApp(Button()) + some setup before
        button = Button()
        self.render(button)

        # get your Window instance safely
        from kivy.base import EventLoop
        EventLoop.ensure_window()
        window = EventLoop.window

        touch = UnitTestTouch(
            *[s / 2.0 for s in window.size]
        )

        # bind something to test the touch with
        button.bind(
            on_release=lambda instance: setattr(
                instance, 'test_released', True
            )
        )

```

(continues on next page)

```

        )
    )

    # then let's touch the Window's center
    touch.touch_down()
    touch.touch_up()
    self.assertTrue(button.test_released)

if __name__ == '__main__':
    import unittest
    unittest.main()

```

Note: Make sure you check the source of *kivy.tests.common* before writing comprehensive test cases.

GL unit tests

GL unit test are more difficult. You must know that even if OpenGL is a standard, the output/rendering is not. It depends on your GPU and the driver used. For these tests, the goal is to save the output of the rendering at frame X, and compare it to a reference image.

Currently, images are generated at 320x240 pixels, in *png* format.

Note: Currently, image comparison is done per-pixel. This means the reference image that you generate will only be correct for your GPU/driver. If somebody can implement image comparison with “delta” support, patches are welcome :)

To execute GL unit tests, you need to create a directory:

```

mkdir kivy/tests/results
KIVY_UNITTEST_SCREENSHOTS=1 make test

```

The results directory will contain all the reference images and the generated images. After the first execution, if the results directory is empty, no comparison will be done. It will use the generated images as reference. After the second execution, all the images will be compared to the reference images.

A html file is available to show the comparison before/after the test, and a snippet of the associated unit test. It will be generated at:

```
kivy/tests/build/index.html
```

Note: The build directory is cleaned after each call to *make test*. If you don’t want that, just use *pytest* command.

Writing GL Unit tests

The idea is to create a root widget, as you would do in *build()*, or in *kivy.base.runTouchApp()*. You’ll give that root widget to a rendering function which will capture the output in X frames.

Here is an example:

```

from kivy.tests.common import GraphicUnitTest

class VertexInstructionTestCase(GraphicUnitTest):

    def test_ellipse(self):
        from kivy.uix.widget import Widget
        from kivy.graphics import Ellipse, Color
        r = self.render

        # create a root widget
        wid = Widget()

        # put some graphics instruction on it
        with wid.canvas:
            Color(1, 1, 1)
            self.e = Ellipse(pos=(100, 100), size=(200, 100))

        # render, and capture it directly
        r(wid)

        # as alternative, you can capture in 2 frames:
        r(wid, 2)

        # or in 10 frames
        r(wid, 10)

```

Each call to *self.render* (or *r* in our example) will generate an image named as follows:

```
<classname>_<funcname>-<r-call-count>.png
```

r-call-count represents the number of times that *self.render* is called inside the test function.

The reference images are named:

```
ref_<classname>_<funcname>-<r-call-count>.png
```

You can easily replace the reference image with a new one if you wish.

Coverage reports

Coverage is based on the execution of previous tests. Statistics on code coverage are automatically calculated during execution. You can generate an html report of the coverage with the command:

```
make cover
```

Then, open *kivy/htmlcov/index.html* with your favorite web browser.

2.8 GSOC

2.8.1 Google Summer of Code - 2017

Introduction

Kivy is a cross-platform, business friendly, GPU accelerated open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps.

The Kivy Organization oversees several major projects:

- The **Kivy** GUI Library
- The **Python-For-Android** compilation tool.
- The **Kivy-iOS** compilation tool.
- The **PyJNIus** library for interfacing with Java from Python.
- The **PyOBJus** library for interfacing with Objective-C from Python.
- The **Plyer** platform-independent Python wrapper for platform dependent APIs.
- **Buildozer** - A generic Python packager for Android, iOS, and desktop.
- **KivEnt** - A 2d Game Engine that provides optimized methods of handling large amounts of dynamic visual data.
- **Kivy Designer** - A graphical GUI designer for Kivy built in Kivy.

Altogether, these projects allow the user to create applications for every major operating system that make use of any native APIs present. Our goal is to enable development of Python applications that run everywhere off the same codebase and make use of platform dependent APIs and features that users of specific operating systems have come to expect.

Depending on which project you choose you may need to know Cython, OpenGL ES2, Java, Objective-C, or C in addition to Python. We make heavy use of Cython and OpenGL for computational and graphics performance where it matters, and the other languages are typically involved in accessing OS or provider level APIs.

We are hoping to participate in Google Summer of Code 2017. This page showcases some ideas for GSoC projects and corresponding guidelines for students contributing to the Kivy Framework.

Requirements

It is assumed that the incoming student meets some basic requirements as highlighted here:

- Intermediate level familiarity with Python.
- Comfortable with git and github (Kivy and its sister projects are all managed on github) If you have never used github before you may be interested in this [tutorial](#).
- Comfortable with event driven programming.
- Has suitable tools/environment for Kivy or the sister project you are going to work on. For example to be able to work on PyOBJus you would need access to an iOS device, OS X with Xcode and a developer license, to work on PyJNIus you would need an Android device, and to work on plyer you would need access to hardware for both platforms.

Additional desired skills may be listed with specific projects.

Familiarize yourself with the [contribution guide](#) We can help you get up to speed, however students demonstrating ability in advance will be given preference.

How to get started

For Kivy, the easiest way is to follow the installation instructions for the development version for your specific platform:

<http://kivy.org/docs/installation/installation.html#development-version>

For the rest it's usually sufficient to install the relevant project from git and add it to your PYTHON-PATH.

e.g. for PyJNIus:

```
git clone http://github.com/kivy/pyjnius
export PYTHONPATH=/path/to/pyjnius:$PYTHONPATH
```

Project Ideas

Here are some prospective ideas sourced from the Kivy development team, if none of these projects interest you come talk to us in #kivy-dev about a project idea of your own.

Beginner Projects

These projects should be suitable for anyone with a college level familiarity with Python and require little knowledge of platform specifics.

Intermediate Projects

These projects may involve cursory level knowledge of several OS level details, some OpenGL interaction, or other topics that may be a bit out of the wheelhouse of the average Pythonista.

Plyer:

Description: Plyer is a platform-independent Python API to use features commonly found on the desktop and mobile platforms supported by Kivy. The idea is to provide a stable API to the user for accessing features of their desktop or mobile device.

The student would replace some *.java* code currently in the p4a project to a more appropriate place in Plyer. In addition, the student would work on improving access to platform specific features through Plyer, including accessibility, Bluetooth Low Energy, accessing and editing contacts, sharing, NFC, in-app browser, Wi-Fi (enable, disable, access to Wi-Fi services (Wi-Fi direct, network accessibility, current IP info on network etc.), Camera capture (video), camera display, Google Play integration, launch phone call interface, sms interface, geolocation, interaction with notifications, internationalization (I18N), and all the missing platform implementations from existing features.

Under the hood you'll use PyJNIus on Android, PyObjus on OS X and iOS, ctypes on Windows, and native APIs on Linux. This probably would also include improving PyObjus and PyJNIus to handle interfaces that they can't right now.

References:

- <https://github.com/kivy/plyer>
- <https://github.com/kivy/pyjnius>
- <https://github.com/kivy/pyobjus>
- <https://github.com/kivy/python-for-android>
- <https://github.com/kivy/kivy-ios>

Expected outcome: A successful outcome would include moving the Java/PyObjus code from p4a/kivy-ios to plyer and implementing some or all of the new facades to be decided with the student.

- **Mentors:** Akshay Arora
- **Requirements:** Access to Linux, Windows, OS X, iOS device, Android device.
- **Task level:** Intermediate
- **Desired Skills:** Familiarity with PyJNIus, PyObjus.

Font Reshaping and Font Fallback Support

Description: Currently Kivy does not support reshaping for alphabets such as Arabic, Persian, Thai, or Devanagari. The solution is to integrate a text shaping and layout engine (Pango and Harfbuzz). You would need to ensure that Pango and Harfbuzz can be compiled on every platform, and integrate it as a core text provider.

The second part of the same project would involve font fallback support. If a particular character/glyph is missing, currently we show a [] box. The solution for this would involve either using an OS API if available or maintaining a hashtable for the default fonts on each OS which can be used for glyph fallback.

References:

- <http://www.pango.org>
- <https://www.freedesktop.org/wiki/Software/HarfBuzz/>
- <https://github.com/kivy/kivy/tree/master/kivy/core/text>

Expected outcome: Font fallback and text reshaping support in Kivy, compilation recipes for Python-For-Android and packaging on desktop platforms.

- **Mentors:** Akshay Arora, Jacob Kovac, Matthew Einhorn
- **Requirements:** Access to a desktop OS and ideally at least one mobile platform
- **Task level:** Intermediate
- **Desired Skills:** Familiarity with text rendering, Pango, HarfBuzz and Kivy's provider abstraction.

Advanced Projects

These projects may involve very in-depth knowledge of Kivy's existing internals, the hairy details of cross-platform compilation, or other fairly advanced topics. If you are comfortable with the internals of Python, working with C code, and using Cython to build your own C extensions these projects may appeal to you.

Kivent: Chipmunk 7 Integration

Description: KivEnt is a modular entity-component based game engine built on top of Kivy. KivEnt provides a highly performant approach to building games in Python that avoids some of the worst overhead of Python using specialized Cython constructs.

At the moment, KivEnt internally makes use of the cymunk library (<https://github.com/tito/cymunk>) for physics simulation and collision detection. Cymunk is based on Chipmunk2d 6.x, recently Chipmunk 7 has released and brought many previously premium features into the core library. In addition to the API changes present in the newest Chipmunk, the KivEnt - Cymunk bridging does not make most efficient use of the KivEnt API for handling C level objects and data. The student will be responsible for creating a new wrapper over Chipmunk2d 7 that better matches KivEnt's approach to handling game data.

References:

- <http://chipmunk-physics.net/>
- <https://github.com/kivy/kivent>

Expected Outcome: A successful outcome involves a new kivent_tiled module being released for the KivEnt game engine.

- **Mentors:** Jacob Kovac

- **Requirements:** Access to at least one Kivy platform.
- **Task level:** Advanced
- **Desired Skills:** Familiarity with Cython, Python, and game dev related math concepts.

KV Compiler: A compiler for the KV language

Description: The KV language is a fundamental component of Kivy. The KV language allows one to describe a GUI; from the creation of a Widget tree to the actions that should be taken in response value changes and events. In effect it is a concise way to create rule bindings using the Kivy properties and events. Internally, python code that reflects these rules are created and bound to the properties and events. Currently, these bindings are not at all optimized because upon each widget creation all of these rules are re-evaluated and bound. This process can be significantly optimized by pre-compiling the kv code, especially the bindings. A compiler would also allow us to update and fix some of the long-standing kv language issues.

Work on a kv-compiler has already progressed quite far, in fact a PR in the pre-alpha stage, is currently open. However, it is out of sync with the current codebase due to some unrelated kv changes in the meantime. Also, that PR would require a significant re-write to make things more modular, self-contained, and extensible. So there is much work still to be done on it.

Theming has also been a prepatual issue in Kivy, a KV compiler may help implement bindings that facilitate theming.

References:

- <https://kivy.org/docs/guide/lang.html>
- <https://github.com/kivy/kivy/pull/3456>
- <https://github.com/kivy/kivy/wiki/KEP001:-Instantiate-things-other-than-widgets-from-kv>
- <https://github.com/kivy/kivy/issues/691>
- <https://github.com/kivy/kivy/issues/2727>

Expected Outcome: A successful outcome would be a compiler which compiles kv code into python code. The compiler should be modular and extensible so that we can continue to improve the kv language. The compiler should have the common debug/optimization options. The compiled code should also be human readable so issues could be traced back to the original kv code. The compiler should also be a drop in replacement for the current KV runtime compiler, and would require extensive testing.

- **Mentors:** Matthew Einhorn
- **Requirements:** Access to at least one Kivy platform.
- **Task level:** Advanced
- **Desired Skills:** Familiarity with Cython, Python, and Kivy. Familiarity with typical computer science concepts and data structures is also desired.

How to Contact devs

All communication must happen via public channels, private emails and Discord private messages are discouraged.

Ask your questions on the Kivy Users forum <https://groups.google.com/group/kivy-users> or send a mail at kivy-users@googlegroups.com

Make sure to join the kivy-dev user group too: <https://groups.google.com/forum/#!forum/kivy-dev>.

You can also try to contact us on Discord, to get the Discord handles of the devs mentioned above visit <https://kivy.org/#aboutus>.

Make sure to read the [Discord rules](#) before connecting. [Connect to Discord](#).

Most of our developers are located in Europe, India, and North America so keep in mind typical waking hours for these areas.

How to be a good student

If you want to participate as a student and want to maximize your chances of being accepted, start talking to us today and try fixing some smaller problems to get used to our workflow. If we know you can work well with us, you will have much better chances of being selected.

Here's a checklist:

- Make sure to read through the website and at least skim the documentation.
- Look at the source code.
- Read our contribution guidelines.
- Make a contribution! Kivy would like to see how you engage with the development process. Take a look at the issue tracker for a Kivy project that interests you and submit a Pull Request. It can be a simple bug or a documentation change. We are looking to get a feel for how you work, not evaluating your capabilities. Don't worry about trying to pick something to impress us.
- Pick an idea that you think is interesting from the ideas list or come up with your own idea.
- Do some research **yourself**. GSoC is about give and take, not just one sided interaction. It is about you trying to achieve agreed upon goals with our support. The main driving force in this should be, obviously, yourself. Many students pop up and ask what they should do. You should base that decision on your interests and your skills. Show us you're serious about it and take the initiative.
- Write a draft [proposal](#) about what you want to do. Include what you understand the current state of the project to be, what you would like to improve, how, etc.
- Discuss that proposal with us in a timely manner. Get feedback.
- Be patient! Especially on Discord. We will try to get to you if we're available. If not, send an email and just wait. Most questions are already answered in the docs or somewhere else and can be found with some research. Your questions should reflect that you've actually thought through what you're asking and done some rudimentary research.
- Most of all don't forget to have fun and interact with the community. The community is as big a part of Open Source as the code itself.

What to expect if you are chosen

- All students should join the #support and the #dev Discord channels daily, this is how the development team communicates both internally and with the users.
- You and your mentors will agree on two week milestones for the duration of the summer.
- Development will occur in your fork of the master branch of Kivy, we expect you to submit at least one PR a week from your branch into a branch reserved for you in the primary repo. This will be your forum for reporting progress as well as documenting any struggles you may have encountered.
- Missing 2 weekly PR or 2 milestones will result in your failure unless there have been extenuating circumstances. If something comes up, please inform your mentors as soon as possible. If a milestone seems out of reach we will work with you to reevaluate the goals.

- Your changes will be merged into master once the project has been completed and we have thoroughly tested on every platform that is relevant.

FAQ

There are a number of questions that repeatedly need to be answered. The following document tries to answer some of them.

3.1 Technical FAQ

3.1.1 Unable to get a Window, abort.

If Kivy cannot instantiate a Window core provider (mostly SDL2), you'll see this. The underlying issue depends on many things:

- Check your installation. Twice.
- Check that your graphics driver support OpenGL 2.1 at the minimum. Otherwise, Kivy can't run.
- If you use windows and ANGLE (`KIVY_GL_BACKEND=angle_sdl2`), check that you have DirectX 9 support.
- If your platform doesn't supports OpenGL, SDL2 cannot initialize OpenGL.
- Don't mix the architecture of the dependencies (e.g. Python 64-bit and 32-bit extensions/SDL2)
- Don't mix python installation: e.g. if you have Python and Anaconda installed, the Python actually run may be different than you think. Similarly, if you have multiple Python versions available on the `PATH`, they may clash.
- Check your `PATH` to ensure that other programs in it don't provide the same dlls as Kivy/Python, or bad stuff can happen.
 - This commonly happens if some other program that uses similar dependencies as Kivy adds itself to the `PATH` so that Kivy's dependencies clash with theirs.
 - Please read [this](#) and [this](#) for more details on `PATH`.
 - The best tool to troubleshoot this is with [Dependency Walker](#) explained [here](#) and [here](#).
 - But ensure that you're launching it from the identical environment that you start Python.

- Ensure you have all dependencies installed (like `kivy_deps.sdl2`).
- Maybe your drivers have some missing OpenGL symbols? Try to switch to another graphics backend with `KIVY_GL_BACKEND`.
- Maybe your [Pycharm configuration is incorrect](#).

3.1.2 Fatal Python error: (pygame parachute) Segmentation Fault

Most of time, this issue is due to the usage of old graphics drivers. Install the latest graphics driver available for your graphics card, and it should be ok.

If not, this means you have probably triggered some OpenGL code without an available OpenGL context. If you are loading images, atlases, using graphics instructions, you must spawn a Window first:

```
# method 1 (preferred)
from kivy.base import EventLoop
EventLoop.ensure_window()

# method 2
from kivy.core.window import Window
```

If not, please report a detailed issue on github by following the instructions in the [Reporting an Issue](#) section of the [Contributing](#) documentation. This is very important for us because that kind of error can be very hard to debug. Give us all the information you can give about your environment and execution.

3.1.3 undefined symbol: glGenerateMipmap

Your graphics card or its drivers might be too old. Update your graphics drivers to the latest available version and retry.

3.1.4 ImportError: No module named event

If you use Kivy from our development version, you must compile it before using it. In the kivy directory, do:

```
make force
```

3.2 Android FAQ

3.2.1 could not extract public data

This error message can occur under various circumstances. Ensure that:

- you have a phone with an sdcard
- you are not currently in “USB Mass Storage” mode
- you have permissions to write to the sdcard

In the case of the “USB Mass Storage” mode error, and if you don’t want to keep unplugging the device, set the usb option to Power.

3.2.2 Crash on touch interaction on Android 2.3.x

There have been reports of crashes on Adreno 200/205 based devices. Apps otherwise run fine but crash when interacted with/through the screen.

These reports also mentioned the issue being resolved when moving to an ICS or higher ROM.

3.2.3 Is it possible to have a kiosk app on android 3.0 ?

Thomas Hansen have wrote a detailed answer on the kivy-users mailing list:

https://groups.google.com/d/msg/kivy-users/QKoCekAR1c0/yV-85Y_iAwoJ

Basically, you need to root the device, remove the SystemUI package, add some lines to the xml configuration, and you're done.

3.2.4 What's the difference between python-for-android from Kivy and SL4A?

Despite having the same name, Kivy's python-for-android is not related to the python-for-android project from SL4A, Py4A, or android-python27. They are distinctly different projects with different goals. You may be able to use Py4A with Kivy, but no code or effort has been made to do so. The Kivy team feels that our python-for-android is the best solution for us going forward, and attempts to integrate with and support Py4A is not a good use of our time.

3.3 Project FAQ

3.3.1 Why do you use Python? Isn't it slow?

Let us try to give a thorough answer; please bear with us.

Python is a very agile language that allows you to do many things in a (by comparison) short time. For many development scenarios, we strongly prefer writing our application quickly in a high-level language such as Python, testing it, then optionally optimizing it.

But what about speed? If you compare execution speeds of implementations for a certain set of algorithms (esp. number crunching) you will find that Python is a lot slower than say, C++. Now you may be even more convinced that it's not a good idea in our case to use Python. Drawing sophisticated graphics (and we are not talking about your grandmother's OpenGL here) is computationally quite expensive and given that we often want to do that for rich user experiences, that would be a fair argument. **But**, in virtually every case your application ends up spending most of the time (by far) executing the same part of the code. In Kivy, for example, these parts are event dispatching and graphics drawing. Now Python allows you to do something to make these parts much faster.

By using Cython, you can compile your code down to the C level, and from there your usual C compiler optimizes things. This is a pretty pain free process and if you add some hints to your code, the result becomes even faster. We are talking about a speed up in performance by a factor of anything between 1x and up to more than 1000x (greatly depends on your code). In Kivy, we did this for you and implemented the portions of our code, where efficiency really is critical, on the C level.

For graphics drawing, we also leverage today's GPUs which are, for some tasks such as graphics rasterization, much more efficient than a CPU. Kivy does as much as is reasonable on the GPU to maximize performance. If you use our Canvas API to do the drawing, there is even a compiler that we invented which optimizes your drawing code automatically. If you keep your drawing mostly on the GPU, much of your program's execution speed is not determined by the programming language used, but by the graphics hardware you throw at it.

We believe that these (and other) optimizations that Kivy does for you already make most applications fast enough by far. Often you will even want to limit the speed of the application in order not to waste resources. But even if this is not sufficient, you still have the option of using Cython for your own code to *greatly* speed it up.

Trust us when we say that we have given this very careful thought. We have performed many different benchmarks and come up with some clever optimizations to make your application run smoothly.

3.3.2 Does Kivy support Python 3.x?

Yes! As of version 1.8.0 Kivy supports both Python ≥ 2.7 and Python ≥ 3.4 with the same codebase. Python 3 is also now supported by python-for-android.

However, be aware that while Kivy will run in Python 3.4+, our iOS build tools still require Python 2.7.

3.3.3 How is Kivy related to PyMT?

Our developers are professionals and are pretty savvy in their area of expertise. However, before Kivy came around there was (and still is) a project named PyMT that was led by our core developers. We learned a great deal from that project during the time that we developed it. In the more than two years of research and development we found many interesting ways to improve the design of our framework. We have performed numerous benchmarks and as it turns out, to achieve the great speed and flexibility that Kivy has, we had to rewrite quite a big portion of the codebase, making this a backwards-incompatible but future-proof decision. Most notable are the performance increases, which are just incredible. Kivy starts and operates just so much faster, due to these heavy optimizations. We also had the opportunity to work with businesses and associations using PyMT. We were able to test our product on a large diversity of setups and made PyMT work on all of them. Writing a system such as Kivy or PyMT is one thing. Making it work under all these different conditions is another. We have a good background here, and brought our knowledge to Kivy.

Furthermore, since some of our core developers decided to drop their full-time jobs and turn to this project completely, it was decided that a more professional foundation had to be laid. Kivy is that foundation. It is supposed to be a stable and professional product. Technically, Kivy is not really a successor to PyMT because there is no easy migration path between them. However, the goal is the same: Producing high-quality applications for novel user interfaces. This is why we encourage everyone to base new projects on Kivy instead of PyMT. Active development of PyMT has stalled. Maintenance patches are still accepted.

3.3.4 Do you accept patches?

Yes, we love patches. In order to ensure a smooth integration of your precious changes however, please make sure to read our contribution guidelines. Obviously we don't accept every patch. Your patch has to be consistent with our styleguide and, more importantly, make sense. It does make sense to talk to us before you come up with bigger changes, especially new features.

3.3.5 Does the Kivy project participate in Google's Summer of Code ?

Potential students ask whether we participate in GSoC. The clear answer is: Indeed. :-)

If you want to participate as a student and want to maximize your chances of being accepted, start talking to us today and try fixing some smaller (or larger, if you can ;-)) problems to get used to our workflow. If we know you can work well with us, that'd be a big plus.

Here's a checklist:

- Make sure to read through the website and at least skim the documentation.

- Look at the source code.
- Read our contribution guidelines.
- Pick an idea that you think is interesting from the ideas list (see link above) or come up with your own idea.
- Do some research **yourself**. GSoC is not about us teaching you something and you getting paid for that. It is about you trying to achieve agreed upon goals by yourself with our support. The main driving force in this should be, obviously, yourself. Many students come up and ask what they should do. Well, we don't know because we know neither your interests nor your skills. Show us you're serious about it and take initiative.
- Write a draft proposal about what you want to do. Include what you understand the current state is (very roughly), what you would like to improve and how, etc.
- Discuss that proposal with us in a timely manner. Get feedback.
- Be patient! Especially on Discord. We will try to get to you if we're available. If not, send an email and just wait. Most questions are already answered in the docs or somewhere else and can be found with some research. If your questions don't reflect that you've actually thought through what you're asking, it might not be well received.

Good luck! :-)

Contact Us

You can contact us in several different ways:

4.1 Issue Tracker

If you have found an issue with the code or have a feature request, please see our [issue tracker](#). If there is no issue yet that matches your inquiry, feel free to create a new one. Please make sure you receive the mails that github sends if we comment on the issue in case we need more information. For bugs, please provide all the information necessary, like the operating system you're using, the **full error message** or any other logs, a description of what you did to trigger the bug and what the actual bug was, as well as anything else that might be of interest. Obviously, we can only help if you tell us precisely what the actual problem is.

4.2 Mail

For users of our framework, there is a mailing list for support inquiries on the [kivy-users Google Group](#). Use this list if you have issues with your Kivy-based app. We also have a mailing list for matters that deal with development of the actual Kivy framework code on the [kivy-dev Google Group](#).

4.3 Discord

Kivy on Discord at <https://chat.kivy.org>

Discord is great for real-time communication, but please make sure to **wait** after you asked your question. If you just join, ask and quit we have **no way** of knowing who you were and where we're supposed to send our answer. Also, keep in mind we're mostly based in Europe, so take into account any timezone issues. If you're unlucky more than once, try the mailing list.

If you don't have the Discord app, you can also use [Discord's web client](#), but please, don't leave too soon. Just make sure to ask on the **#support** channel.

Please read our [Community Guidelines](#) before asking for help on the mailing list or Discord channel.

Part II

PROGRAMMING GUIDE

Kivy Basics

5.1 Installation of the Kivy environment

Kivy depends on many Python libraries, such as pygame, gstreamer, PIL, Cairo, and more. They are not all required, but depending on the platform you're working on, they can be a pain to install. For Windows and MacOS X, we provide a portable package that you can just unzip and use.

Have a look at one of these pages for detailed installation instructions:

- [installation_windows](#)
- [installation_osx](#)
- [installation_linux](#)
- [installation_rpi](#)

Alternatively, instructions for the development version can be found [here](#):

- [installation](#)

5.2 Create an application

Creating a kivy application is as simple as:

- sub-classing the **App** class
- implementing its **build()** method so it returns a **Widget** instance (the root of your widget tree)
- instantiating this class, and calling its **run()** method.

Here is an example of a minimal application:

```
import kivy
kivy.require('1.0.6') # replace with your current kivy version !
```

(continues on next page)

(continued from previous page)

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

class MyApp(App):

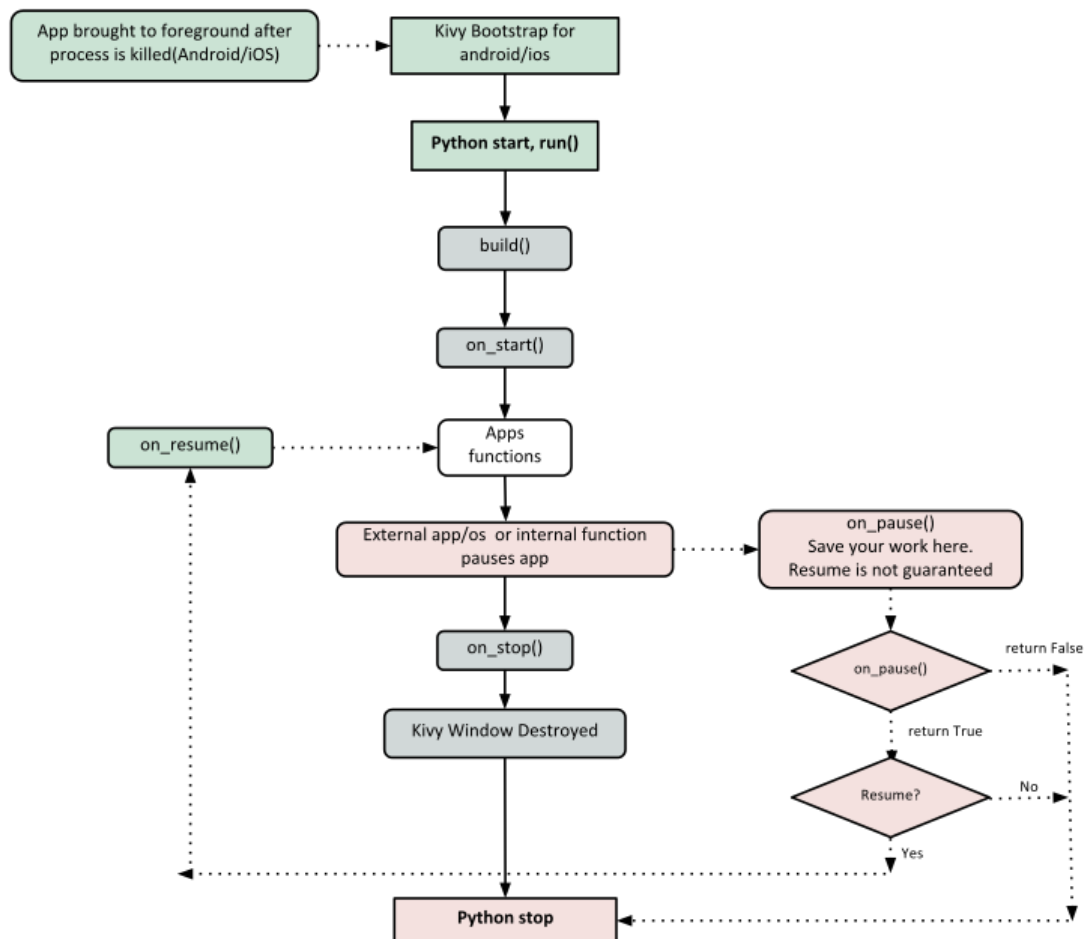
    def build(self):
        return Label(text='Hello world')

if __name__ == '__main__':
    MyApp().run()
```

You can save this to a text file, *main.py* for example, and run it.

5.3 Kivy App Life Cycle

First off, let's get familiar with the Kivy app life cycle.



As you can see above, for all intents and purposes, our entry point into our App is the run() method, and in our case that is "MyApp().run()". We will get back to this, but let's start from the third line:

```
from kivy.app import App
```

It's required that the base Class of your App inherits from the *App* class. It's present in the

kivy_installation_dir/kivy/app.py.

Note: Go ahead and open up that file if you want to delve deeper into what the Kivy App class does. We encourage you to open the code and read through it. Kivy is based on Python and uses Sphinx for documentation, so the documentation for each class is in the actual file.

Similarly on line 2:

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

One important thing to note here is the way packages/classes are laid out. The *uix* module is the section that holds the user interface elements like layouts and widgets.

Moving on to line 5:

```
class MyApp(App):
```

This is where we are *defining* the Base Class of our Kivy App. You should only ever need to change the name of your app *MyApp* in this line.

Further on to line 7:

```
def build(self):
```

As highlighted by the image above, showing casing the *Kivy App Life Cycle*, this is the function where you should initialize and return your *Root Widget*. This is what we do on line 8:

```
return Label(text='Hello world')
```

Here we initialize a Label with text 'Hello World' and return its instance. This Label will be the Root Widget of this App.

Note: Python uses indentation to denote code blocks, therefore take note that in the code provided above, at line 9 the class and function definition ends.

Now on to the portion that will make our app run at line 11 and 12:

```
if __name__ == '__main__':  
    MyApp().run()
```

Here the class *MyApp* is initialized and its *run()* method called. This initializes and starts our Kivy application.

5.4 Running the application

To run the application, follow the instructions for your operating system:

Linux Follow the instructions for running a Kivy application on Linux:

```
$ python main.py
```

Windows Follow the instructions for running a Kivy application on Windows:

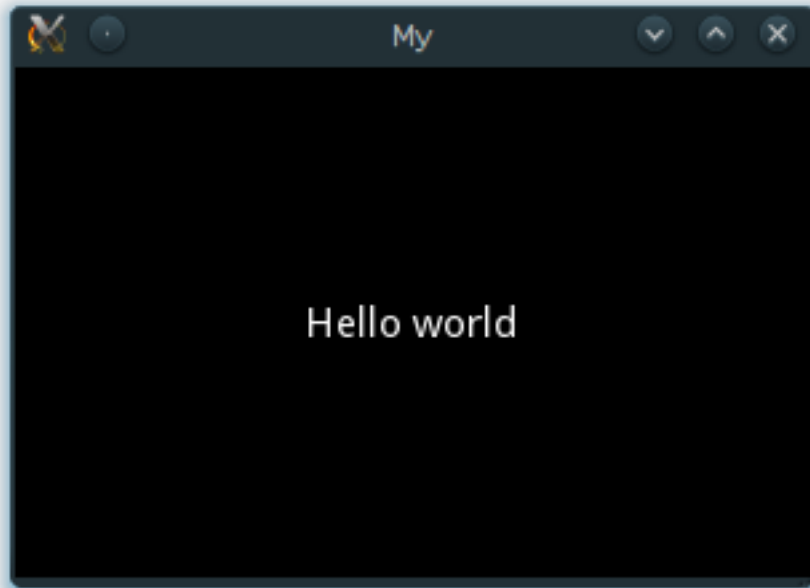
```
$ python main.py  
# or  
C:\appdir>kivy.bat main.py
```

Mac OS X Follow the instructions for running a Kivy application on OS X:

```
$ kivy main.py
```

Android Your application needs some complementary files to be able to run on Android.
See [Create a package for Android](#) for further reference.

A window should open, showing a single Label (with the Text 'Hello World') that covers the entire window's area. That's all there is to it.



5.5 Customize the application

Lets extend this application a bit, say a simple UserName/Password page.

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput

class LoginScreen(GridLayout):

    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.cols = 2
        self.add_widget(Label(text='User Name'))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
        self.add_widget(Label(text='password'))
        self.password = TextInput(password=True, multiline=False)
        self.add_widget(self.password)
```

(continues on next page)


```
class MyApp(App):
    def build(self):
        return LoginScreen()

if __name__ == '__main__':
    MyApp().run()
```

At line 2 we import a GridLayout:

```
from kivy.uix.gridlayout import GridLayout
```

This class is used as a Base for our Root Widget (LoginScreen) defined at line 9:

```
class LoginScreen(GridLayout):
```

At line 12 in the class LoginScreen, we override the method `__init__()` so as to add widgets and to define their behavior:

```
def __init__(self, **kwargs):
    super(LoginScreen, self).__init__(**kwargs)
```

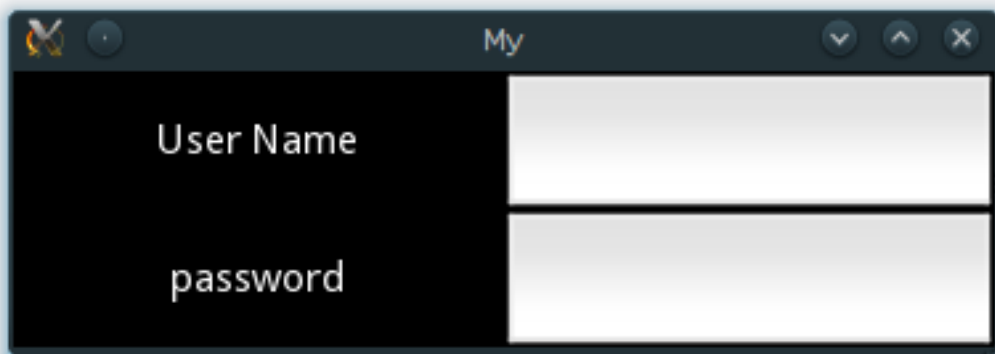
One should not forget to call `super` in order to implement the functionality of the original class being overloaded. Also note that it is good practice not to omit the `**kwargs` while calling `super`, as they are sometimes used internally.

Moving on to Line 15 and beyond:

```
self.cols = 2
self.add_widget(Label(text='User Name'))
self.username = TextInput(multiline=False)
self.add_widget(self.username)
self.add_widget(Label(text='password'))
self.password = TextInput(password=True, multiline=False)
self.add_widget(self.password)
```

We ask the GridLayout to manage its children in two columns and add a *Label* and a *TextInput* for the username and password.

Running the above code will give you a window that should look like this:



Try re-sizing the window and you will see that the widgets on screen adjust themselves according to the size of the window without you having to do anything. This is because widgets use size hinting by default.

The code above doesn't handle the input from the user, does no validation or anything else. We will delve deeper into this and `Widget` size and positioning in the coming sections.

Controlling the environment

Many environment variables are available to control the initialization and behavior of Kivy.

For example, in order to restrict text rendering to the PIL implementation:

```
$ KIVY_TEXT=pil python main.py
```

Environment variables should be set before importing kivy:

```
import os
os.environ['KIVY_TEXT'] = 'pil'
import kivy
```

6.1 Path control

New in version 1.0.7.

You can control the default directories where config files, modules and kivy data are located.

KIVY_DATA_DIR Location of the Kivy data, defaults to *<kivy path>/data*

KIVY_MODULES_DIR Location of the Kivy modules, defaults to *<kivy path>/modules*

KIVY_HOME Location of the Kivy home. This directory is used for local configuration, and must be in a writable location.

Defaults to:

- Desktop: *<user home>/kivy*
- Android: *<android app path>/kivy*
- iOS: *<user home>/Documents/kivy*

New in version 1.9.0.

KIVY_SDL2_PATH If set, the SDL2 libraries and headers from this path are used when compiling kivy instead of the ones installed system-wide. To use the same libraries while running a kivy app, this path must be added at the start of the PATH environment variable.

New in version 1.9.0.

Warning: This path is required for the compilation of Kivy. It is not required for program execution.

6.2 Configuration

KIVY_USE_DEFAULTCONFIG If this name is found in environ, Kivy will not read the user config file.

KIVY_NO_CONFIG If set, no configuration file will be read or written to. This also applies to the user configuration directory.

KIVY_NO_FILELOG If set, logs will be not print to a file

KIVY_NO_CONSOLELOG If set, logs will be not print to the console

KIVY_NO_ARGS If set, the argument passed in command line will not be parsed and used by Kivy. Ie, you can safely make a script or an app with your own arguments without requiring the – delimiter:

```
import os
os.environ["KIVY_NO_ARGS"] = "1"
import kivy
```

New in version 1.9.0.

KCFG_section_key If a such format environment name is detected, it will be mapped to the Config object. They are loaded only once when *kivy* is imported. The behavior can be disabled using **KIVY_NO_ENV_CONFIG**.

```
import os
os.environ["KCFG_KIVY_LOG_LEVEL"] = "warning"
import kivy
# during import it will map it to:
# Config.set("kivy", "log_level", "warning")
```

New in version 1.11.0.

KIVY_NO_ENV_CONFIG If set, no environment key will be mapped to configuration object. If unset, any *KCFG_section_key=value* will be mapped to Config.

New in version 1.11.0.

6.3 Restrict core to specific implementation

kivy.core try to select the best implementation available for your platform. For testing or custom installation, you might want to restrict the selector to a specific implementation.

KIVY_WINDOW Implementation to use for creating the Window

Values: sdl2, pygame, x11, egl_rpi

KIVY_TEXT Implementation to use for rendering text

Values: sdl2, pil, pygame, sdlttf

KIVY_VIDEO Implementation to use for rendering video

Values: gstplayer, ffpyplayer, ffmpeg, null

KIVY_AUDIO Implementation to use for playing audio

Values: sdl2, gstplayer, ffpyplayer, pygame, avplayer

KIVY_IMAGE Implementation to use for reading image

Values: sdl2, pil, pygame, imageio, tex, dds, gif

KIVY_CAMERA Implementation to use for reading camera

Values: avfoundation, android, opencv

KIVY_SPELLING Implementation to use for spelling

Values: enchant, osxappkit

KIVY_CLIPBOARD Implementation to use for clipboard management

Values: sdl2, pygame, dummy, android

6.4 Metrics

KIVY_DPI If set, the value will be used for `Metrics.dpi`.

New in version 1.4.0.

KIVY_METRICS_DENSITY If set, the value will be used for `Metrics.density`.

New in version 1.5.0.

KIVY_METRICS_FONTSCALE

If set, the value will be used for `Metrics.fontscale`.

New in version 1.5.0.

6.5 Graphics

KIVY_GL_BACKEND The OpenGL backend to use. See [cgl](#).

KIVY_GL_DEBUG Whether to log OpenGL calls. See [cgl](#).

KIVY_GRAPHICS Whether to use OpenGL ES2. See [cgl](#).

KIVY_GLES_LIMITS Whether the GLES2 restrictions are enforced (the default, or if set to 1). If set to false, Kivy will not be truly GLES2 compatible.

Following is a list of the potential incompatibilities that result when set to true.

Mesh in-dices	If true, the number of indices in a mesh is limited to 65535
Texture blit	When blitting to a texture, the data (color and buffer) format must be the same format as the one used at the texture creation. On desktop, the conversion of different color is correctly handled by the driver, while on Android, most of devices fail to do it. Ref: https://github.com/kivy/kivy/issues/1600

New in version 1.8.1.

KIVY_BCM_DISPMANX_ID Change the default Raspberry Pi display to use. The list of available value is accessible in *vc_dispmanx_types.h*. Default value is 0:

- 0: DISPMANX_ID_MAIN_LCD
- 1: DISPMANX_ID_AUX_LCD
- 2: DISPMANX_ID_HDMI
- 3: DISPMANX_ID_SDTV
- 4: DISPMANX_ID_FORCE_LCD
- 5: DISPMANX_ID_FORCE_TV
- 6: DISPMANX_ID_FORCE_OTHER

KIVY_BCM_DISPMANX_LAYER Change the default Raspberry Pi dispmanx layer. Default value is 0.

New in version 1.10.1.

6.6 Event Loop

KIVY_EVENTLOOP Which async library should be used when the app is run in an asynchronous manner. See *kivy.app* for example usage.

'asyncio': When the app is run in an asynchronous manner and the **standard** library `asyncio` package should be used. The default if not set.

'trio': When the app is run in an asynchronous manner and the *trio* package should be used.

New in version 2.0.0.

Configure Kivy

The configuration file for kivy is named *config.ini*, and adheres to the **standard INI** format.

7.1 Locating the configuration file

The location of the configuration file is controlled by the environment variable *KIVY_HOME*:

```
<KIVY_HOME>/config.ini
```

On desktop, this defaults to:

```
<HOME_DIRECTORY>/ .kivy/config.ini
```

Therefore, if your user is named “tito”, the file will be here:

- Windows: C:\Users\tito\.kivy\config.ini
- OS X: /Users/tito/.kivy/config.ini
- Linux: /home/tito/.kivy/config.ini

On Android, this defaults to:

```
<ANDROID_APP_PATH>/ .kivy/config.ini
```

If your app is named “org.kivy.launcher”, the file will be here:

```
/data/data/org.kivy.launcher/files/.kivy/config.ini
```

On iOS, this defaults to:

```
<HOME_DIRECTORY>/Documents/.kivy/config.ini
```

7.2 Local configuration

Sometimes it's desired to change configuration only for certain applications or during testing of a separate part of Kivy for example input providers. To create a separate configuration file you can simply use these commands:

```
from kivy.config import Config

Config.read(<file>)
# set config
Config.write()
```

When a local configuration of single `.ini` file isn't enough, e.g. when you want to have separate environment for *garden*, kivy logs and other things, you'll need to change the `KIVY_HOME` environment variable in your application to get desired result:

```
import os
os.environ['KIVY_HOME'] = <folder>
```

or before each run of the application change it manually in the console:

1. Windows:

```
set KIVY_HOME=<folder>
```

2. Linux & OSX:

```
export KIVY_HOME=<folder>
```

After the change of `KIVY_HOME`, the folder will behave exactly the same as the default `.kivy/` folder mentioned above.

7.3 Understanding config tokens

All the configuration tokens are explained in the *kivy.config* module.

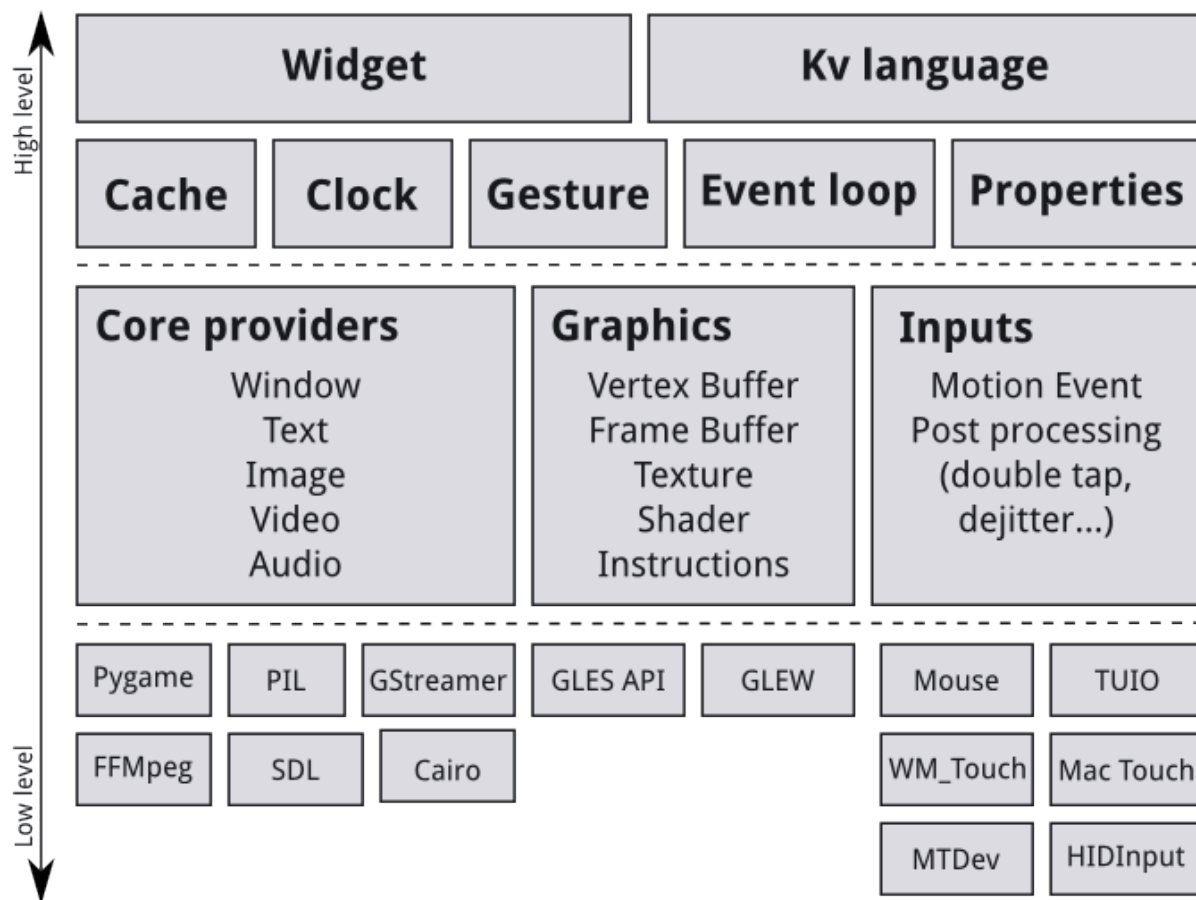
Architectural Overview

We would like to take a moment to explain how we designed Kivy from a software engineering point of view. This is key to understanding how everything works together. If you just look at the code, chances are you will get a rough idea already, but since this approach certainly is daunting for most users, this section explains the basic ideas of the implementation in more detail. You can skip this section and refer to it later, but we suggest at least skimming it for a rough overview.

Kivy consists of several building blocks that we will explain shortly. Here is a graphical summary of the architecture:



Kivy Architecture



8.1 Core Providers and Input Providers

One idea that is key to understanding Kivy's internals is that of modularity and abstraction. We try to abstract basic tasks such as opening a window, displaying images and text, playing audio, getting images from a camera, spelling correction and so on. We call these *core* tasks. This makes the API both easy to use and easy to extend. Most importantly, it allows us to use – what we call – specific providers for the respective scenarios in which your app is being run. For example, on OSX, Linux and Windows, there are different native APIs for the different core tasks. A piece of code that uses one of these specific APIs to talk to the operating system on one side and to Kivy on the other (acting as an intermediate communication layer) is what we call a *core provider*. The advantage of using specialized core providers for each platform is that we can fully leverage the functionality exposed by the operating system and act as efficiently as possible. It also gives users a choice. Furthermore, by using libraries that are shipped with any one platform, we effectively reduce the size of the Kivy distribution and make packaging easier. This also makes it easier to port Kivy to other platforms. The Android port benefited greatly from this.

We follow the same concept with input handling. An *input provider* is a piece of code that adds support for a specific input device, such as Apple's trackpads, TUIO or a mouse emulator. If you need to add support for a new input device, you can simply provide a new class that reads your input data from your device and transforms them into Kivy basic events.

8.2 Graphics

Kivy's graphics API is our abstraction of OpenGL. On the lowest level, Kivy issues hardware-accelerated drawing commands using OpenGL. Writing OpenGL code however can be a bit confusing, especially to newcomers. That's why we provide the graphics API that lets you draw things using simple metaphors that do not exist as such in OpenGL (e.g. Canvas, Rectangle, etc.).

All of our widgets themselves use this graphics API, which is implemented on the C level for performance reasons.

Another advantage of the graphics API is its ability to automatically optimize the drawing commands that your code issues. This is especially helpful if you're not an expert at tuning OpenGL. This makes your drawing code more efficient in many cases.

You can, of course, still use raw OpenGL commands if you prefer. The version we target is OpenGL 2.0 ES (GL ES2) on all devices, so if you want to stay cross-platform compatible, we advise you to only use the GL ES2 functions.

8.3 Core

The code in the core package provides commonly used features, such as:

Clock You can use the clock to schedule timer events. Both one-shot timers and periodic timers are supported.

Cache If you need to cache something that you use often, you can use our class for that instead of writing your own.

Gesture Detection We ship a simple gesture recognizer that you can use to detect various kinds of strokes, such as circles or rectangles. You can train it to detect your own strokes.

Kivy Language The kivy language is used to easily and efficiently describe user interfaces.

Properties These are not the normal properties that you may know from python. They are our own property classes that link your widget code with the user interface description.

8.4 UIX (Widgets & Layouts)

The UIX module contains commonly used widgets and layouts that you can reuse to quickly create a user interface.

Widgets Widgets are user interface elements that you add to your program to provide some kind of functionality. They may or may not be visible. Examples would be a file browser, buttons, sliders, lists and so on. Widgets receive MotionEvents.

Layouts You use layouts to arrange widgets. It is of course possible to calculate your widgets' positions yourself, but often it is more convenient to use one of our ready made layouts. Examples would be Grid Layouts or Box Layouts. You can also nest layouts.

8.5 Modules

If you've ever used a modern web browser and customized it with some add-ons then you already know the basic idea behind our module classes. Modules can be used to inject functionality into Kivy programs, even if the original author did not include it.

An example would be a module that always shows the FPS of the current application and some graph depicting the FPS over time.

You can also write your own modules.

8.6 Input Events (Touches)

Kivy abstracts different input types and sources such as touches, mice, TUIO or similar. What all of these input types have in common is that you can associate a 2D onscreen-position with any individual input event. (There are other input devices such as accelerometers where you cannot easily find a 2D position for e.g. a tilt of your device. This kind of input is handled separately. In the following we describe the former types.)

All of these input types are represented by instances of the `Touch()` class. (Note that this does not only refer to finger touches, but all the other input types as well. We just called it *Touch* for the sake of simplicity. Think of it of something that *touches* the user interface or your screen.) A touch instance, or object, can be in one of three states. When a touch enters one of these states, your program is informed that the event occurred. The three states a touch can be in are:

Down A touch is down only once, at the very moment where it first appears.

Move A touch can be in this state for a potentially unlimited time. A touch does not have to be in this state during its lifetime. A 'Move' happens whenever the 2D position of a touch changes.

Up A touch goes up at most once, or never. In practice you will almost always receive an up event because nobody is going to hold a finger on the screen for all eternity, but it is not guaranteed. If you know the input sources your users will be using, you will know whether or not you can rely on this state being entered.

8.7 Widgets and Event Dispatching

The term *widget* is often used in GUI programming contexts to describe some part of the program that the user interacts with. In Kivy, a widget is an object that receives input events. It does not necessarily have to have a visible representation on the screen. All widgets are arranged in a *widget tree* (which is a tree data structure as known from computer science classes): One widget can have any number of child widgets or none. There is exactly one *root widget* at the top of the tree that has no parent widget, and all other widgets are directly or indirectly children of this widget (which is why it's called the root).

When new input data is available, Kivy sends out one event per touch. The root widget of the widget tree first receives the event. Depending on the state of the touch, the `on_touch_down`, `on_touch_move` or `on_touch_up` event is dispatched (with the touch as the argument) to the root widget, which results in the root widget's corresponding `on_touch_down`, `on_touch_move` or `on_touch_up` event handler being called.

Each widget (this includes the root widget) in the tree can choose to either digest or pass the event on. If an event handler returns `True`, it means that the event has been digested and handled properly. No further processing will happen with that event. Otherwise, the event handler passes the widget on to its own children by calling its superclass's implementation of the respective event handler. This goes all the way up to the base `Widget` class, which – in its touch event handlers – does nothing but pass the touches to its children:

```
# This is analogous for move/up:
def on_touch_down(self, touch):
    for child in self.children[:]:
```

(continues on next page)

```
if child.dispatch('on_touch_down', touch):  
    return True
```

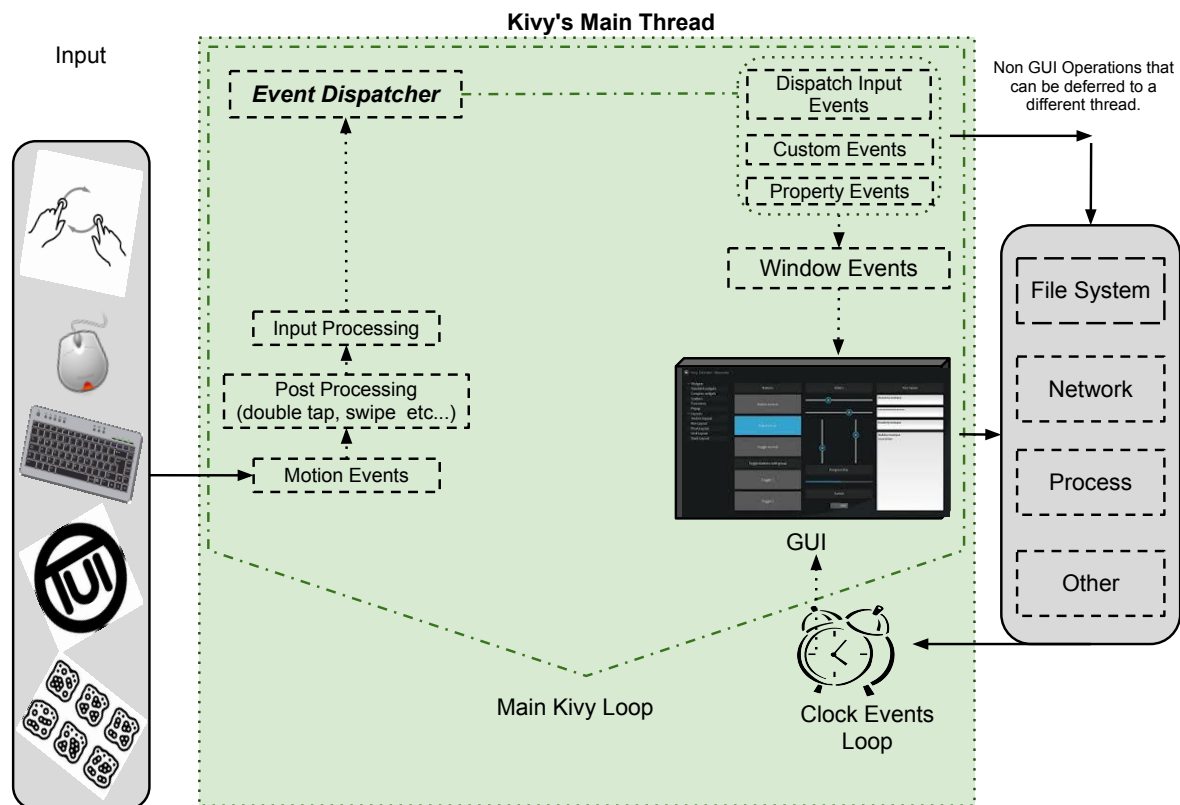
This really is much easier than it first seems. An example of how this can be used to create nice applications quickly will be given in the following section.

Often times you will want to restrict the *area* on the screen that a widget watches for touches. You can use a widget's `collide_point()` method to achieve this. You simply pass it the touch's position and it returns `True` if the touch is within the 'watched area' or `False` otherwise. By default, this checks the rectangular region on the screen that's described by the widget's `pos` (for position; `x` & `y`) and size (`width` & `height`), but you can override this behaviour in your own class.

Events and Properties

Events are an important part of Kivy programming. That may not be surprising to those with GUI development experience, but it's an important concept for newcomers. Once you understand how events work and how to bind to them, you will see them everywhere in Kivy. They make it easy to build whatever behavior you want into Kivy.

The following illustration shows how events are handled in the Kivy framework.



9.1 Introduction to the Event Dispatcher

One of the most important base classes of the framework is the *EventDispatcher* class. This class allows you to register event types, and to dispatch them to interested parties (usually other event dispatchers). The *Widget*, *Animation* and *Clock* classes are examples of event dispatchers.

EventDispatcher objects depend on the main loop to generate and handle events.

9.2 Main loop

As outlined in the illustration above, Kivy has a *main loop*. This loop is running during all of the application's lifetime and only quits when exiting the application.

Inside the loop, at every iteration, events are generated from user input, hardware sensors or a couple of other sources, and frames are rendered to the display.

Your application will specify callbacks (more on this later), which are called by the main loop. If a callback takes too long or doesn't quit at all, the main loop is broken and your app doesn't work properly anymore.

In Kivy applications, you have to avoid long/infinite loops or sleeping. For example the following code does both:

```
while True:
    animate_something()
    time.sleep(.10)
```

When you run this, the program will never exit your loop, preventing Kivy from doing all of the other things that need doing. As a result, all you'll see is a black window which you won't be able to interact with. Instead, you need to "schedule" your `animate_something()` function to be called repeatedly.

9.2.1 Scheduling a repetitive event

You can call a function or a method every X times per second using `schedule_interval()`. Here is an example of calling a function named `my_callback` 30 times per second:

```
def my_callback(dt):
    print 'My callback is called', dt
event = Clock.schedule_interval(my_callback, 1 / 30.)
```

You have multiple ways of unscheduling a previously scheduled event. One, is to use `cancel()` or `unschedule()`:

```
event.cancel()
```

or:

```
Clock.unschedule(event)
```

Alternatively, you can return `False` in your callback, and your event will be automatically unscheduled:

```
count = 0
def my_callback(dt):
    global count
    count += 1
    if count == 10:
```

(continues on next page)


```

    print 'Last call of my callback, bye bye !'
    return False
    print 'My callback is called'
Clock.schedule_interval(my_callback, 1 / 30.)

```

9.2.2 Scheduling a one-time event

Using `schedule_once()`, you can call a function “later”, like in the next frame, or in X seconds:

```

def my_callback(dt):
    print 'My callback is called !'
Clock.schedule_once(my_callback, 1)

```

This will call `my_callback` in one second. The second argument is the amount of time to wait before calling the function, in seconds. However, you can achieve some other results with special values for the second argument:

- If X is greater than 0, the callback will be called in X seconds
- If X is 0, the callback will be called after the next frame
- If X is -1, the callback will be called before the next frame

The -1 is mostly used when you are already in a scheduled event, and if you want to schedule a call BEFORE the next frame is happening.

A second method for repeating a function call is to first schedule a callback once with `schedule_once()`, and a second call to this function inside the callback itself:

```

def my_callback(dt):
    print 'My callback is called !'
    Clock.schedule_once(my_callback, 1)
Clock.schedule_once(my_callback, 1)

```

While the main loop will try to keep to the schedule as requested, there is some uncertainty as to when exactly a scheduled callback will be called. Sometimes another callback or some other task in the application will take longer than anticipated and thus the timing can be a little off.

In the latter solution to the repetitive callback problem, the next iteration will be called at least one second after the last iteration ends. With `schedule_interval()` however, the callback is called every second.

9.2.3 Trigger events

Sometimes you may want to schedule a function to be called only once for the next frame, preventing duplicate calls. You might be tempted to achieve that like so:

```

# First, schedule once.
event = Clock.schedule_once(my_callback, 0)

# Then, in another place you will have to unschedule first
# to avoid duplicate call. Then you can schedule again.
Clock.unschedule(event)
event = Clock.schedule_once(my_callback, 0)

```

This way of programming a trigger is expensive, since you’ll always call `unschedule`, even if the event has already completed. In addition, a new event is created every time. Use a trigger instead:

```
trigger = Clock.create_trigger(my_callback)
# later
trigger()
```

Each time you call `trigger()`, it will schedule a single call of your callback. If it was already scheduled, it will not be rescheduled.

9.3 Widget events

A widget has 2 default types of events:

- Property event: if your widget changes its position or size, an event is fired.
- Widget-defined event: e.g. an event will be fired for a `Button` when it's pressed or released.

For a discussion on how widget touch events managed and propagated, please refer to the *Widget touch event bubbling* section.

9.4 Creating custom events

To create an event dispatcher with custom events, you need to register the name of the event in the class and then create a method of the same name.

See the following example:

```
class MyEventDispatcher(EventDispatcher):
    def __init__(self, **kwargs):
        self.register_event_type('on_test')
        super(MyEventDispatcher, self).__init__(**kwargs)

    def do_something(self, value):
        # when do_something is called, the 'on_test' event will be
        # dispatched with the value
        self.dispatch('on_test', value)

    def on_test(self, *args):
        print "I am dispatched", args
```

9.5 Attaching callbacks

To use events, you have to bind callbacks to them. When the event is dispatched, your callbacks will be called with the parameters relevant to that specific event.

A callback can be any python callable, but you need to ensure it accepts the arguments that the event emits. For this, it's usually safest to accept the **args* argument, which will catch all arguments in the *args* list.

Example:

```
def my_callback(value, *args):
    print "Hello, I got an event!", args

ev = MyEventDispatcher()
```

(continues on next page)

```
ev.bind(on_test=my_callback)
ev.do_something('test')
```

Please refer to the [kivy.event.EventDispatcher.bind\(\)](#) method documentation for more examples on how to attach callbacks.

9.6 Introduction to Properties

Properties are an awesome way to define events and bind to them. Essentially, they produce events such that when an attribute of your object changes, all properties that reference that attribute are automatically updated.

There are different kinds of properties to describe the type of data you want to handle.

- [StringProperty](#)
- [NumericProperty](#)
- [BoundedNumericProperty](#)
- [ObjectProperty](#)
- [DictProperty](#)
- [ListProperty](#)
- [OptionProperty](#)
- [AliasProperty](#)
- [BooleanProperty](#)
- [ReferenceListProperty](#)

9.7 Declaration of a Property

To declare properties, you must declare them at the class level. The class will then do the work to instantiate the real attributes when your object is created. These properties are not attributes: they are mechanisms for creating events based on your attributes:

```
class MyWidget(Widget):
    text = StringProperty('')
```

When overriding `__init__`, always accept `**kwargs` and use `super()` to call the parent's `__init__` method, passing in your class instance:

```
def __init__(self, **kwargs):
    super(MyWidget, self).__init__(**kwargs)
```

9.8 Dispatching a Property event

Kivy properties, by default, provide an `on_<property_name>` event. This event is called when the value of the property is changed.

Note: If the new value for the property is equal to the current value, then the `on_<property_name>` event will not be called.

For example, consider the following code:

```
1 class CustomBtn(Widget):
2
3     pressed = ListProperty([0, 0])
4
5     def on_touch_down(self, touch):
6         if self.collide_point(*touch.pos):
7             self.pressed = touch.pos
8             return True
9         return super(CustomBtn, self).on_touch_down(touch)
10
11     def on_pressed(self, instance, pos):
12         print ('pressed at {pos}'.format(pos=pos))
```

In the code above at line 3:

```
pressed = ListProperty([0, 0])
```

We define the `pressed` Property of type `ListProperty`, giving it a default value of `[0, 0]`. From this point forward, the `on_pressed` event will be called whenever the value of this property is changed.

At Line 5:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):
        self.pressed = touch.pos
        return True
    return super(CustomBtn, self).on_touch_down(touch)
```

We override the `on_touch_down()` method of the `Widget` class. Here, we check for collision of the `touch` with our widget.

If the touch falls inside of our widget, we change the value of `pressed` to `touch.pos` and return `True`, indicating that we have consumed the touch and don't want it to propagate any further.

Finally, if the touch falls outside our widget, we call the original event using `super(...)` and return the result. This allows the touch event propagation to continue as it would normally have occurred.

Finally on line 11:

```
def on_pressed(self, instance, pos):
    print ('pressed at {pos}'.format(pos=pos))
```

We define an `on_pressed` function that will be called by the property whenever the property value is changed.

Note: This `on_<prop_name>` event is called within the class where the property is defined. To monitor/observe any change to a property outside of the class where it's defined, you should bind to the property as shown below.

Binding to the property

How to monitor changes to a property when all you have access to is a widget instance? You *bind* to the property:

```
your_widget_instance.bind(property_name=function_name)
```

For example, consider the following code:

```
1 class RootWidget(BoxLayout):
2
3     def __init__(self, **kwargs):
4         super(RootWidget, self).__init__(**kwargs)
5         self.add_widget(Button(text='btn 1'))
6         cb = CustomBtn()
7         cb.bind(pressed=self.btn_pressed)
8         self.add_widget(cb)
9         self.add_widget(Button(text='btn 2'))
10
11     def btn_pressed(self, instance, pos):
12         print ('pos: printed from root widget: {pos}'.format(pos=pos))
```

If you run the code as is, you will notice two print statements in the console. One from the *on_pressed* event that is called inside the *CustomBtn* class and another from the *btn_pressed* function that we bind to the property change.

The reason that both functions are called is simple. Binding doesn't mean overriding. Having both of these functions is redundant and you should generally only use one of the methods of listening/reacting to property changes.

You should also take note of the parameters that are passed to the *on_<property_name>* event or the function bound to the property.

```
def btn_pressed(self, instance, pos):
```

The first parameter is *self*, which is the instance of the class where this function is defined. You can use an in-line function as follows:

```
1 cb = CustomBtn()
2
3 def _local_func(instance, pos):
4     print ('pos: printed from root widget: {pos}'.format(pos=pos))
5
6 cb.bind(pressed=_local_func)
7 self.add_widget(cb)
```

The first parameter would be the *instance* of the class the property is defined.

The second parameter would be the *value*, which is the new value of the property.

Here is the complete example, derived from the snippets above, that you can use to copy and paste into an editor to experiment.

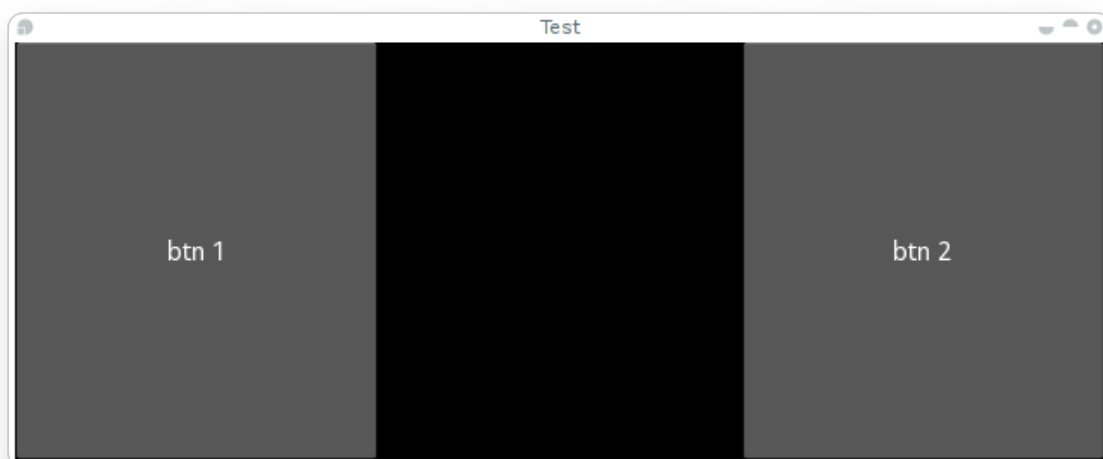
```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.uix.button import Button
4 from kivy.uix.boxlayout import BoxLayout
5 from kivy.properties import ListProperty
6
7 class RootWidget(BoxLayout):
8
9     def __init__(self, **kwargs):
10         super(RootWidget, self).__init__(**kwargs)
11         self.add_widget(Button(text='btn 1'))
12         cb = CustomBtn()
```

(continues on next page)

(continued from previous page)

```
13     cb.bind(pressed=self.btn_pressed)
14     self.add_widget(cb)
15     self.add_widget(Button(text='btn 2'))
16
17     def btn_pressed(self, instance, pos):
18         print ('pos: printed from root widget: {pos}'.format(pos=pos))
19
20 class CustomBtn(Widget):
21
22     pressed = ListProperty([0, 0])
23
24     def on_touch_down(self, touch):
25         if self.collide_point(*touch.pos):
26             self.pressed = touch.pos
27             # we consumed the touch. return False here to propagate
28             # the touch further to the children.
29             return True
30         return super(CustomBtn, self).on_touch_down(touch)
31
32     def on_pressed(self, instance, pos):
33         print ('pressed at {pos}'.format(pos=pos))
34
35 class TestApp(App):
36
37     def build(self):
38         return RootWidget()
39
40
41 if __name__ == '__main__':
42     TestApp().run()
```

Running the code above will give you the following output:



Our CustomBtn has no visual representation and thus appears black. You can touch/click on the black area to see the output on your console.

9.9 Compound Properties

When defining an *AliasProperty*, you normally define a getter and a setter function yourself. Here, it falls on to you to define when the getter and the setter functions are called using the *bind* argument.

Consider the following code.

```
1  cursor_pos = AliasProperty(_get_cursor_pos, None,
2                             bind=('cursor', 'padding', 'pos', 'size',
3                                 'focus', 'scroll_x', 'scroll_y',
4                                 'line_height', 'line_spacing'),
5                             cache=True)
6  '''Current position of the cursor, in (x, y).
7
8  :attr:`cursor_pos` is an :class:`~kivy.properties.AliasProperty`,
9  read-only.
10 '''
```

Here *cursor_pos* is a *AliasProperty* which uses the getter *_get_cursor_pos* with the setter part set to *None*, implying this is a read only Property.

The bind argument at the end defines that *on_cursor_pos* event is dispatched when any of the properties used in the *bind=* argument change.

Input management

10.1 Input architecture

Kivy is able to handle most types of input: mouse, touchscreen, accelerometer, gyroscope, etc. It handles the native multitouch protocols on the following platforms: Tuio, WM_Touch, MacMultitouchSupport, MT Protocol A/B and Android.

The global architecture can be viewed as:

Input providers -> Motion event -> Post processing -> Dispatch to Window

The class of all input events is the *MotionEvent*. It generates 2 kinds of events:

- Touch events: a motion event that contains at least an X and Y position. All the touch events are dispatched across the Widget tree.
- No-touch events: all the rest. For example, the accelerometer is a continuous event, without position. It never starts or stops. These events are not dispatched across the Widget tree.

A Motion event is generated by an *Input Provider*. An Input Provider is responsible for reading the input event from the operating system, the network or even from another application. Several input providers exist, such as:

- *TuioMotionEventProvider*: create a UDP server and listen for TUIO/OSC messages.
- *WM_MotionEventProvider*: use the windows API for reading multitouch information and sending it to Kivy.
- *ProbeSysfsHardwareProbe*: In Linux, iterate over all the hardware connected to the computer, and attaches a multitouch input provider for each multitouch device found.
- and much more!

When you write an application, you don't need to create an input provider. Kivy tries to automatically detect available hardware. However, if you want to support custom hardware, you will need to configure kivy to make it work.

Before the newly-created Motion Event is passed to the user, Kivy applies post-processing to the input. Every motion event is analyzed to detect and correct faulty input, as well as make meaningful interpretations like:

- Double/triple-tap detection, according to a distance and time threshold
- Making events more accurate when the hardware is not accurate
- Reducing the amount of generated events if the native touch hardware is sending events with nearly the same position

After processing, the motion event is dispatched to the Window. As explained previously, not all events are dispatched to the whole widget tree: the window filters them. For a given event:

- if it's only a motion event, it will be dispatched to `on_motion()`
- if it's a touch event, the (x,y) position of the touch (0-1 range) will be scaled to the Window size (width/height), and dispatched to:
 - `on_touch_down()`
 - `on_touch_move()`
 - `on_touch_up()`

10.2 Motion event profiles

Depending on your hardware and the input providers used, more information may be made available to you. For example, a touch input has an (x,y) position, but might also have pressure information, blob size, an acceleration vector, etc.

A profile is a string that indicates what features are available inside the motion event. Let's imagine that you are in an `on_touch_move` method:

```
def on_touch_move(self, touch):  
    print(touch.profile)  
    return super(..., self).on_touch_move(touch)
```

The print could output:

```
['pos', 'angle']
```

Warning: Many people mix up the profile's name and the name of the corresponding property. Just because 'angle' is in the available profile doesn't mean that the touch event object will have an angle property.

For the 'pos' profile, the properties `pos`, `x`, and `y` will be available. With the 'angle' profile, the property `a` will be available. As we said, for touch events 'pos' is a mandatory profile, but not 'angle'. You can extend your interaction by checking if the 'angle' profile exists:

```
def on_touch_move(self, touch):  
    print('The touch is at position', touch.pos)  
    if 'angle' in touch.profile:  
        print('The touch angle is', touch.a)
```

You can find a list of available profiles in the [motionevent](#) documentation.

10.3 Touch events

A touch event is a specialized *MotionEvent* where the property *is_touch* evaluates to True. For all touch events, you automatically have the X and Y positions available, scaled to the Window width and height. In other words, all touch events have the 'pos' profile.

10.3.1 Touch event basics

By default, touch events are dispatched to all currently displayed widgets. This means widgets receive the touch event whether it occurs within their physical area or not.

This can be counter intuitive if you have experience with other GUI toolkits. These typically divide the screen into geometric areas and only dispatch touch or mouse events to the widget if the coordinate lies within the widgets area.

This requirement becomes very restrictive when working with touch input. Swipes, pinches and long presses may well originate from outside of the widget that wants to know about them and react to them.

In order to provide the maximum flexibility, Kivy dispatches the events to all the widgets and lets them decide how to react to them. If you only want to respond to touch events inside the widget, you simply check:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):
        # The touch has occurred inside the widgets area. Do stuff!
        pass
```

10.3.2 Coordinates

You must take care of matrix transformation in your touch as soon as you use a widget with matrix transformation. Some widgets such as *Scatter* have their own matrix transformation, meaning the touch must be multiplied by the scatter matrix to be able to correctly dispatch touch positions to the Scatter's children.

- Get coordinate from parent space to local space: *to_local()*
- Get coordinate from local space to parent space: *to_parent()*
- Get coordinate from local space to window space: *to_window()*
- Get coordinate from window space to local space: *to_widget()*

You must use one of them to scale coordinates correctly to the context. Let's look the scatter implementation:

```
def on_touch_down(self, touch):
    # push the current coordinate, to be able to restore it later
    touch.push()

    # transform the touch coordinate to local space
    touch.apply_transform_2d(self.to_local)

    # dispatch the touch as usual to children
    # the coordinate in the touch is now in local space
    ret = super(..., self).on_touch_down(touch)

    # whatever the result, don't forget to pop your transformation
    # after the call, so the coordinate will be back in parent space
```

(continues on next page)

```
touch.pop()

# return the result (depending what you want.)
return ret
```

10.3.3 Touch shapes

If the touch has a shape, it will be reflected in the 'shape' property. Right now, only a *ShapeRect* can be exposed:

```
from kivy.input.shape import ShapeRect

def on_touch_move(self, touch):
    if isinstance(touch.shape, ShapeRect):
        print('My touch have a rectangle shape of size',
              (touch.shape.width, touch.shape.height))
    # ...
```

10.3.4 Double tap

A double tap is the action of tapping twice within a time and a distance. It's calculated by the doubletap post-processing module. You can test if the current touch is one of a double tap or not:

```
def on_touch_down(self, touch):
    if touch.is_double_tap:
        print('Touch is a double tap !')
        print(' - interval is', touch.double_tap_time)
        print(' - distance between previous is', touch.double_tap_distance)
    # ...
```

10.3.5 Triple tap

A triple tap is the action of tapping thrice within a time and a distance. It's calculated by the triplete tap post-processing module. You can test if the current touch is one of a triple tap or not:

```
def on_touch_down(self, touch):
    if touch.is_triple_tap:
        print('Touch is a triple tap !')
        print(' - interval is', touch.triple_tap_time)
        print(' - distance between previous is', touch.triple_tap_distance)
    # ...
```

10.3.6 Grabbing touch events

It's possible for the parent widget to dispatch a touch event to a child widget from within `on_touch_down`, but not from `on_touch_move` or `on_touch_up`. This can happen in certain scenarios, like when a touch movement is outside the bounding box of the parent, so the parent decides not to notify its children of the movement.

But you might want to do something in `on_touch_up`. Say you started something in the `on_touch_down` event, like playing a sound, and you'd like to finish things on the `on_touch_up` event. Grabbing is what you need.

When you grab a touch, you will always receive the move and up event. But there are some limitations to grabbing:

- You will receive the event at least twice: one time from your parent (the normal event), and one time from the window (grab).
- You might receive an event with a grabbed touch, but not from you: it can be because the parent has sent the touch to its children while it was in the grabbed state.

Here is an example of how to use grabbing:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):

        # if the touch collides with our widget, let's grab it
        touch.grab(self)

        # and accept the touch.
        return True

def on_touch_up(self, touch):
    # here, you don't check if the touch collides or things like that.
    # you just need to check if it's a grabbed touch event
    if touch.grab_current is self:

        # ok, the current touch is dispatched for us.
        # do something interesting here
        print('Hello world!')

        # don't forget to ungrab yourself, or you might have side effects
        touch.ungrab(self)

        # and accept the last up
        return True
```

10.3.7 Touch Event Management

In order to see how touch events are controlled and propagated between widgets, please refer to the *Widget touch event bubbling* section.

10.4 Joystick events

A joystick input represents raw values received directly from physical or virtual controllers through the SDL2 provider via these events:

- `SDL_JOYAXISMOTION`
- `SDL_JOYHATMOTION`
- `SDL_JOYBALLMOTION`
- `SDL_JOYBUTTONDOWN`
- `SDL_JOYBUTTONUP`

Every motion event has a minimum, maximum and default value which can reach:

Event	Minimum	Maximum	Default
on_joy_axis	-32767	32767	0
on_joy_hat	(-1, -1)	(1, 1)	(0, 0)
on_joy_ball	Unknown	Unknown	Unknown

Button events, on the other hand represent basically only a state of each button i.e. *up* and *down*, therefore no such values are present.

- on_joy_button_up
- on_joy_button_down

10.4.1 Joystick event basics

Unlike touch events, joystick events are dispatched directly to the Window, which means there's only a single value passed for e.g. a specified axis, not multiple ones. This makes things harder if you want to separate input to different widgets, yet not impossible. You can use [Multiple dropfile example](#) as an inspiration.

To get a joystick event, you first need to bind some function to the Window joystick event like this:

```
Window.bind(on_joy_axis=self.on_joy_axis)
```


Then you need to fetch the parameters specified in Window for each event you use, for example:

```
def on_joy_axis(self, win, stickid, axisid, value):
    print(win, stickid, axisid, value)
```

A variable *stickid* is an id of a controller that sent the value, *axisid* is an id of an axis to which the value belongs.

10.4.2 Joystick input

Kivy should be able to fetch input from any device specified as *gamepad*, *joystick* or basically any other type of game controller recognized by the SDL2 provider. To make things easier, here are layouts of some common controllers together with ids for each part.

			
#	ID	#	ID
1	axis 1	2	axis 0
3	hat Y	4	hat X
5	axis 4	6	axis 3
7	axis 2	8	axis 5
9	button 4	10	button 5
X	button 2	Y	button 3
A	button 0	B	button 1
back	button 6	start	button 7
center	button 10		

10.4.3 Joystick debugging

Mostly you'd want to debug your application with multiple controllers, or test it against `_other_` types of controllers (e.g. different brands). As an alternative you might want to use some of the available controller emulators, such as [vJoy](#).

Widgets

11.1 Introduction to Widget

A *Widget* is the base building block of GUI interfaces in Kivy. It provides a *Canvas* that can be used to draw on screen. It receives events and reacts to them. For an in-depth explanation about the *Widget* class, look at the module documentation.

11.2 Manipulating the Widget tree

Widgets in Kivy are organized in trees. Your application has a *root widget*, which usually has *children* that can have *children* of their own. Children of a widget are represented as the *children* attribute, a Kivy *ListProperty*.

The widget tree can be manipulated with the following methods:

- *add_widget()*: add a widget as a child
- *remove_widget()*: remove a widget from the children list
- *clear_widgets()*: remove all children from a widget

For example, if you want to add a button inside a *BoxLayout*, you can do:

```
layout = BoxLayout(padding=10)
button = Button(text='My first button')
layout.add_widget(button)
```

The button is added to layout: the button's parent property will be set to layout; the layout will have the button added to its children list. To remove the button from the layout:

```
layout.remove_widget(button)
```

With removal, the button's parent property will be set to *None*, and the layout will have button removed from its children list.

If you want to clear all the children inside a widget, use `clear_widgets()` method:

```
layout.clear_widgets()
```

Warning: Never manipulate the children list yourself, unless you really know what you are doing. The widget tree is associated with a graphic tree. For example, if you add a widget into the children list without adding its canvas to the graphics tree, the widget will be a child, yes, but nothing will be drawn on the screen. Moreover, you might have issues on further calls of `add_widget`, `remove_widget` and `clear_widgets`.

11.3 Traversing the Tree

The Widget class instance's `children` list property contains all the children. You can easily traverse the tree by doing:

```
root = BoxLayout()
# ... add widgets to root ...
for child in root.children:
    print(child)
```

However, this must be used carefully. If you intend to modify the children list with one of the methods shown in the previous section, you must use a copy of the list like this:

```
for child in root.children[:]:
    # manipulate the tree. For example here, remove all widgets that have a
    # width < 100
    if child.width < 100:
        root.remove_widget(child)
```

Widgets don't influence the size/pos of their children by default. The `pos` attribute is the absolute position in screen co-ordinates (unless, you use the `relativelayout`. More on that later) and `size`, is an absolute size.

11.4 Widgets Z Index

The order of widget drawing is based on the widget's position in the widget tree. The `add_widget` method takes an `index` parameter which can be used to specify its position in the widget tree:

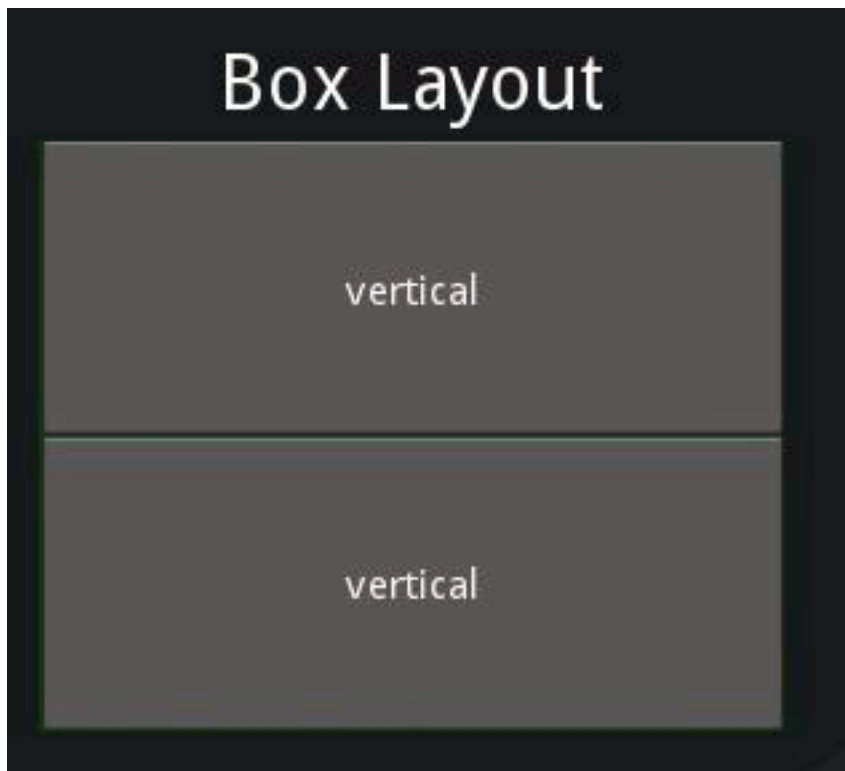
```
root.add_widget(widget, index)
```

The lower indexed widgets will be drawn above those with a higher index. Keep in mind that the default for `index` is 0, so widgets added later are drawn on top of the others unless specified otherwise.

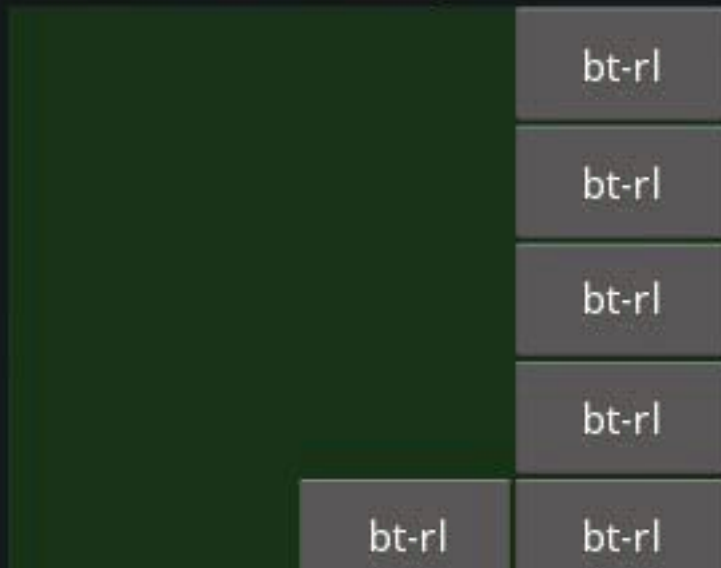
11.5 Organize with Layouts

`layout` is a special kind of widget that controls the size and position of its children. There are different kinds of layouts, allowing for different automatic organization of their children. Layouts use `size_hint` and `pos_hint` properties to determine the `size` and `pos` of their `children`.

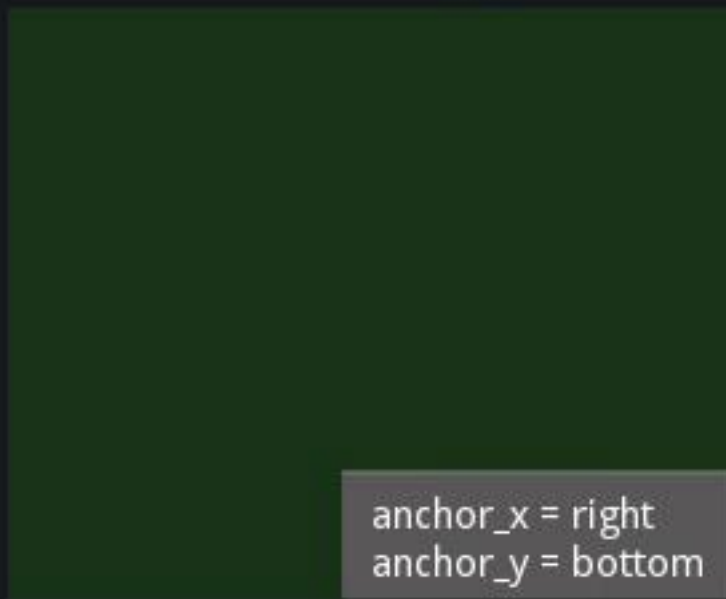
BoxLayout: Arranges widgets in an adjacent manner (either vertically or horizontally) manner, to fill all the space. The `size_hint` property of children can be used to change proportions allowed to each child, or set fixed size for some of them.



Stack Layout



Anchor Layout





GridLayout: Arranges widgets in a grid. You must specify at least one dimension of the grid so kivy can compute the size of the elements and how to arrange them.

StackLayout: Arranges widgets adjacent to one another, but with a set size in one of the dimensions, without trying to make them fit within the entire space. This is useful to display children of the same predefined size.

AnchorLayout: A simple layout only caring about children positions. It allows putting the children at a position relative to a border of the layout. *size_hint* is not honored.

FloatLayout: Allows placing children with arbitrary locations and size, either absolute or relative to the layout size. Default *size_hint* (1, 1) will make every child the same size as the whole layout, so you probably want to change this value if you have more than one child. You can set *size_hint* to (None, None) to use absolute size with *size*. This widget honors *pos_hint* also, which as a dict setting position relative to layout position.

RelativeLayout: Behaves just like FloatLayout, except children positions are relative to layout position, not the screen.

Examine the documentation of the individual layouts for a more in-depth understanding.

size_hint and *pos_hint*:

- *floatlayout*
- *boxlayout*
- *gridlayout*
- *stacklayout*
- *relativelayout*
- *anchorlayout*

size_hint is a *ReferenceListProperty* of *size_hint_x* and *size_hint_y*. It accepts values from 0 to 1 or *None* and defaults to (1, 1). This signifies that if the widget is in a layout, the layout will allocate it as much place as possible in both directions (relative to the layouts size).

Setting `size_hint` to (0.5, 0.8), for example, will make the widget 50% the width and 80% the height of available size for the `Widget` inside a `layout`.

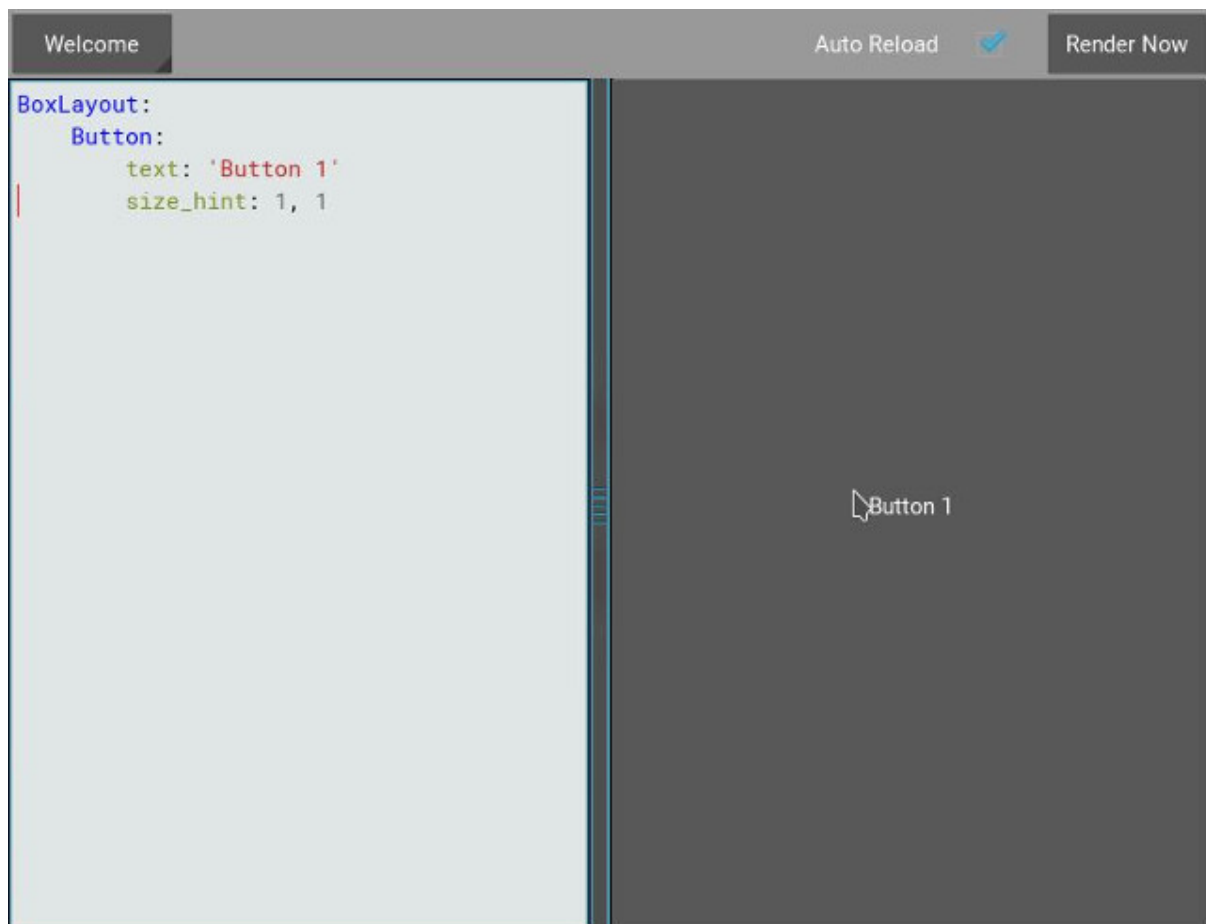
Consider the following example:

```
BoxLayout:
    Button:
        text: 'Button 1'
        # default size_hint is 1, 1, we don't need to specify it explicitly
        # however it's provided here to make things clear
        size_hint: 1, 1
```

Now load kivy catalog by typing the following, but replacing `$KIVYDIR` with the directory of your installation (discoverable via `os.path.dirname(kivy.__file__)`):

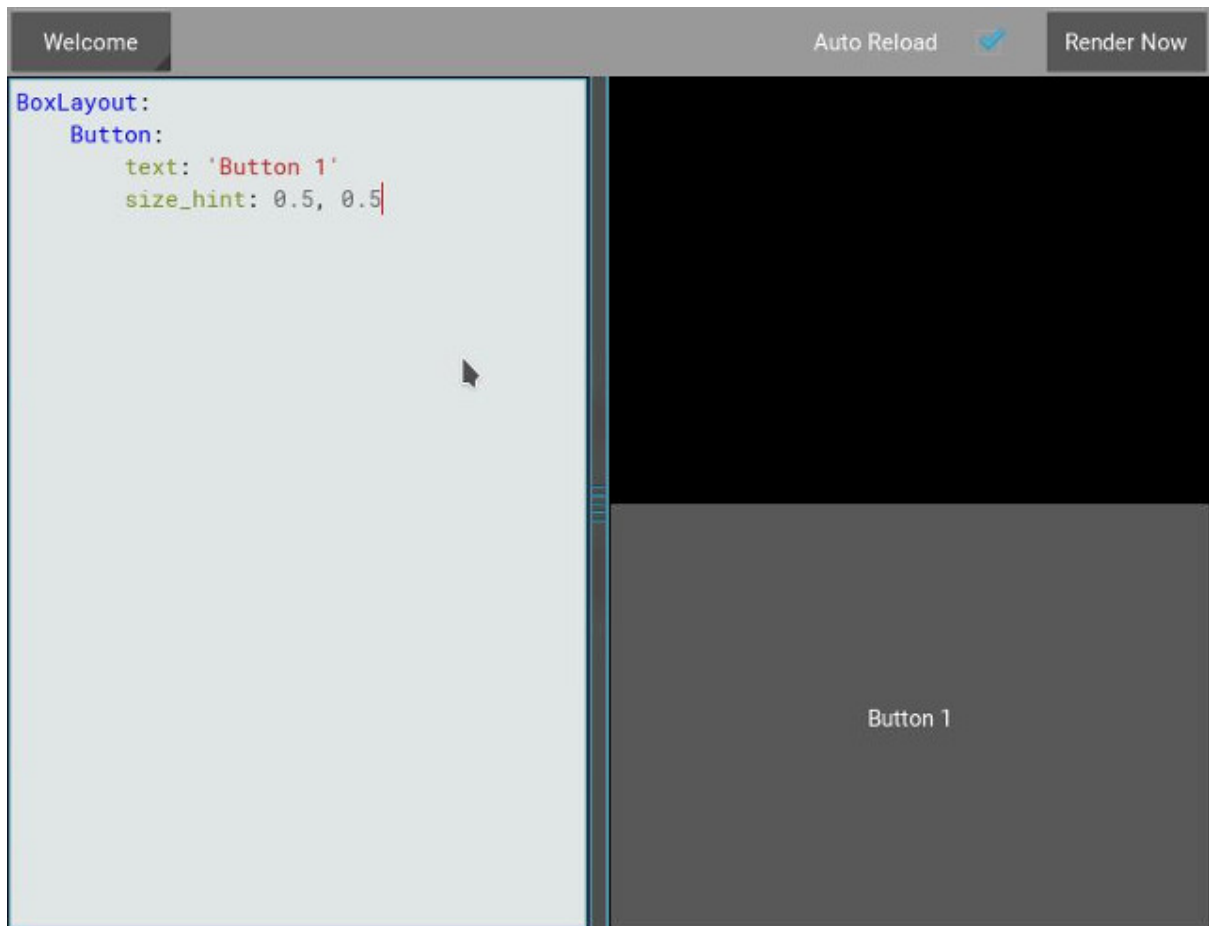
```
cd $KIVYDIR/examples/demo/kivycatalog
python main.py
```

A new window will appear. Click in the area below the 'Welcome' `Spinner` on the left and replace the text there with your kv code from above.



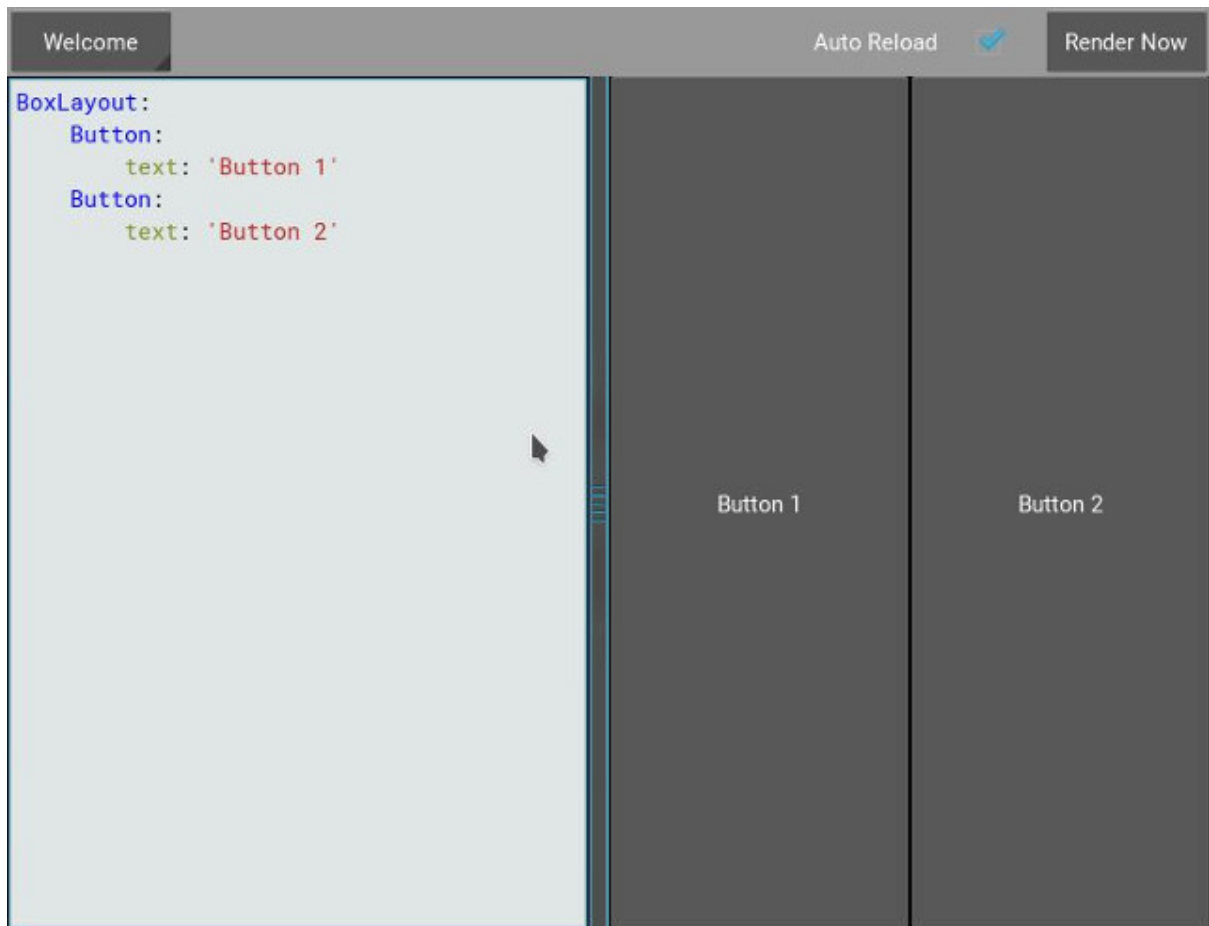
As you can see from the image above, the `Button` takes up 100% of the layout `size`.

Changing the `size_hint_x/size_hint_y` to .5 will make the `Widget` take 50% of the `layout width/height`.

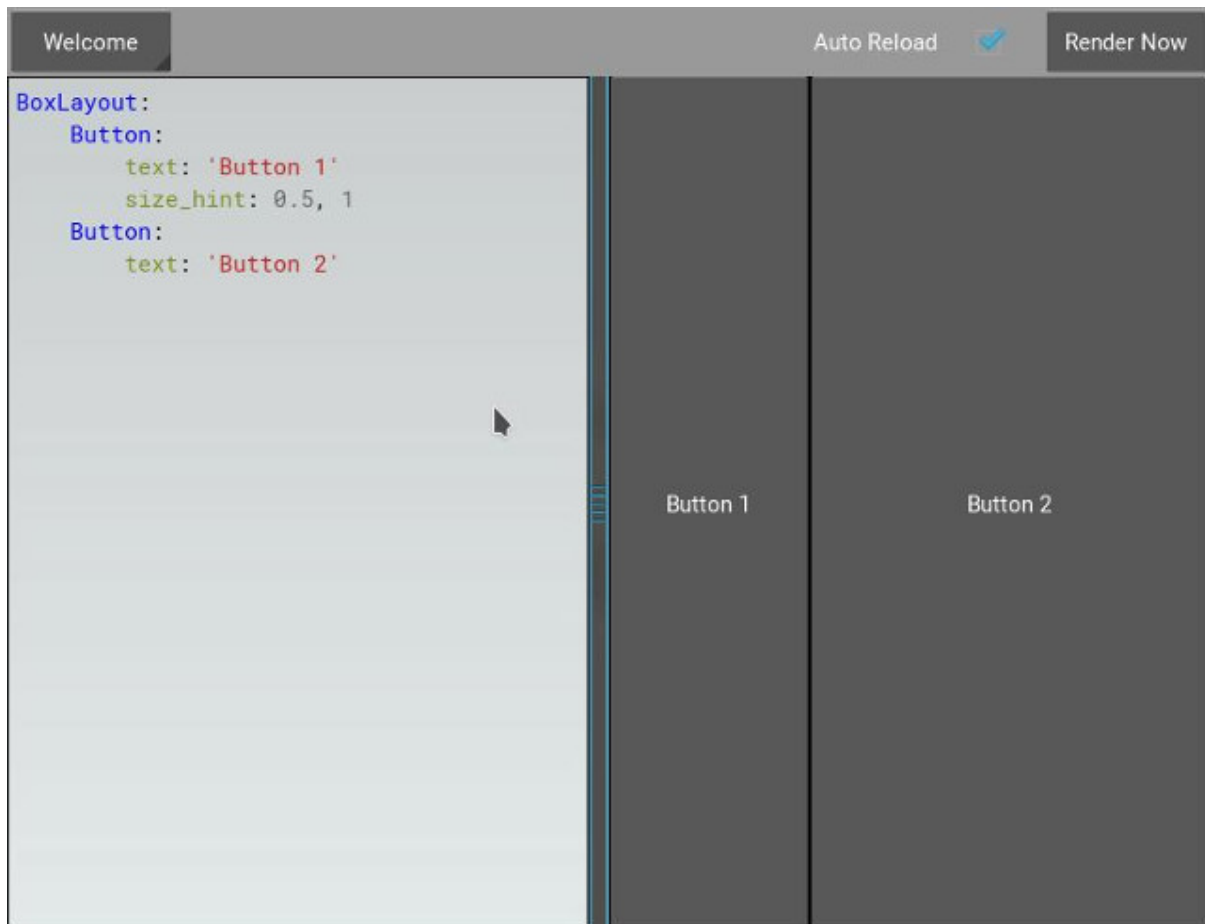


You can see here that, although we specify `size_hint_x` and `size_hint_y` both to be `.5`, only `size_hint_y` seems to be honored. That is because `boxlayout` controls the `size_hint_y` when `orientation` is `vertical` and `size_hint_x` when `orientation` is `'horizontal'`. The controlled dimension's size is calculated depending upon the total no. of `children` in the `boxlayout`. In this example, one child has `size_hint_y` controlled ($.5/.5 = 1$). Thus, the widget takes 100% of the parent layout's height.

Let's add another `Button` to the `layout` and see what happens.



boxlayout by its very nature divides the available space between its *children* equally. In our example, the proportion is 50-50, because we have two *children*. Let's use `size_hint` on one of the children and see the results.



If a child specifies `size_hint`, this specifies how much space the *Widget* will take out of the *size* given to it by the *boxlayout*. In our example, the first *Button* specifies .5 for `size_hint_x`. The space for the widget is calculated like so:

```
first child's size_hint divided by
first child's size_hint + second child's size_hint + ...n(no of children)

.5/(.5+1) = .333...
```

The rest of the *BoxLayout*'s *width* is divided among the rest of the *children*. In our example, this means the second *Button* takes up 66.66% of the *layout width*.

Experiment with `size_hint` to get comfortable with it.

If you want to control the absolute *size* of a *Widget*, you can set `size_hint_x/size_hint_y` or both to *None* so that the widget's *width* and or *height* attributes will be honored.

`pos_hint` is a dict, which defaults to empty. As for `size_hint`, layouts honor `pos_hint` differently, but generally you can add values to any of the `pos` attributes (*x*, *y*, *right*, *top*, *center_x*, *center_y*) to have the *Widget* positioned relative to its *parent*.

Let's experiment with the following code in *kivycatalog* to understand `pos_hint` visually:

```
FloatLayout:
    Button:
        text: "We Will"
        pos: 100, 100
        size_hint: .2, .4
    Button:
        text: "Wee Wiill"
```

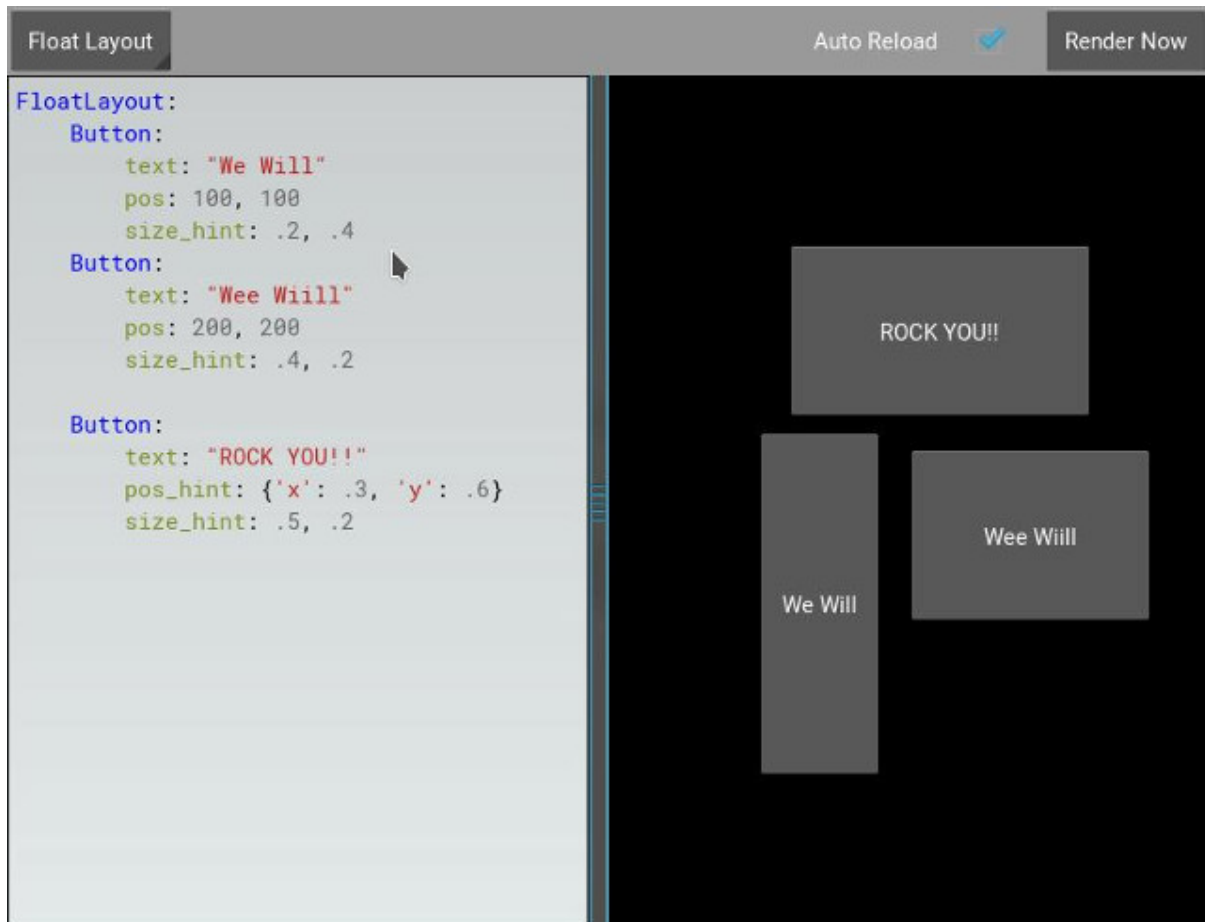
(continues on next page)

(continued from previous page)

```
pos: 200, 200
size_hint: .4, .2

Button:
    text: "ROCK YOU!!"
    pos_hint: {'x': .3, 'y': .6}
    size_hint: .5, .2
```

This gives us:



As with *size_hint*, you should experiment with *pos_hint* to understand the effect it has on the widget positions.

11.6 Adding a Background to a Layout

One of the frequently asked questions about layouts is::

"How to add a background image/color/video/... to a Layout"

Layouts by their nature have no visual representation: they have no canvas instructions by default. However you can add canvas instructions to a layout instance easily, as with adding a colored background:

In Python:

```

from kivy.graphics import Color, Rectangle

with layout_instance.canvas.before:
    Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-255
    self.rect = Rectangle(size=layout_instance.size,
                          pos=layout_instance.pos)

```

Unfortunately, this will only draw a rectangle at the layout's initial position and size. To make sure the rect is drawn inside the layout, when the layout size/pos changes, we need to listen to any changes and update the rectangles size and pos. We can do that as follows:

```

with layout_instance.canvas.before:
    Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-255
    self.rect = Rectangle(size=layout_instance.size,
                          pos=layout_instance.pos)

def update_rect(instance, value):
    instance.rect.pos = instance.pos
    instance.rect.size = instance.size

# listen to size and position changes
layout_instance.bind(pos=update_rect, size=update_rect)

```

In kv:

```

FloatLayout:
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            # self here refers to the widget i.e BoxLayout
            pos: self.pos
            size: self.size

```

The kv declaration sets an implicit binding: the last two kv lines ensure that the *pos* and *size* values of the rectangle will update when the *pos* of the *floatlayout* changes.

Now we put the snippets above into the shell of Kivy App.

Pure Python way:

```

from kivy.app import App
from kivy.graphics import Color, Rectangle
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.button import Button

class RootWidget(FloatLayout):

    def __init__(self, **kwargs):
        # make sure we aren't overriding any important functionality
        super(RootWidget, self).__init__(**kwargs)

        # let's add a Widget to this layout
        self.add_widget(
            Button(
                text="Hello World",
                size_hint=(.5, .5),
                pos_hint={'center_x': .5, 'center_y': .5}))

```

(continues on next page)

```

class MainApp(App):

    def build(self):
        self.root = root = RootWidget()
        root.bind(size=self._update_rect, pos=self._update_rect)

        with root.canvas.before:
            Color(0, 1, 0, 1) # green; colors range from 0-1 not 0-255
            self.rect = Rectangle(size=root.size, pos=root.pos)
        return root

    def _update_rect(self, instance, value):
        self.rect.pos = instance.pos
        self.rect.size = instance.size

if __name__ == '__main__':
    MainApp().run()

```

Using the kv Language:

```

from kivy.app import App
from kivy.lang import Builder

root = Builder.load_string('''
FloatLayout:
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            # self here refers to the widget i.e FloatLayout
            pos: self.pos
            size: self.size
    Button:
        text: 'Hello World!!'
        size_hint: .5, .5
        pos_hint: {'center_x':.5, 'center_y': .5}
''')

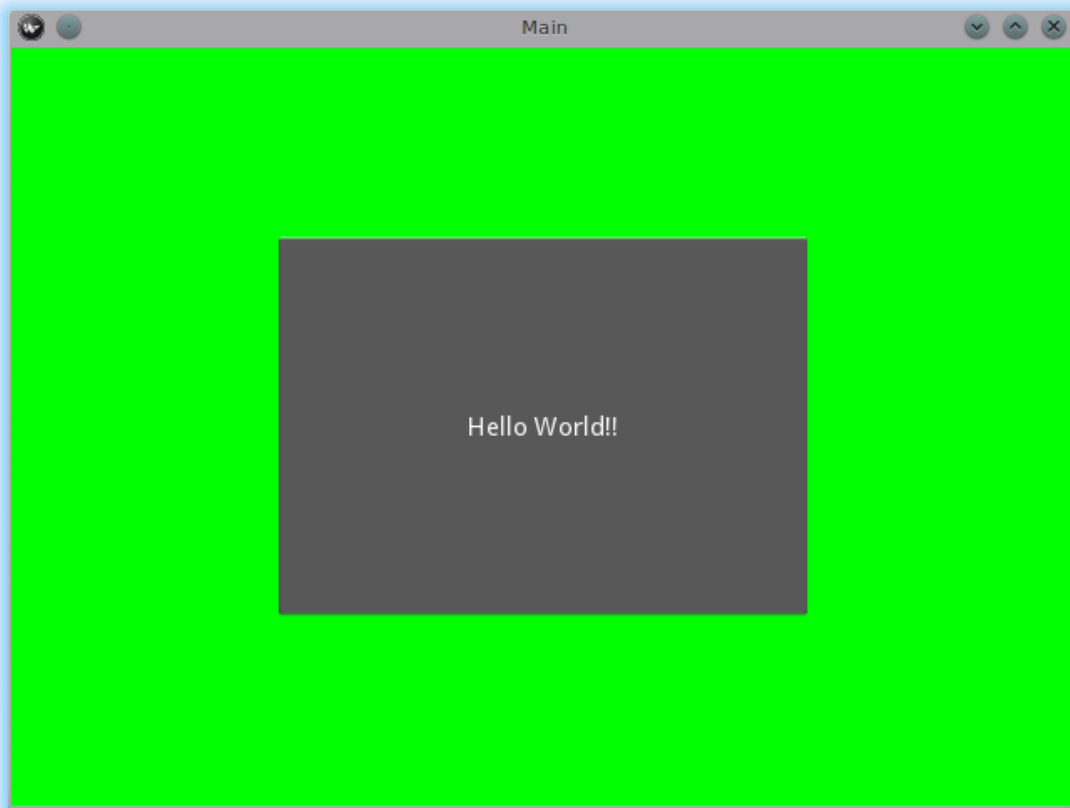
class MainApp(App):

    def build(self):
        return root

if __name__ == '__main__':
    MainApp().run()

```

Both of the Apps should look something like this:



11.6.1 Add a color to the background of a **custom layouts rule/class**

The way we add background to the layout's instance can quickly become cumbersome if we need to use multiple layouts. To help with this, you can subclass the Layout and create your own layout that adds a background.

Using Python:

```
from kivy.app import App
from kivy.graphics import Color, Rectangle
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.image import AsyncImage

class RootWidget(BoxLayout):
    pass

class CustomLayout(FloatLayout):

    def __init__(self, **kwargs):
        # make sure we aren't overriding any important functionality
        super(CustomLayout, self).__init__(**kwargs)

        with self.canvas.before:
            Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-255
```

(continues on next page)

(continued from previous page)

```
        self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self._update_rect, pos=self._update_rect)

    def _update_rect(self, instance, value):
        self.rect.pos = instance.pos
        self.rect.size = instance.size

class MainApp(App):

    def build(self):
        root = RootWidget()
        c = CustomLayout()
        root.add_widget(c)
        c.add_widget(
            AsyncImage(
                source="http://www.everythingzoomer.com/wp-content/uploads/2013/01/
→Monday-joke-289x277.jpg",
                size_hint= (1, .5),
                pos_hint={'center_x':.5, 'center_y':.5}))
        root.add_widget(AsyncImage(source='http://www.stuffistumbledupon.com/wp-
→content/uploads/2012/05/Have-you-seen-this-dog-because-its-awesome-meme-puppy-
→doggy.jpg'))
        c = CustomLayout()
        c.add_widget(
            AsyncImage(
                source="http://www.stuffistumbledupon.com/wp-content/uploads/2012/
→04/Get-a-Girlfriend-Meme-empty-wallet.jpg",
                size_hint= (1, .5),
                pos_hint={'center_x':.5, 'center_y':.5}))
        root.add_widget(c)
        return root

if __name__ == '__main__':
    MainApp().run()
```

Using the kv Language:

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder

Builder.load_string('''
<CustomLayout>
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            pos: self.pos
            size: self.size

<RootWidget>
    CustomLayout:
        AsyncImage:
            source: 'http://www.everythingzoomer.com/wp-content/uploads/2013/01/
→Monday-joke-289x277.jpg'
```

(continues on next page)

(continued from previous page)

```
        size_hint: 1, .5
        pos_hint: {'center_x':.5, 'center_y': .5}
    AsyncImage:
        source: 'http://www.stuffistumbledupon.com/wp-content/uploads/2012/05/Have-
        ↳you-seen-this-dog-because-its-awesome-meme-puppy-doggy.jpg'
    CustomLayout
        AsyncImage:
            source: 'http://www.stuffistumbledupon.com/wp-content/uploads/2012/04/
            ↳Get-a-Girlfriend-Meme-empty-wallet.jpg'
            size_hint: 1, .5
            pos_hint: {'center_x':.5, 'center_y': .5}
'''

class RootWidget(BoxLayout):
    pass

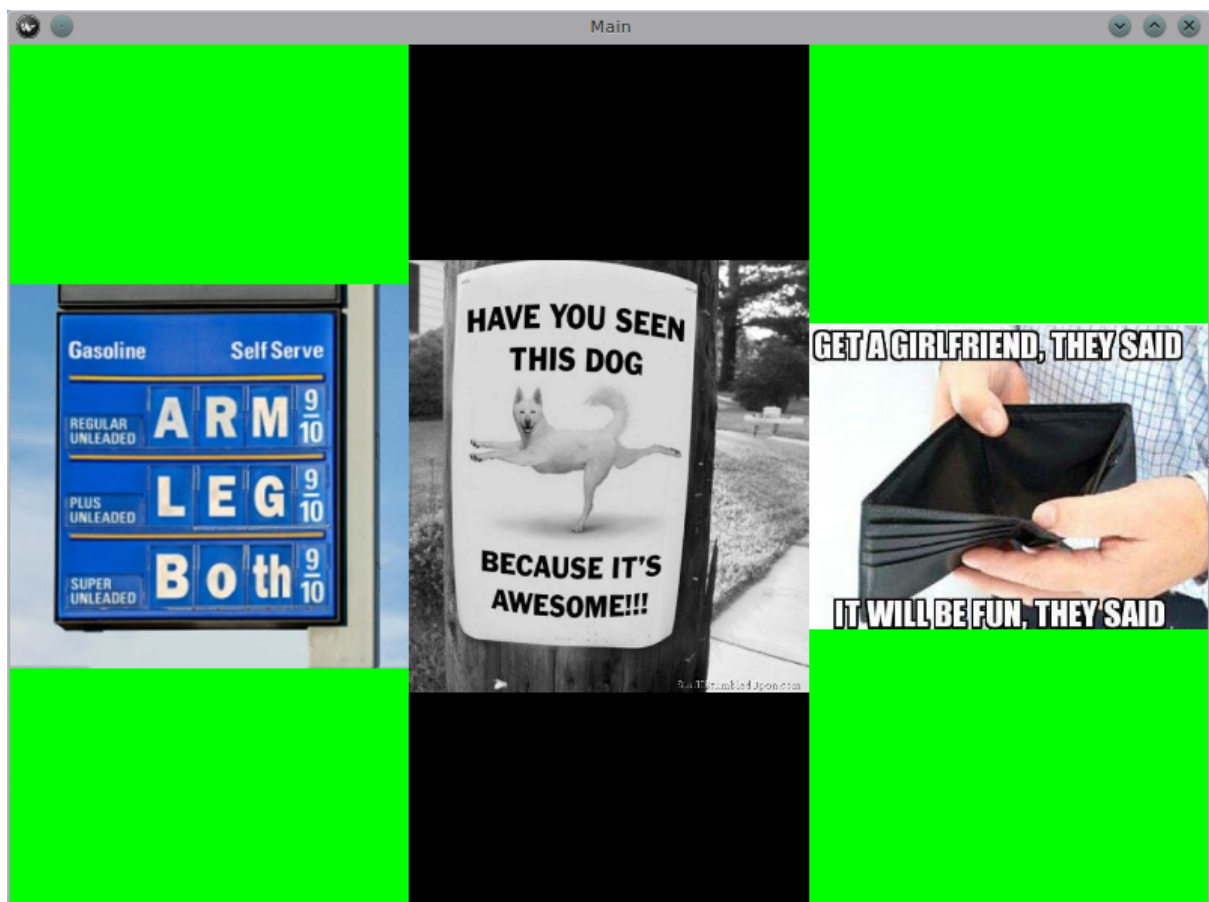
class CustomLayout(FloatLayout):
    pass

class MainApp(App):

    def build(self):
        return RootWidget()

if __name__ == '__main__':
    MainApp().run()
```

Both of the Apps should look something like this:



Defining the background in the custom layout class, assures that it will be used in every instance of CustomLayout.

Now, to add an image or color to the background of a built-in Kivy layout, **globally**, we need to override the kv rule for the layout in question. Consider GridLayout:

```
<GridLayout>
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        BorderImage:
            source: '../examples/widgets/sequenced_images/data/images/button_white.
→png'
            pos: self.pos
            size: self.size
```

Then, when we put this snippet into a Kivy app:

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder

Builder.load_string('''
<GridLayout>
    canvas.before:
        BorderImage:
            # BorderImage behaves like the CSS BorderImage
            border: 10, 10, 10, 10
            source: '../examples/widgets/sequenced_images/data/images/button_white.
→png'
            pos: self.pos
            size: self.size

<RootWidget>
    GridLayout:
        size_hint: .9, .9
        pos_hint: {'center_x': .5, 'center_y': .5}
        rows:1
        Label:
            text: "I don't suffer from insanity, I enjoy every minute of it"
            text_size: self.width-20, self.height-20
            valign: 'top'
        Label:
            text: "When I was born I was so surprised; I didn't speak for a year_
→and a half."
            text_size: self.width-20, self.height-20
            valign: 'middle'
            halign: 'center'
        Label:
            text: "A consultant is someone who takes a subject you understand and_
→makes it sound confusing"
            text_size: self.width-20, self.height-20
            valign: 'bottom'
            halign: 'justify'
''')

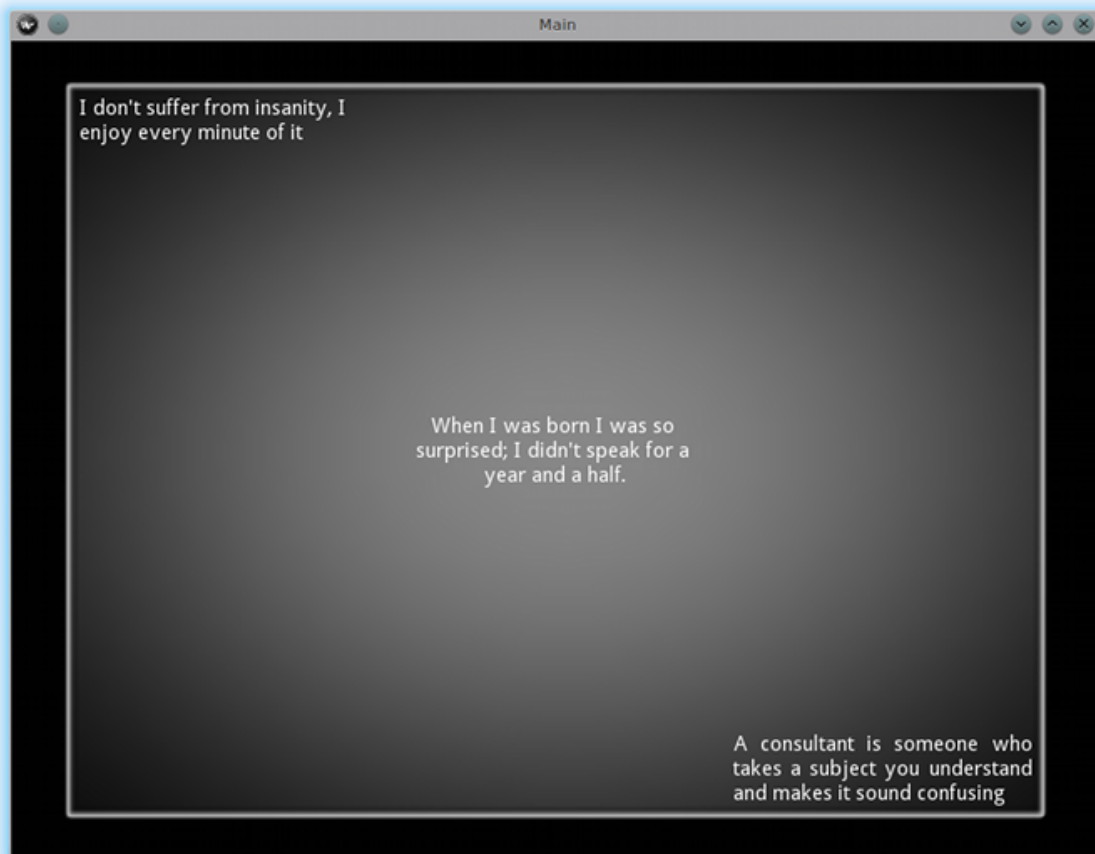
class RootWidget(FloatLayout):
    pass
```

(continues on next page)

(continued from previous page)

```
class MainApp(App):  
    def build(self):  
        return RootWidget()  
  
if __name__ == '__main__':  
    MainApp().run()
```

The result should look something like this:



As we are overriding the rule of the class GridLayout, any use of this class in our app will display that image.

How about an **Animated background**?

You can set the drawing instructions like Rectangle/BorderImage/Ellipse/... to use a particular texture:

```
Rectangle:  
    texture: reference to a texture
```

We use this to display an animated background:

```
from kivy.app import App  
from kivy.uix.floatlayout import FloatLayout  
from kivy.uix.gridlayout import GridLayout
```

(continues on next page)

```

from kivy.uix.image import Image
from kivy.properties import ObjectProperty
from kivy.lang import Builder

Builder.load_string('''
<CustomLayout>
    canvas.before:
        BorderImage:
            # BorderImage behaves like the CSS BorderImage
            border: 10, 10, 10, 10
            texture: self.background_image.texture
            pos: self.pos
            size: self.size

<RootWidget>
    CustomLayout:
        size_hint: .9, .9
        pos_hint: {'center_x': .5, 'center_y': .5}
        rows:1
        Label:
            text: "I don't suffer from insanity, I enjoy every minute of it"
            text_size: self.width-20, self.height-20
            valign: 'top'
        Label:
            text: "When I was born I was so surprised; I didn't speak for a year_
→and a half."
            text_size: self.width-20, self.height-20
            valign: 'middle'
            halign: 'center'
        Label:
            text: "A consultant is someone who takes a subject you understand and_
→makes it sound confusing"
            text_size: self.width-20, self.height-20
            valign: 'bottom'
            halign: 'justify'
''')

class CustomLayout(GridLayout):

    background_image = ObjectProperty(
        Image(
            source='../examples/widgets/sequenced_images/data/images/button_white_
→animated.zip',
            anim_delay=.1))

class RootWidget(FloatLayout):
    pass

class MainApp(App):

    def build(self):
        return RootWidget()

if __name__ == '__main__':

```

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

To try to understand what is happening here, start from line 13:

```
texture: self.background_image.texture
```

This specifies that the *texture* property of *BorderImage* will be updated whenever the *texture* property of *background_image* updates. We define the *background_image* property at line 40:

```
background_image = ObjectProperty(...
```

This sets up *background_image* as an *ObjectProperty* in which we add an *Image* widget. An image widget has a *texture* property; where you see *self.background_image.texture*, this sets a reference, *texture*, to this property. The *Image* widget supports animation: the texture of the image is updated whenever the animation changes, and the texture of *BorderImage* instruction is updated in the process.

You can also just blit custom data to the texture. For details, look at the documentation of *Texture*.

11.7 Nesting Layouts

Yes! It is quite fun to see how extensible the process can be.

11.8 Size and position metrics

Kivy's default unit for length is the pixel, all sizes and positions are expressed in it by default. You can express them in other units, which is useful to achieve better consistency across devices (they get converted to the size in pixels automatically).

Available units are *pt*, *mm*, *cm*, *inch*, *dp* and *sp*. You can learn about their usage in the *metrics* documentation.

You can also experiment with the *screen* usage to simulate various devices screens for your application.

11.9 Screen Separation with Screen Manager

If your application is composed of various screens, you likely want an easy way to navigate from one *Screen* to another. Fortunately, there is the *ScreenManager* class, that allows you to define screens separately, and to set the *TransitionBase* from one to another.

Graphics

12.1 Introduction to Canvas

A Widgets graphical representation is rendered using a canvas, which you can see as both an unlimited drawing board or as a set of drawing instructions. There are numerous instructions you can apply (add) to your canvas, but there are two main variations:

- *context instructions*
- *vertex instructions*

Context instructions don't draw anything, but they change the results of the vertex instructions.

Canvasses can contain two subsets of instructions. They are the *canvas.before* and the *canvas.after* instruction groups. The instructions in these groups will be executed before and after the *canvas* group respectively. This means that they will appear under (be executed before) and above (be executed after) them. Those groups are not created until the user accesses them.

To add a canvas instruction to a widget, you use the canvas context:

```
class MyWidget(Widget):
    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        with self.canvas:
            # add your instruction for main canvas here

        with self.canvas.before:
            # you can use this to add instructions rendered before

        with self.canvas.after:
            # you can use this to add instructions rendered after
```

12.2 Context instructions

Context instructions manipulate the opengl context. You can rotate, translate, and scale your canvas. You can also attach a texture or change the drawing color. This one is the most commonly used, but others are really useful too:

```
with self.canvas.before:  
    Color(1, 0, .4, mode='rgb')
```

12.3 Drawing instructions

Drawing instructions range from very simple ones, like drawing a line or a polygon, to more complex ones, like meshes or bezier curves:

```
with self.canvas:  
    # draw a line using the default color  
    Line(points=(x1, y1, x2, y2, x3, y3))  
  
    # lets draw a semi-transparent red square  
    Color(1, 0, 0, .5, mode='rgba')  
    Rectangle(pos=self.pos, size=self.size)
```

12.4 Manipulating instructions

Sometimes you want to update or remove the instructions you have added to a canvas. This can be done in various ways depending on your needs:

You can keep a reference to your instructions and update them:

```
class MyWidget(Widget):  
    def __init__(self, **kwargs):  
        super(MyWidget, self).__init__(**kwargs)  
        with self.canvas:  
            self.rect = Rectangle(pos=self.pos, size=self.size)  
  
            self.bind(pos=self.update_rect)  
            self.bind(size=self.update_rect)  
  
    def update_rect(self, *args):  
        self.rect.pos = self.pos  
        self.rect.size = self.size
```

Or you can clean your canvas and start fresh:

```
class MyWidget(Widget):  
    def __init__(self, **kwargs):  
        super(MyWidget, self).__init__(**kwargs)  
        self.draw_my_stuff()  
  
        self.bind(pos=self.draw_my_stuff)  
        self.bind(size=self.draw_my_stuff)  
  
    def draw_my_stuff(self, *args):  
        self.canvas.clear()
```

(continues on next page)

(continued from previous page)

```
with self.canvas:  
    self.rect = Rectangle(pos=self.pos, size=self.size)
```

Note that updating the instructions is considered the best practice as it involves less overhead and avoids creating new instructions.

Kv language

13.1 Concept behind the language

As your application grows more complex, it's common that the construction of widget trees and explicit declaration of bindings becomes verbose and hard to maintain. The KV Language is an attempt to overcome these shortcomings.

The KV language, sometimes called kvleng or the kivy language, allows you to create your widget tree in a declarative way and to bind widget properties to each other or to callbacks in a natural manner. It allows for very fast prototypes and agile changes to your UI. It also facilitates separating the logic of your application and its User Interface.

13.2 How to load KV

There are two ways to load Kv code into your application:

- By name convention:

Kivy looks for a Kv file with the same name as your App class in lowercase, minus "App" if it ends with 'App' e.g:

```
MyApp -> my.kv
```

If this file defines a *Root Widget* it will be attached to the App's *root* attribute and used as the base of the application widget tree.

- **Builder:** You can tell Kivy to directly load a string or a file. If this string or file defines a root widget, it will be returned by the method:

```
Builder.load_file('path/to/file.kv')
```

or:

```
Builder.load_string(kv_string)
```

13.3 Rule context

A Kv source constitutes of *rules* which are used to describe the content of a Widget. You can have one *root* rule, and any number of *class* or *template* rules.

The *root* rule is declared by declaring the class of your root widget, without any indentation, followed by `:` and will be set as the *root* attribute of the App instance:

```
Widget:
```

A *class* rule, declared by the name of a widget class between `< >` and followed by `:`, defines the appearance and behavior of any instance of that class:

```
<MyWidget>:
```

Rules use indentation for delimitation, like Python. Indentation should be four spaces per level, like the Python style guide [recommends](#).

There are three keywords specific to the Kv language:

- *app*: always refers to the instance of your application.
- *root*: refers to the base widget/template in the current rule
- *self*: always refer to the current widget

13.4 Special syntax

There is a special syntax to define values for the whole Kv context.

To access Python modules and classes from kv, use `#:import`

```
#:import name x.y.z  
#:import isdir os.path.isdir  
#:import np numpy
```

is equivalent to:

```
from x.y import z as name  
from os.path import isdir  
import numpy as np
```

in Python.

To set a global value, use `#:set`

```
#:set name value
```

is equivalent to:

```
name = value
```

in Python.

13.5 Instantiate children

To declare a widget instance of some class as a child widget, just declare that child inside the rule:

```
MyRootWidget:
    BoxLayout:
        Button:
        Button:
```

The example above defines that our root widget, an instance of *MyRootWidget*, has a child that is an instance of the *BoxLayout*, and that *BoxLayout* further has two children, instances of the *Button* class.

The Python equivalent of this code might be:

```
root = MyRootWidget()
box = BoxLayout()
box.add_widget(Button())
box.add_widget(Button())
root.add_widget(box)
```

Which you may find less nice, both to read and to write.

Of course, in Python, you can pass keyword arguments to your widgets at creation to specify their behaviour. For example, to set the number of columns of a *gridlayout*, we would do:

```
grid = GridLayout(cols=3)
```

To do the same thing in kv, you can set properties of the child widget directly in the rule:

```
GridLayout:
    cols: 3
```

The value is evaluated as a Python expression, and all the properties used in the expression will be observed, that means that if you had something like this in Python (this assume *self* is a widget with a *data* *ListProperty*):

```
grid = GridLayout(cols=len(self.data))
self.bind(data=grid.setter('cols'))
```

To have your display updated when your data change, you can now have just:

```
GridLayout:
    cols: len(root.data)
```

Note: Widget names should start with upper case letters while property names should start with lower case ones. Following the [PEP8 Naming Conventions](#) is encouraged.

13.6 Event Bindings

You can bind to events in Kv using the “:” syntax, that is, associating a callback to an event:

```
Widget:
    on_size: my_callback()
```

You can pass the values dispatched by the signal using the *args* keyword:

```
TextInput:
    on_text: app.search(args[1])
```

More complex expressions can be used, like:

```
pos: self.center_x - self.texture_size[0] / 2., self.center_y - self.texture_
    ↪size[1] / 2.
```

This expression listens for a change in `center_x`, `center_y`, and `texture_size`. If one of them changes, the expression will be re-evaluated to update the `pos` field.

You can also handle `on_` events inside your kv language. For example the `TextInput` class has a `focus` property whose auto-generated `on_focus` event can be accessed inside the kv language like so:

```
TextInput:
    on_focus: print(args)
```

13.7 Extend canvas

Kv lang can be used to define the canvas instructions of your widget like this:

```
MyWidget:
    canvas:
        Color:
            rgba: 1, .3, .8, .5
        Line:
            points: zip(self.data.x, self.data.y)
```

And they get updated when properties values change.

Of course you can use *canvas.before* and *canvas.after*.

13.8 Referencing Widgets

In a widget tree there is often a need to access/reference other widgets. The Kv Language provides a way to do this using id's. Think of them as class level variables that can only be used in the Kv language. Consider the following:

```
<MyFirstWidget>:
    Button:
        id: f_but
    TextInput:
        text: f_but.state

<MySecondWidget>:
    Button:
        id: s_but
    TextInput:
        text: s_but.state
```

An `id` is limited in scope to the rule it is declared in, so in the code above `s_but` can not be accessed outside the `<MySecondWidget>` rule.

Warning: When assigning a value to `id`, remember that the value isn't a string. There are no quotes: good -> `id: value`, bad -> `id: 'value'`

An `id` is a `weakref` to the widget and not the widget itself. As a consequence, storing the `id` is not sufficient to keep the widget from being garbage collected. To demonstrate:

```
<MyWidget>:
    label_widget: label_widget
    Button:
        text: 'Add Button'
        on_press: root.add_widget(label_widget)
    Button:
        text: 'Remove Button'
        on_press: root.remove_widget(label_widget)
    Label:
        id: label_widget
        text: 'widget'
```

Although a reference to `label_widget` is stored in `MyWidget`, it is not sufficient to keep the object alive once other references have been removed because it's only a `weakref`. Therefore, after the remove button is clicked (which removes any direct reference to the widget) and the window is resized (which calls the garbage collector resulting in the deletion of `label_widget`), when the add button is clicked to add the widget back, a `ReferenceError: weakly-referenced object no longer exists` will be thrown.

To keep the widget alive, a direct reference to the `label_widget` widget must be kept. This is achieved using `id.__self__` or `label_widget.__self__` in this case. The correct way to do this would be:

```
<MyWidget>:
    label_widget: label_widget.__self__
```

13.9 Accessing Widgets defined inside Kv lang in your Python code

Consider the code below in `my.kv`:

```
<MyFirstWidget>:
    # both these variables can be the same name and this doesn't lead to
    # an issue with uniqueness as the id is only accessible in kv.
    txt_inpt: txt_inpt
    Button:
        id: f_but
    TextInput:
        id: txt_inpt
        text: f_but.state
        on_text: root.check_status(f_but)
```

In `myapp.py`:

```
...
class MyFirstWidget(BoxLayout):

    txt_inpt = ObjectProperty(None)

    def check_status(self, btn):
        print('button state is: {state}'.format(state=btn.state))
```

(continues on next page)

(continued from previous page)

```
print('text input text is: {txt}'.format(txt=self.txt_inpt))
...
```

`txt_inpt` is defined as a *ObjectProperty* initialized to *None* inside the Class.

```
txt_inpt = ObjectProperty(None)
```

At this point `self.txt_inpt` is *None*. In Kv lang this property is updated to hold the instance of the `TextInput` referenced by the id `txt_inpt`:

```
txt_inpt: txt_inpt
```

From this point onwards, `self.txt_inpt` holds a reference to the widget identified by the id `txt_inpt` and can be used anywhere in the class, as in the function `check_status`. In contrast to this method you could also just pass the *id* to the function that needs to use it, like in case of `f_but` in the code above.

There is a simpler way to access objects with *id* tags in Kv using the *ids* lookup object. You can do this as follows:

```
<Marvel>
Label:
    id: loki
    text: 'loki: I AM YOUR GOD!'
Button:
    id: hulk
    text: "press to smash loki"
    on_release: root.hulk_smash()
```

In your Python code:

```
class Marvel(BoxLayout):

    def hulk_smash(self):
        self.ids.hulk.text = "hulk: puny god!"
        self.ids["loki"].text = "loki: >_<!!!" # alternative syntax
```

When your kv file is parsed, kivy collects all the widgets tagged with id's and places them in this `self.ids` dictionary type property. That means you can also iterate over these widgets and access them dictionary style:

```
for key, val in self.ids.items():
    print("key={0}, val={1}".format(key, val))
```

Note: Although the `self.ids` method is very concise, it is generally regarded as 'best practice' to use the `ObjectProperty`. This creates a direct reference, provides faster access and is more explicit.

13.10 Dynamic Classes

Consider the code below:

```
<MyWidget>:
    Button:
        text: "Hello world, watch this text wrap inside the button"
        text_size: self.size
```

(continues on next page)

(continued from previous page)

```
        font_size: '25sp'
        markup: True
    Button:
        text: "Even absolute is relative to itself"
        text_size: self.size
        font_size: '25sp'
        markup: True
    Button:
        text: "Repeating the same thing over and over in a comp = fail"
        text_size: self.size
        font_size: '25sp'
        markup: True
    Button:
```

Instead of having to repeat the same values for every button, we can just use a template instead, like so:

```
<MyBigButt@Button>:
    text_size: self.size
    font_size: '25sp'
    markup: True

<MyWidget>:
    MyBigButt:
        text: "Hello world, watch this text wrap inside the button"
    MyBigButt:
        text: "Even absolute is relative to itself"
    MyBigButt:
        text: "repeating the same thing over and over in a comp = fail"
    MyBigButt:
```

This class, created just by the declaration of this rule, inherits from the Button class and allows us to change default values and create bindings for all its instances without adding any new code on the Python side.

13.11 Re-using styles in multiple widgets

Consider the code below in my.kv:

```
<MyFirstWidget>:
    Button:
        on_press: root.text(txt_inpt.text)
    TextInput:
        id: txt_inpt

<MySecondWidget>:
    Button:
        on_press: root.text(txt_inpt.text)
    TextInput:
        id: txt_inpt
```

In myapp.py:

```
class MyFirstWidget(BoxLayout):

    def text(self, val):
        print('text input text is: {txt}'.format(txt=val))
```

(continues on next page)

```
class MySecondWidget(BoxLayout):

    writing = StringProperty('')

    def text(self, val):
        self.writing = val
```

Because both classes share the same .kv style, this design can be simplified if we reuse that style for both widgets. You can do this in .kv as follows. In my.kv:

```
<MyFirstWidget,MySecondWidget>:
    Button:
        on_press: root.text(txt_inpt.text)
    TextInput:
        id: txt_inpt
```

By separating the class names with a comma, all the classes listed in the declaration will have the same kv properties.

13.12 Designing with the Kivy Language

One of aims of the Kivy language is to **separate the concerns** of presentation and logic. The presentation (layout) side is addressed by your kv file and the logic by your .py file.

13.12.1 The code goes in py files

Let's start with a little example: a Python file named *main.py*:

```
import kivy
kivy.require('1.0.5')

from kivy.uix.floatlayout import FloatLayout
from kivy.app import App
from kivy.properties import ObjectProperty, StringProperty

class Controller(FloatLayout):
    '''Create a controller that receives a custom widget from the kv lang file.

    Add an action to be called from the kv lang file.
    '''
    label_wid = ObjectProperty()
    info = StringProperty()

    def do_action(self):
        self.label_wid.text = 'My label after button press'
        self.info = 'New info text'

class ControllerApp(App):

    def build(self):
        return Controller(info='Hello world')
```



```
if __name__ == '__main__':
    ControllerApp().run()
```

In this example, we are creating a `Controller` class with 2 properties:

- `info` for receiving some text
- `label_wid` for receiving the label widget

In addition, we are creating a `do_action()` method that will use both of these properties. It will change the `info` text and change text in the `label_wid` widget.

13.12.2 The layout goes in controller.kv

Executing this application without a corresponding `.kv` file will work, but nothing will be shown on the screen. This is expected, because the `Controller` class has no widgets in it, it's just a `FloatLayout`. We can create the UI around the `Controller` class in a file named `controller.kv`, which will be loaded when we run the `ControllerApp`. How this is done and what files are loaded is described in the `kivy.app.App.load_kv()` method.

```
1 #:kivy 1.0
2
3 <Controller>:
4     label_wid: my_custom_label
5
6     BoxLayout:
7         orientation: 'vertical'
8         padding: 20
9
10        Button:
11            text: 'My controller info is: ' + root.info
12            on_press: root.do_action()
13
14        Label:
15            id: my_custom_label
16            text: 'My label before button press'
```

One label and one button in a vertical `BoxLayout`. Seems very simple. There are 3 things going on here:

1. Using data from the `Controller`. As soon as the `info` property is changed in the controller, the expression `text: 'My controller info is: ' + root.info` will automatically be re-evaluated, changing the text in the `Button`.
2. Giving data to the `Controller`. The expression `id: my_custom_label` is assigning the created `Label` the id of `my_custom_label`. Then, using `my_custom_label` in the expression `label_wid: my_custom_label` gives the instance of that `Label` widget to your `Controller`.
3. Creating a custom callback in the `Button` using the `Controller`'s `on_press` method.
 - `root` and `self` are reserved keywords, useable anywhere. `root` represents the top widget in the rule and `self` represents the current widget.
 - You can use any id declared in the rule the same as `root` and `self`. For example, you could do this in the `on_press()`:

```
Button:  
    on_press: root.do_action(); my_custom_label.font_size = 18
```

And that's that. Now when we run *main.py*, *controller.kv* will be loaded so that the **Button** and **Label** will show up and respond to our touch events.

Integrating with other Frameworks

New in version 1.0.8.

14.1 Using Twisted inside Kivy

Note: You can use the `kivy.support.install_twisted_reactor` function to install a twisted reactor that will run inside the kivy event loop.

Any arguments or keyword arguments passed to this function will be passed on the `threadedselect` reactors `interleave` function. These are the arguments one would usually pass to twisted's `reactor.startRunning`

Warning: Unlike the default twisted reactor, the installed reactor will not handle any signals unless you set the `'installSignalHandlers'` keyword argument to 1 explicitly. This is done to allow kivy to handle the signals as usual, unless you specifically want the twisted reactor to handle the signals (e.g. SIGINT).

The kivy examples include a small example of a twisted server and client. The server app has a simple twisted server running and logs any messages. The client app can send messages to the server and will print its message and the response it got. The examples are based mostly on the simple Echo example from the twisted docs, which you can find here:

- http://twistedmatrix.com/documents/current/_downloads/simple serv.py
- http://twistedmatrix.com/documents/current/_downloads/simple client.py

To try the example, run `echo_server_app.py` first, and then launch `echo_client_app.py`. The server will reply with simple echo messages to anything the client app sends when you hit enter after typing something in the textbox.

14.1.1 Server App

```
# install_twisted_reactor must be called before importing and using the reactor
from kivy.support import install_twisted_reactor

install_twisted_reactor()

from twisted.internet import reactor
from twisted.internet import protocol

class EchoServer(protocol.Protocol):
    def dataReceived(self, data):
        response = self.factory.app.handle_message(data)
        if response:
            self.transport.write(response)

class EchoServerFactory(protocol.Factory):
    protocol = EchoServer

    def __init__(self, app):
        self.app = app

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

class TwistedServerApp(App):
    label = None

    def build(self):
        self.label = Label(text="server started\n")
        reactor.listenTCP(8000, EchoServerFactory(self))
        return self.label

    def handle_message(self, msg):
        msg = msg.decode('utf-8')
        self.label.text = "received: {}\n".format(msg)

        if msg == "ping":
            msg = "Pong"
        if msg == "plop":
            msg = "Kivy Rocks!!!"
        self.label.text += "responded: {}\n".format(msg)
        return msg.encode('utf-8')

if __name__ == '__main__':
    TwistedServerApp().run()
```

14.1.2 Client App

```
# install_twisted_reactor must be called before importing the reactor
from __future__ import unicode_literals
```

(continues on next page)

```

from kivy.support import install_twisted_reactor

install_twisted_reactor()

# A Simple Client that send messages to the Echo Server
from twisted.internet import reactor, protocol

class EchoClient(protocol.Protocol):
    def connectionMade(self):
        self.factory.app.on_connection(self.transport)

    def dataReceived(self, data):
        self.factory.app.print_message(data.decode('utf-8'))

class EchoClientFactory(protocol.ClientFactory):
    protocol = EchoClient

    def __init__(self, app):
        self.app = app

    def startedConnecting(self, connector):
        self.app.print_message('Started to connect.')

    def clientConnectionLost(self, connector, reason):
        self.app.print_message('Lost connection.')

    def clientConnectionFailed(self, connector, reason):
        self.app.print_message('Connection failed.')

from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.boxlayout import BoxLayout

# A simple kivy App, with a textbox to enter messages, and
# a large label to display all the messages received from
# the server
class TwistedClientApp(App):
    connection = None
    textbox = None
    label = None

    def build(self):
        root = self.setup_gui()
        self.connect_to_server()
        return root

    def setup_gui(self):
        self.textbox = TextInput(size_hint_y=.1, multiline=False)
        self.textbox.bind(on_text_validate=self.send_message)
        self.label = Label(text='connecting...\n')
        layout = BoxLayout(orientation='vertical')
        layout.add_widget(self.label)
        layout.add_widget(self.textbox)

```

```
    return layout

    def connect_to_server(self):
        reactor.connectTCP('localhost', 8000, EchoClientFactory(self))

    def on_connection(self, connection):
        self.print_message("Connected successfully!")
        self.connection = connection

    def send_message(self, *args):
        msg = self.textbox.text
        if msg and self.connection:
            self.connection.write(msg.encode('utf-8'))
            self.textbox.text = ""

    def print_message(self, msg):
        self.label.text += "{}\n".format(msg)

if __name__ == '__main__':
    TwistedClientApp().run()
```

Packaging your application

15.1 Create a package for Windows

Note: This document only applies for kivy 1.9.1 and greater.

Packaging your application for the Windows platform can only be done inside the Windows OS. The following process has been tested on Windows with the Kivy **wheels** installation, see at the end for alternate installations.

The package will be either 32 or 64 bits depending on which version of Python you ran it with.

15.1.1 Requirements

- Latest Kivy (installed as described in `installation_windows`).
- PyInstaller 3.1+ (`pip install --upgrade pyinstaller`).

15.2 PyInstaller default hook

This section applies to PyInstaller (≥ 3.1) that includes the kivy hooks. To overwrite the default hook the following examples need to be slightly modified. See *Overwriting the default hook*.

15.2.1 Packaging a simple app

For this example, we'll package the **touchtracer** example project and embed a custom icon. The location of the kivy examples is, when using the wheels, installed to `python\share\kivy-examples` and when using the github source code installed as `kivy\examples`. We'll just refer to the full path leading to the examples as `examples-path`. The touchtracer example is in `examples-path\demo\touchtracer` and the main file is named `main.py`.

1. Open your command line shell and ensure that python is on the path (i.e. `python` works).
2. Create a folder into which the packaged app will be created. For example create a TouchApp folder and [change to that directory](#) with e.g. `cd TouchApp`. Then type:

```
python -m PyInstaller --name touchtracer examples-path\demo\touchtracer\main.  
→py
```

You can also add an *icon.ico* file to the application folder in order to create an icon for the executable. If you don't have a .ico file available, you can convert your *icon.png* file to ico using the web app [ConvertICO](#). Save the *icon.ico* in the touchtracer directory and type:

```
python -m PyInstaller --name touchtracer --icon examples-  
→path\demo\touchtracer\icon.ico examples-path\demo\touchtracer\main.py
```

For more options, please consult the [PyInstaller Manual](#).

3. The spec file will be `touchtracer.spec` located in TouchApp. Now we need to edit the spec file to add the dependencies hooks to correctly build the exe. Open the spec file with your favorite editor and add these lines at the beginning of the spec (assuming sdl2 is used, the default now):

```
from kivy_deps import sdl2, glew
```

Then, find `COLLECT()` and add the data for touchtracer (*touchtracer.kv*, *particle.png*, ...): Change the line to add a `Tree()` object, e.g. `Tree('examples-path\\demo\\touchtracer\\')`. This Tree will search and add every file found in the touchtracer directory to your final package.

To add the dependencies, before the first keyword argument in `COLLECT` add a Tree object for every path of the dependencies. E.g. `*[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins)]` so it'll look something like:

```
coll = COLLECT(exe, Tree('examples-path\\demo\\touchtracer\\'),  
                a.binaries,  
                a.zipfiles,  
                a.datas,  
                *[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins)],  
                strip=False,  
                upx=True,  
                name='touchtracer')
```

4. Now we build the spec file in TouchApp with:

```
python -m PyInstaller touchtracer.spec
```

5. The compiled package will be in the *TouchApp\dist\touchtracer* directory.

15.2.2 Packaging a video app with gstreamer

Following we'll slightly modify the example above to package a app that uses gstreamer for video. We'll use the `videoplayer` example found at `examples-path\widgets\videoplayer.py`. Create a folder somewhere called `VideoPlayer` and on the command line change your current directory to that folder and do:

```
python -m PyInstaller --name gstvideo examples-path\widgets\videoplayer.py
```

to create the `gstvideo.spec` file. Edit as above and this time include the gstreamer dependency as well:


```
from kivy_deps import sdl2, glew, gstreamer
```

and add the `Tree()` to include the video files, e.g. `Tree('examples-path\\widgets')` as well as the gstreamer dependencies so it should look something like:

```
coll = COLLECT(exe, Tree('examples-path\\widgets'),
               a.binaries,
               a.zipfiles,
               a.datas,
               *[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins + gstreamer.dep_
→bins)],
               strip=False,
               upx=True,
               name='gstvideo')
```

Then build the spec file in `VideoPlayer` with:

```
python -m PyInstaller gstvideo.spec
```

and you should find `gstvideo.exe` in `VideoPlayer\dist\gstvideo`, which when run will play a video.

Note: If you're using Pygame and need PyGame in your packaging app, you'll have to add the following code to your spec file due to kivy issue #1638. After the imports add the following:

```
def getResource(identifier, *args, **kwargs):
    if identifier == 'pygame_icon.tiff':
        raise IOError()
    return _original_getResource(identifier, *args, **kwargs)

import pygame.pkgdata
_original_getResource = pygame.pkgdata.getResource
pygame.pkgdata.getResource = getResource
```

15.3 Overwriting the default hook

15.3.1 Including/excluding video and audio and reducing app size

PyInstaller includes a hook for kivy that by default adds **all** the core modules used by kivy, e.g. audio, video, spelling etc (you still need to package the gstreamer dlls manually with `Tree()` - see the example above) and their dependencies. If the hook is not installed or to reduce app size some of these modules may be excluded, e.g. if no audio/video is used, with an alternative hook.

Kivy provides the alternate hook at `hookspath()`. In addition, if and only if PyInstaller doesn't have the default hooks `runtime_hooks()` must also be provided. When overwriting the hook, the latter one typically is not required to be overwritten.

The alternate `hookspath()` hook does not include any of the kivy providers. To add them, they have to be added with `get_deps_minimal()` or `get_deps_all()`. See their documentation and `pyinstaller_hooks` for more details. But essentially, `get_deps_all()` add all the providers like in the default hook while `get_deps_minimal()` only adds those that are loaded when the app is run. Each method provides a list of hidden kivy imports and excluded imports that can be passed on to `Analysis`.

One can also generate a alternate hook which literally lists every kivy provider module and those not required can be commented out. See `pyinstaller_hooks`.

To use the the alternate hooks with the examples above modify as following to add the hooks with `hookspath()` and `runtime_hooks` (if required) and `**get_deps_minimal()` or `**get_deps_all()` to specify the providers.

For example, add the import statement:

```
from kivy.tools.packaging.pyinstaller_hooks import get_deps_minimal, get_deps_all,   
hookspath, runtime_hooks
```

and then modify `Analysis` as follows:

```
a = Analysis(['examples-path\\demo\\touchtracer\\main.py'],  
            ...  
            hookspath=hookspath(),  
            runtime_hooks=runtime_hooks(),  
            ...  
            **get_deps_all())
```

to include everything like the default hook. Or:

```
a = Analysis(['examples-path\\demo\\touchtracer\\main.py'],  
            ...  
            hookspath=hookspath(),  
            runtime_hooks=runtime_hooks(),  
            ...  
            **get_deps_minimal(video=None, audio=None))
```

e.g. to exclude the audio and video providers and for the other core modules only use those loaded.

The key points is to provide the alternate `hookspath()` which does not list by default all the kivy providers and instead manually to `hiddenimports` add the required providers while removing the undesired ones (audio and video in this example) with `get_deps_minimal()`.

15.3.2 Alternate installations

The previous examples used e.g. `*[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins + gstreamer.dep_bins)]`, to make PyInstaller add all the dlls used by these dependencies. If kivy was not installed using the wheels method these commands will not work and e.g. `kivy_deps.sdl2` will fail to import. Instead, one must find the location of these dlls and manually pass them to the `Tree` class in a similar fashion as the example.

15.4 Create a package for Android

You can create a package for android using the [python-for-android](#) project. This page explains how to download and use it directly on your own machine (see [Packaging with python-for-android](#)) or use the [Buildozer](#) tool to automate the entire process. You can also see [Packaging your application for the Kivy Launcher](#) to run kivy programs without compiling them.

For new users, we recommend using [Buildozer](#) as the easiest way to make a full APK. You can also run your Kivy app without a compilation step with the [Kivy Launcher](#) app.

Kivy applications can be [released on an Android market](#) such as the Play store, with a few extra steps to create a fully signed APK.

The Kivy project includes tools for accessing Android APIs to accomplish vibration, sensor access, texting etc. These, along with information on debugging on the device, are documented at the [main Android page](#).

15.4.1 Buildozer

Buildozer is a tool that automates the entire build process. It downloads and sets up all the prerequisites for python-for-android, including the android SDK and NDK, then builds an apk that can be automatically pushed to the device.

Buildozer currently works only in Linux, and is an alpha release, but it already works well and can significantly simplify the apk build.

You can get buildozer at <https://github.com/kivy/buildozer>:

```
git clone https://github.com/kivy/buildozer.git
cd buildozer
sudo python setup.py install
```

This will install buildozer in your system. Afterwards, navigate to your project directory and run:

```
buildozer init
```

This creates a *buildozer.spec* file controlling your build configuration. You should edit it appropriately with your app name etc. You can set variables to control most or all of the parameters passed to python-for-android.

Install buildozer's [dependencies](#).

Finally, plug in your android device and run:

```
buildozer android debug deploy run
```

to build, push and automatically run the apk on your device.

Buildozer has many available options and tools to help you, the steps above are just the simplest way to build and run your APK. The full documentation is available [here](#). You can also check the Buildozer README at <https://github.com/kivy/buildozer>.

15.4.2 Packaging with python-for-android

You can also package directly with python-for-android, which can give you more control but requires you to manually download parts of the Android toolchain.

See the [python-for-android documentation](#) for full details.

15.4.3 Packaging your application for the Kivy Launcher

The [Kivy launcher](#) is an Android application that runs any Kivy examples stored on your SD Card. To install the Kivy launcher, you must:

1. Go to the [Kivy Launcher page](#) on the Google Play Store
2. Click on Install
3. Select your phone... And you're done!

If you don't have access to the Google Play Store on your phone/tablet, you can download and install the APK manually from <http://kivy.org/#download>.

Once the Kivy launcher is installed, you can put your Kivy applications in the Kivy directory in your external storage directory (often available at `/sdcard` even in devices where this memory is internal), e.g.

```
/sdcard/kivy/<yourapplication>
```

<yourapplication> should be a directory containing:

```
# Your main application file:
main.py
# Some info Kivy requires about your app on android:
android.txt
```

The file *android.txt* must contain:

```
title=<Application Title>
author=<Your Name>
orientation=<portrait|landscape>
```

These options are just a very basic configuration. If you create your own APK using the tools above, you can choose many other settings.

Installation of Examples

Kivy comes with many examples, and these can be a great place to start trying the Kivy launcher. You can run them as below:

```
#. Download the `Kivy demos for Android` <https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/kivy/kivydemo-for-android.zip>`_
#. Unzip the contents and go to the folder `kivydemo-for-android`
#. Copy all the the subfolders here to
```

```
/sdcard/kivy
```

1. Run the launcher and select one of the Pictures, Showcase, Touchtracer, Cymunk or other demos...

15.4.4 Release on the market

If you have built your own APK with Buildozer or with python-for-android, you can create a release version that may be released on the Play store or other Android markets.

To do this, you must run Buildozer with the `release` parameter (e.g. `buildozer android release`), or if using python-for-android use the `--release` option to `build.py`. This creates a release APK in the `bin` directory, which you must properly sign and zipalign. The procedure for doing this is described in the Android documentation at <https://developer.android.com/studio/publish/app-signing.html#signing-manually> - all the necessary tools come with the Android SDK.

15.4.5 Targeting Android

Kivy is designed to operate identically across platforms and as a result, makes some clear design decisions. It includes its own set of widgets and by default, builds an APK with all the required core dependencies and libraries.

It is possible to target specific Android features, both directly and in a (somewhat) cross-platform way. See the *Using Android APIs* section of the [Kivy on Android documentation](#) for more details.

15.5 Kivy on Android

You can run Kivy applications on Android, on (more or less) any device with OpenGL ES 2.0 (Android 2.2 minimum). This is standard on modern devices; Google reports the requirement is met by [99.9% of devices](#).

Kivy APKs are normal Android apps that you can distribute like any other, including on stores like the Play store. They behave properly when paused or restarted, may utilise Android services and have access to most of the normal java API as described below.

Follow the instructions below to learn how to *package your app for Android*, *debug your code on the device*, and *use Android APIs* such as for vibration and reading sensors.

15.5.1 Package for Android

The Kivy project provides all the necessary tools to package your app on Android, including building your own standalone APK that may be distributed on a market like the Play store. This is covered fully in the [Create a package for Android](#) documentation.

15.5.2 Debugging your application on the Android platform

You can view the normal output of your code (stdout, stderr), as well as the normal Kivy logs, through the Android logcat stream. This is accessed through adb, provided by the [Android SDK](#). You may need to enable adb in your device's developer options, then connect your device to your computer and run:

```
adb logcat
```

You'll see all the logs including your stdout/stderr and Kivy logger.

If you packaged your app with Buildozer, the *adb* tool may not be in your *\$PATH* and the above command may not work. You can instead run:

```
buildozer android logcat
```

to run the version installed by Buildozer, or find the SDK tools at *\$HOME/.buildozer/android/platform*.

You can also run and debug your application using the [Kivy Launcher](#). If you run your application this way, you will find log files inside the *"/.kivy/logs"* sub-folder within your application folder.

15.5.3 Using Android APIs

Although Kivy is a Python framework, the Kivy project maintains tools to easily use the normal java APIs, for everything from vibration to sensors to sending messages through SMS or email.

For new users, we recommend using [Plyer](#). For more advanced access or for APIs not currently wrapped, you can use [Pyjnius](#) directly. Kivy also supplies an *android module* for basic Android functionality.

User contributed Android code and examples are available on the [Kivy wiki](#).

Plyer

Plyer is a pythonic, platform-independent API to use features commonly found on various platforms, particularly mobile ones. The idea is that your app can simply call a Plyer function, such as to present a notification to the user, and Plyer will take care of doing so in the right way regardless of the platform or operating system. Internally, Plyer uses Pyjnius (on Android), Pyobjus (on iOS) and some platform specific APIs on desktop platforms.

For instance, the following code would make your Android device vibrate, or raise a `NotImplementedError` that you can handle appropriately on other platforms such as desktops that don't have appropriate hardware::

```
from plyer import vibrator
vibrator.vibrate(10) # vibrate for 10 seconds
```

Plyer's list of supported APIs is growing quite quickly, you can see the full list in the Plyer [README](#).

Pyjnius

Pyjnius is a Python module that lets you access java classes directly from Python, automatically converting arguments to the right type, and letting you easily convert the java results to Python.

Pyjnius can be obtained from [github](#), and has its [own documentation](#).

Here is a simple example showing Pyjnius' ability to access the normal Android vibration API, the same result of the plyer code above:

```
# 'autoclass' takes a java class and gives it a Python wrapper
from jnius import autoclass

# Context is a normal java class in the Android API
Context = autoclass('android.content.Context')

# PythonActivity is provided by the Kivy bootstrap app in python-for-android
PythonActivity = autoclass('org.renpy.android.PythonActivity')

# The PythonActivity stores a reference to the currently running activity
# We need this to access the vibrator service
activity = PythonActivity.mActivity

# This is almost identical to the java code for the vibrator
vibrator = activity.getSystemService(Context.VIBRATOR_SERVICE)

vibrator.vibrate(10000) # The value is in milliseconds - this is 10s
```

This code directly follows the java API functions to call the vibrator, with Pyjnius automatically translating the api to Python code and our calls back to the equivalent java. It is much more verbose and java-like than Plyer's version, for no benefit in this case, though Plyer does not wrap every API available to Pyjnius.

Pyjnius also has powerful abilities to implement java interfaces, which is important for wrapping some APIs, but these are not documented here - you can see Pyjnius' [own documentation](#).

Android module

Python-for-android includes a python module (actually cython wrapping java) to access a limited set of Android APIs. This has been largely superseded by the more flexible Pyjnius and Plyer as above, but

may still occasionally be useful. The available functions are given in the [python-for-android documentation](#).

This includes code for billing/IAP and creating/accessing Android services, which is not yet available in the other tools above.

15.5.4 Status of the Project and Tested Devices

These sections previously described the existence of Kivy's Android build tools, with their limitations and some devices that were known to work.

The Android tools are now quite stable, and should work with practically any device; our minimum requirements are OpenGL ES 2.0 and Android 2.2. These are very common now - Kivy has even been run on an Android smartwatch!

A current technical limitation is that the Android build tools compile only ARM APKs, which will not run on Android devices with x86 processors (these are currently rare). This should be added soon.

As Kivy works fine on most devices, the list of supported phones/tablets has been retired - all Android devices are likely to work if they meet the conditions above.

15.6 Creating packages for OS X

Note: This guide describes multiple ways for packaging Kivy applications. Packaging with PyInstaller is recommended for general use.

15.6.1 Using PyInstaller and Homebrew

Note: Package your app on the oldest OS X version you want to support.

Complete guide

1. Install [Homebrew](#)
2. Install Python:

```
$ brew install python
```

Note: To use Python 3, `brew install python3` and replace `pip` with `pip3` in the guide below.

3. (Re)install your dependencies with `--build-bottle` to make sure they can be used on other machines:

```
$ brew reinstall --build-bottle sdl2 sdl2_image sdl2_ttf sdl2_mixer
```

Note: If your project depends on GStreamer or other additional libraries (re)install them with `--build-bottle` as described [below](#).

4. Install Cython and Kivy:

```
$ pip install Cython==0.29.14
$ pip install -U kivy
```

5. Install PyInstaller:

```
$ pip install -U pyinstaller
```

6. Package your app using the path to your main.py:

```
$ pyinstaller -y --clean --windowed --name touchtracer \
--exclude-module _tkinter \
--exclude-module Tkinter \
--exclude-module enchant \
--exclude-module twisted \
/usr/local/share/kivy-examples/demo/touchtracer/main.py
```

Note: This will not yet copy additional image or sound files. You would need to adapt the created .spec file for that.

Editing the spec file

The specs file is named *touchtracer.spec* and is located in the directory where you ran the pyinstaller command.

You need to change the *COLLECT()* call to add the data of touchtracer (*touchtracer.kv*, *particle.png*, ...). Change the line to add a *Tree()* object. This *Tree* will search and add every file found in the touchtracer directory to your final package. Your *COLLECT* section should look something like this:

```
coll = COLLECT(exe, Tree('/usr/local/share/kivy-examples/demo/touchtracer/'),
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=None,
               upx=True,
               name='touchtracer')
```

This will add the required hooks so that PyInstaller gets the required Kivy files. We are done. Your spec is ready to be executed.

Build the spec and create a DMG

1. Open a console.
2. Go to the PyInstaller directory, and build the spec:

```
$ pyinstaller -y --clean --windowed touchtracer.spec
```

3. Run:

```
$ pushd dist
$ hdiutil create ./Touchtracer.dmg -srcfolder touchtracer.app -ov
$ popd
```

4. You will now have a *Touchtracer.dmg* available in the *dist* directory.

Additional Libraries

GStreamer

If your project depends on GStreamer:

```
$ brew reinstall --build-bottle gstreamer gst-plugins-{base,good,bad,ugly}
```

Note: If your Project needs Ogg Vorbis support be sure to add the `--with-libvorbis` option to the command above.

If you are using Python from Homebrew you will also need the following step until [this pull request](#) gets merged:

```
$ brew reinstall --with-python --build-bottle https://github.com/cbenhagen/  
→homebrew/raw/patch-3/Library/Formula/gst-python.rb
```

15.6.2 Using PyInstaller without Homebrew

First install Kivy and its dependencies without using Homebrew as mentioned here <http://kivy.org/docs/installation/installation.html#development-version>.

Once you have kivy and its deps installed, you need to install PyInstaller.

Let's assume we use a folder like *testpackaging*:

```
cd testpackaging  
git clone http://github.com/pyinstaller/pyinstaller
```

Create a file named *touchtracer.spec* in this directory and add the following code to it:

```
# -*- mode: python -*-  
  
block_cipher = None  
from kivy.tools.packaging.pyinstaller_hooks import get_deps_all, hookspath,  
→runtime_hooks  
  
a = Analysis(['/path/to/your/folder/containing/examples/demo/touchtracer/main.py'],  
             pathex=['/path/to/your/folder/containing/testpackaging'],  
             binaries=None,  
             win_no_prefer_redirects=False,  
             win_private_assemblies=False,  
             cipher=block_cipher,  
             hookspath=hookspath(),  
             runtime_hooks=runtime_hooks(),  
             **get_deps_all())  
pyz = PYZ(a.pure, a.zipped_data,  
          cipher=block_cipher)  
exe = EXE(pyz,  
          a.scripts,  
          exclude_binaries=True,  
          name='touchtracer',  
          debug=False,  
          strip=False,  
          upx=True,  
          console=False )
```

(continues on next page)

(continued from previous page)

```
coll = COLLECT(exe, Tree('../kivy/examples/demo/touchtracer/'),
                Tree('/Library/Frameworks/SDL2_ttf.framework/Versions/A/Frameworks/
→FreeType.framework'),
                a.binaries,
                a.zipfiles,
                a.datas,
                strip=False,
                upx=True,
                name='touchtracer')
app = BUNDLE(coll,
              name='touchtracer.app',
              icon=None,
              bundle_identifier=None)
```

Change the paths with your relevant paths:

```
a = Analysis(['/path/to/your/folder/containing/examples/demo/touchtracer/main.py'],
             pathex=['/path/to/your/folder/containing/testpackaging'],
             ...)
coll = COLLECT(exe, Tree('../kivy/examples/demo/touchtracer/'),
```

Then run the following command:

```
pyinstaller/pyinstaller.py touchtracer.spec
```

Replace *touchtracer* with your app where appropriate. This will give you a <yourapp>.app in the dist/ folder.

15.6.3 Using Buildozer

```
pip install git+http://github.com/kivy/buildozer cd /to/where/I/Want/to/package buildozer init
```

Note: Packaging Kivy applications with the following method must be done inside OS X, 32-bit platforms are no longer supported.

Edit the buildozer.spec and add the details for your app. Dependencies can be added to the *requirements=* section.

By default the kivy version specified in the requirements is ignored.

If you have a Kivy.app at /Applications/Kivy.app then that is used, for packaging. Otherwise the latest build from kivy.org using Kivy master will be downloaded and used.

If you want to package for python 3.x.x simply download the package named Kivy3.7z from the download section of kivy.org and extract it to Kivy.app in /Applications, then run:

```
buildozer osx debug
```

Once the app is packaged, you might want to remove unneeded packages like gstreamer, if you don't need video support. Same logic applies for other things you do not use, just reduce the package to its minimal state that is needed for the app to run.

As an example we are including the showcase example packaged using this method for both Python 2 (9.xMB) and 3 (15.xMB), you can find the packages here: https://drive.google.com/drive/folders/0B1WO07-OL50_aIFzSXJUajBFdnc.

That's it. Enjoy!

Buildozer right now uses the Kivy SDK to package your app. If you want to control more details about your app than buildozer currently offers then you can use the SDK directly, as detailed in the section below.

15.6.4 Using the Kivy SDK

Note: Kivy.app is not available for download at the moment. For details, see [this](#) issue.

Note: Packaging Kivy applications with the following method must be done inside OS X, 32-bit platforms are no longer supported.

Since version 1.9.0, Kivy is released for the OS X platform in a self-contained, portable distribution.

Apps can be packaged and distributed with the Kivy SDK using the method described below, making it easier to include frameworks like SDL2 and GStreamer.

1. Make sure you have the unmodified Kivy SDK (Kivy.app) from the download page.
2. Run the following commands:

```
> mkdir packaging
> cd packaging
packaging> git clone https://github.com/kivy/kivy-sdk-packager
packaging> cd kivy-sdk-packager/osx
osx> cp -a /Applications/Kivy.app ./Kivy.App
```

Note: This step above is important, you have to make sure to preserve the paths and permissions. A command like `cp -rf` will copy but make the app unusable and lead to error later on.

3. Now all you need to do is to include your compiled app in the Kivy.app by running the following command:

```
osx> ./package-app.sh /path/to/your/<app_folder_name>/
```

Where `<app_folder_name>` is the name of your app.

This copies Kivy.app to `<app_folder_name>.app` and includes a compiled copy of your app into this package.

4. That's it, your self-contained package is ready to be deployed! You can now further customize your app as described below.

Installing modules

Kivy package on osx uses its own virtual env that is activated when you run your app using *kivy* command. To install any module you need to install the module like so:

```
$ kivy -m pip install <modulename>
```

Where are the modules/files installed?

Inside the portable venv within the app at:

```
Kivy.app/Contents/Resources/venv/
```

If you install a module that installs a binary for example like kivy-garden That binary will be only available from the venv above, as in after you do:

```
kivy -m pip install kivy-garden
```

The garden lib will be only available when you activate this env.

```
source /Applications/Kivy.app/Contents/Resources/venv/bin/activate garden install  
mapview deactivate
```

To install binary files

Just copy the binary to the Kivy.app/Contents/Resources/venv/bin/ directory.

To include other frameworks

Kivy.app comes with SDL2 and Gstreamer frameworks provided. To include frameworks other than the ones provided do the following:

```
git clone http://github.com/tito/osxrelocator  
export PYTHONPATH=~/path/to/osxrelocator  
cd Kivy.app  
python -m osxrelocator -r . /Library/Frameworks/<Framework_name>.framework/ \  
@executable_path/../Frameworks/<Framework_name>.framework/
```

Do not forget to replace <Framework_name> with your framework. This tool *osxrelocator* essentially changes the path for the libs in the framework such that they are relative to the executable within the .app, making the Framework portable with the .app.

Shrinking the app size

The app has a considerable size right now, however the unneeded parts can be removed from the package.

For example if you don't use GStreamer, simply remove it from YourApp.app/Contents/Frameworks. Similarly you can remove the examples folder from /Applications/Kivy.app/Contents/Resources/kivy/examples/ or kivy/tools, kivy/docs etc.

This way the package can be made to only include the parts that are needed for your app.

Adjust settings

Icons and other settings of your app can be changed by editing YourApp/Contents/info.plist to suit your needs.

Create a DMG

To make a DMG of your app use the following command:

```
osx> ./create-osx-dmg.sh YourApp.app
```

Note the lack of / at the end. This should give you a compressed dmg that will further shrink the size of your distributed app.

15.7 Create a package for IOS

Note: Currently, kivy-ios builds packages with Python 2.7 and 3.7.

The overall process for creating a package for IOS can be explained in 4 steps:

1. *Compile the distribution* (python + modules for IOS)
2. *Create an Xcode project* (and link your source code)
3. *Update the Xcode project*
4. *Customize the Xcode project*

15.7.1 Prerequisites

You need to install some dependencies, like Cython, autotools, etc. We encourage you to use [Homebrew](#) to install those dependencies:

```
brew install autoconf automake libtool pkg-config
brew link libtool
sudo easy_install pip
sudo pip install Cython==0.29.14
```

For more detail, see [IOS Prerequisites](#). Just ensure that everything is ok before starting the second step!

15.7.2 Compile the distribution

Open a terminal, and type:

```
$ git clone git://github.com/kivy/kivy-ios
$ cd kivy-ios
$ ./toolchain.py build kivy
```

Most of the python distribution is packed into *python27.zip*. If you experience any issues, please refer to our [user group](#) or the [kivy-ios project page](#).

15.7.3 Create an Xcode project

Before proceeding to the next step, ensure your application entry point is a file named *main.py*.

We provide a script that creates an initial Xcode project to start with. In the command line below, replace *test* with your project name. It must be a name without any spaces or illegal characters:

```
$ ./toolchain.py create <title> <app_directory>
$ ./toolchain.py create Touchtracer ~/code/kivy/examples/demo/touchtracer
```

Note: You must use a fully qualified path to your application directory.

A directory named `<title>-ios` will be created, with an Xcode project in it. You can open the Xcode project:

```
$ open touchtracer-ios/touchtracer.xcodeproj
```

Then click on *Play*, and enjoy.

Note: Everytime you press *Play*, your application directory will be synced to the `<title>-ios/YourApp` directory. Don't make changes in the `-ios` directory directly.

15.7.4 Update the Xcode project

Let's say you want to add numpy to your project but you did not compile it prior to creating your Xcode project. First, ensure it is built:

```
$ ./toolchain.py build numpy
```

Then, update your Xcode project:

```
$ ./toolchain.py update touchtracer-ios
```

All the libraries / frameworks necessary to run all the compiled recipes will be added to your Xcode project.

15.7.5 Customize the Xcode project

There are various ways to customize and configure your app. Please refer to the [kivy-ios](#) documentation for more information.

15.7.6 Known issues

All known issues with packaging for iOS are currently tracked on our [issues](#) page. If you encounter an issue specific to packaging for iOS that isn't listed there, please feel free to file a new issue, and we will get back to you on it.

While most are too technical to be written here, one important known issue is that removing some libraries (e.g. `SDL_Mixer` for audio) is currently not possible because the kivy project requires it. We will fix this and others in future versions.

15.7.7 FAQ

Application quit abnormally!

By default, all the print statements to the console and files are ignored. If you have an issue when running your application, you can activate the log by commenting out this line in `main.m`:

```
putenv("KIVY_NO_CONSOLELOG=1");
```

Then you should see all the Kivy logging on the Xcode console.

How can Apple accept a python app ?

We managed to merge the app binary with all the libraries into a single binary, called libpython. This means all binary modules are loaded beforehand, so nothing is dynamically loaded.

Have you already submitted a Kivy application to the App store ?

Yes, check:

- [Defletouch on iTunes](#),
- [ProcessCraft on iTunes](#)

For a more complete list, visit the [Kivy wiki](#).

15.8 IOS Prerequisites

The following guide assumes:

- XCode 5.1 or above
- OS X 10.9 or above

Your experience may vary with different versions.

15.8.1 Getting started

In order to submit any application to the iTunes store, you will need an [iOS Developer License](#). For testing, you can use a physical device or the XCode iOS emulator.

Please note that in order to test on the device, you need to register these devices and install your “provisioning profile” on them. Please refer to the Apple’s [Getting started](#) guide for more information.

15.8.2 Homebrew

We use the [Homebrew](#) package manager for OSX to install some of the dependencies and tools used by Kivy. It’s a really helpful tool and is an Open Source project hosted on [Github](#).

Due to the nature of package management (complications with versions and Operating Systems), this process can be error prone and cause failures in the build process. The **Missing requirement: <pkg> is not installed!** message is typically such an error.

The first thing is to ensure you have run the following commands:

```
brew install autoconf automake libtool pkg-config mercurial
brew link libtool
brew link mercurial
sudo easy_install pip
sudo pip install Cython==0.29.14
```

If you still receive build errors, check your Homebrew is in a healthy state:

```
brew doctor
```

For further help, please refer to the [Homebrew wiki](#).

The last, final and desperate step to get things working might be to remove Homebrew altogether, get the latest version, install that and then re-install the dependencies.

[How to Uninstall and Remove Homebrew for Mac OSX](#)

Package licensing

Warning: This is not a legally authoritative guide! The Kivy organisation, authors and contributors take no responsibility for any lack of knowledge, information or advice presented here. The guide is merely informative and is meant to protect inexperienced users.

Your code alone may not require including licensing information or copyright notices of other included software, but binaries are something else. When a binary (.exe, .app, .apk, ...) is created, it includes Kivy, its dependencies and other packages that your application uses.

Some of them are licensed in a way that requires including a copyright notice somewhere in your app (or more). Before distributing any of the binaries, please **check all the created files** that don't belong to your source (.dll, .pyd, .so, ...) and include the appropriate copyright notices if required by the license the files belong to. This way you may satisfy licensing requirements of the Kivy deps.

16.1 Dependencies

All of the dependencies will be used at least partially on each platform Kivy supports. You therefore need to comply to their licenses, which mostly requires only pasting a copyright notice in your app and not pretending you wrote the code.

- `docutils`
- `pygments`
- `sdl2`
- `glew`
- `gststreamer` (if used)
- image & audio libraries(e.g. `SDL_mixer` has them)

You'll probably need to check image and audio libraries manually (most begin with `lib`). The `LICENSE*` files that belong to them should be included by PyInstaller, but are not included by `python-for-android` and you need to find them.

16.2 Windows (PyInstaller)

To access some Windows API features, Kivy uses the `pypiwin32` package. This package is released under the [PSF license](#).

16.2.1 Visual Studio Redistributables

Python compiled with Visual Studio (official) includes files from Microsoft and you are only allowed to redistribute them under specific conditions listed in the CRTlicense. You need to include the names of the files and a reworded version of [Py2 CRT license](#) or [Py3 CRT license](#) (depending which interpreter you use) and present these to the end-user of your application in order to satisfy their requirements.

- [List of redistributables](#)

16.2.2 Other libraries

- [zlib](#)

Note: Please add the attributions for other libraries that you *don't use directly* but are present after packaging with e.g. PyInstaller on Windows.

16.3 Linux

Linux has many distributions which means there's no correct guide for all of the distributions. This applies to the RPi too. However, it can be simplified in two ways depending on how you create a package (also with PyInstaller): with or without including binaries.

If the binaries are included, you should check every file (e.g. `.so`) that's not your source and find the license it belongs to. According to that license, you'll probably need to put an attribution into your application or possibly more, depending on the requirements of that license.

If the binaries are not included (which allows packaging your app as e.g. a `.deb` package), there's a *situation bad for your user*. It's up to you to decide whether you satisfy the conditions of other licenses and, for example, include copyright attributions into your app or not.

16.4 Android

As APK is just an archive of files: you can extract files from it and (as in Windows redistributables) check all the files.

`APK/assets/private.mp3/private.mp3/` contains all the included files. Most of them are related to Kivy, Python or your source, but those that aren't need checking.

Known packages:

- [pygame](#) (if `old_toolchain` is used)

- `sqlite3`
- `six`

There are other included libraries, included either by Kivy directly or through Pygame/SDL2, that are located in `APK/lib/armeabi/`. Most of them are related to dependencies or are produced by `python-for-android` and are part of its source (and licensing).

- `libapplication.so`

16.5 Mac

Missing.

16.6 iOS

Missing.

16.7 Avoiding binaries

There might be a way how to avoid this licensing process by avoiding creating a distribution with third-party stuff completely. With Python you can create a module, which is only your code with `__main__.py` + `setup.py` that only lists required dependencies.

This way, you can still distribute your app - your *code* - and you might not need to care about other licenses. The combination of your code and the dependencies could be specified as a “usage” rather than a “distribution”. The responsibility of satisfying licenses, however, most likely transfers to your user, who needs to assemble the environment to even run the module. If you care about your users, you might want to slow down a little and read more about the [consequences](#).

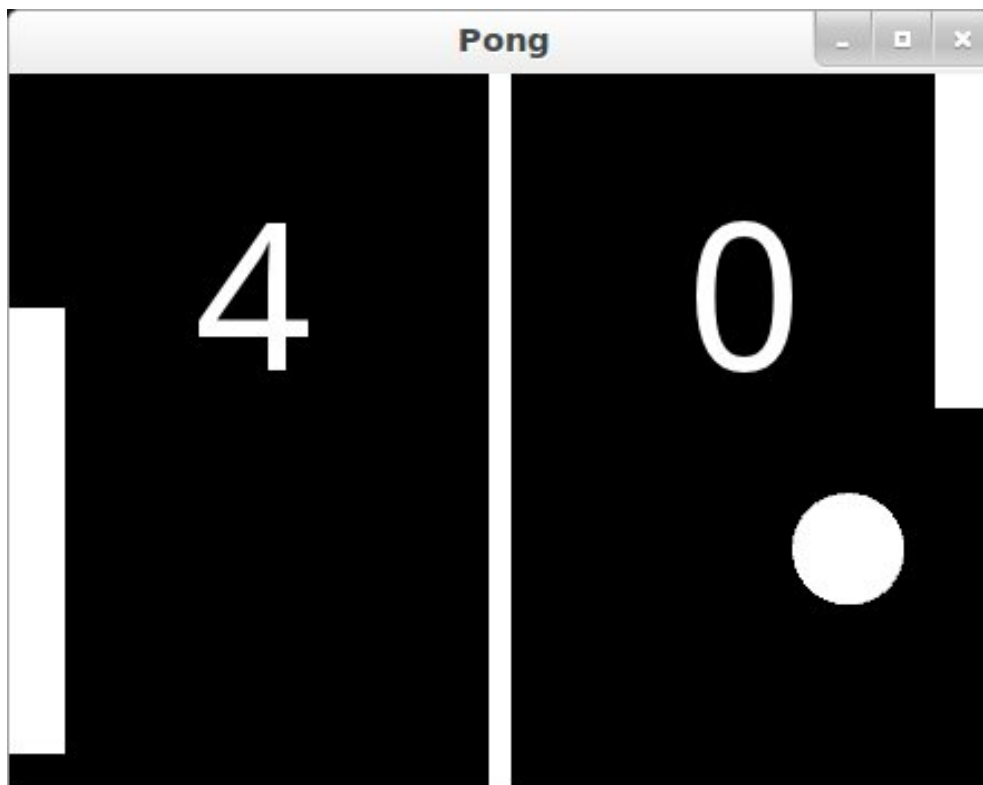
Part III
TUTORIALS

Pong Game Tutorial

17.1 Introduction

Welcome to the Pong tutorial

This tutorial will teach you how to write pong using Kivy. We'll start with a basic application like the one described in the *Create an application* and turn it into a playable pong game, describing each step along the way.



Here is a check list before starting this tutorial:

- You have a working Kivy installation. See the [/installation/installation](#) section for detailed descriptions
- You know how to run a basic Kivy application. See [Create an application](#) if you don't.

If you have read the programming guide, and understand both basic Widget concepts ([A Simple Paint App](#)) and basic concepts of the kv language ([Kv language](#)), you can probably skip the first 2 steps and go straight to step 3.

Note: You can find the entire source code, and source code files for each step in the Kivy examples directory under [tutorials/pong/](#)

Ready? Sweet, let's get started!

17.2 Getting Started

Getting Started

Let's start by getting a really simple Kivy app up and running. Create a directory for the game and a file named `main.py`

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3
4
5 class PongGame(Widget):
6     pass
7
8
9 class PongApp(App):
10     def build(self):
11         return PongGame()
12
13
14 if __name__ == '__main__':
15     PongApp().run()
```

Go ahead and run the application. It should just show a black window at this point. What we've done is create a very simple Kivy [App](#), which creates an instance of our `PongGame` Widget class and returns it as the root element for the applications UI, which you should imagine at this point as a hierarchical tree of Widgets. Kivy places this widget-tree in the default Window. In the next step, we will draw the Pong background and scores by defining how the `PongGame` widget looks.

17.3 Add Simple Graphics

Creation of pong.kv

We will use a .kv file to define the look and feel of the `PongGame` class. Since our [App](#) class is called `PongApp`, we can simply create a file called `pong.kv` in the same directory that will be automatically loaded when the application is run. So create a new file called "`pong.kv`" and add the following contents.

```
1 #:kivy 1.0.9
2
```

(continues on next page)


```

3 <PongGame>:
4     canvas:
5         Rectangle:
6             pos: self.center_x - 5, 0
7             size: 10, self.height
8
9         Label:
10            font_size: 70
11            center_x: root.width / 4
12            top: root.top - 50
13            text: "0"
14
15        Label:
16            font_size: 70
17            center_x: root.width * 3 / 4
18            top: root.top - 50
19            text: "0"

```

Note: COMMON ERROR: The name of the kv file, e.g. pong.kv, must match the name of the app, e.g. PongApp (the part before the App ending).

If you run the app now, you should see a vertical bar in the middle, and two zeros where the player scores will be displayed.

17.3.1 Explaining the Kv File Syntax

Before going on to the next step, you might want to take a closer look at the contents of the kv file we just created and figure out what is going on. If you understand what's happening, you can probably skip ahead to the next step.

On the very first line we have:

```
#:kivy 1.0.9
```

This first line is required in every kv file. It should start with `#:kivy` followed by a space and the Kivy version it is intended for (so Kivy can make sure you have at least the required version, or handle backwards compatibility later on).

After that, we begin defining rules that are applied to all `PongGame` instances:

```
<PongGame>:
    ...
```

Like Python, kv files use indentation to define nested blocks. A block defined with a class name inside the `<` and `>` characters is a *Widget* rule. It will be applied to any instance of the named class. If you replaced `PongGame` with `Widget` in our example, all `Widget` instances would have the vertical line and the two `Label` widgets inside them because it would define these rules for all `Widget` instances.

Inside a rule section, you can add various blocks to define the style and contents of the widgets they will be applied to. You can:

- set property values,
- add child widgets
- define a *canvas* section in which you can add Graphics instructions that define how the widget is rendered.

The first block inside the `<PongGame>` rule we have is a *canvas* block:

```
<PongGame>:
    canvas:
        Rectangle:
            pos: self.center_x - 5, 0
            size: 10, self.height
```

So this canvas block says that the `PongGame` widget should draw some graphics primitives. In this case, we add a rectangle to the canvas. We set the pos of the rectangle to be 5 pixels left of the horizontal center of the widget, and 0 for y. The size of the rectangle is set to 10 pixels in width, and the widget's height in height. The nice thing about defining the graphics like this, is that the rendered rectangle will be automatically updated when the properties of any widgets used in the value expression change.

Note: Try to resize the application window and notice what happens. That's right, the entire UI resizes automatically. The standard behaviour of the Window is to resize an element based on its property *size_hint*. The default widget *size_hint* is (1,1), meaning it will be stretched 100% in both x-direction and y-direction and hence fill the available space. Since the pos and size of the rectangle and center_x and top of the score labels were defined within the context of the `PongGame` class, these properties will automatically update when the corresponding widget properties change. Using the Kv language gives you automatic property binding. :)

The last two sections we add look pretty similar. Each of them adds a `Label` widget as a child widget to the `PongGame` widget. For now, the text on both of them is just set to "0". We'll hook that up to the actual score once we have the logic implemented, but the labels already look good since we set a bigger *font_size*, and positioned them relatively to the root widget. The `root` keyword can be used inside the child block to refer back to the parent/root widget the rule applies to (`PongGame` in this case):

```
<PongGame>:
    # ...

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: "0"

    Label:
        font_size: 70
        center_x: root.width * 3 / 4
        top: root.top - 50
        text: "0"
```

17.4 Add the Ball

Add the Ball

Ok, so we have a basic pong arena to play in, but we still need the players and a ball to hit around. Let's start with the ball. We'll add a new *PongBall* class to create a widget that will be our ball and make it bounce around.

17.4.1 PongBall Class

Here is the Python code for the `PongBall` class:

```

1 class PongBall(Widget):
2
3     # velocity of the ball on x and y axis
4     velocity_x = NumericProperty(0)
5     velocity_y = NumericProperty(0)
6
7     # referencelist property so we can use ball.velocity as
8     # a shorthand, just like e.g. w.pos for w.x and w.y
9     velocity = ReferenceListProperty(velocity_x, velocity_y)
10
11     # ``move`` function will move the ball one step. This
12     # will be called in equal intervals to animate the ball
13     def move(self):
14         self.pos = Vector(*self.velocity) + self.pos

```

And here is the kv rule used to draw the ball as a white circle:

```

<PongBall>:
    size: 50, 50
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size

```

To make it all work, you also have to add the imports for the *Properties* Property classes used and the *Vector*.

Here is the entire updated python code and kv file for this step:

main.py:

```

1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.properties import NumericProperty, ReferenceListProperty
4 from kivy.vector import Vector
5
6
7 class PongBall(Widget):
8     velocity_x = NumericProperty(0)
9     velocity_y = NumericProperty(0)
10    velocity = ReferenceListProperty(velocity_x, velocity_y)
11
12    def move(self):
13        self.pos = Vector(*self.velocity) + self.pos
14
15
16 class PongGame(Widget):
17     pass
18
19
20 class PongApp(App):
21     def build(self):
22         return PongGame()
23
24
25 if __name__ == '__main__':
26     PongApp().run()

```

pong.kv:

```

1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10 <PongGame>:
11     canvas:
12         Rectangle:
13             pos: self.center_x - 5, 0
14             size: 10, self.height
15
16     Label:
17         font_size: 70
18         center_x: root.width / 4
19         top: root.top - 50
20         text: "0"
21
22     Label:
23         font_size: 70
24         center_x: root.width * 3 / 4
25         top: root.top - 50
26         text: "0"
27
28     PongBall:
29         center: self.parent.center
30

```

Note that not only a `<PongBall>` widget rule has been added, but also a child widget `PongBall` in the `<PongGame>` widget rule.

17.5 Adding Ball Animation

Making the ball move

Cool, so now we have a ball, and it even has a `move` function... but it's not moving yet. Let's fix that.

17.5.1 Scheduling Functions on the Clock

We need the `move` method of our ball to be called regularly. Luckily, Kivy makes this pretty easy by letting us schedule any function we want using the `Clock` and specifying the interval:

```
Clock.schedule_interval(game.update, 1.0/60.0)
```

This line for example, would cause the `update` function of the game object to be called once every 60th of a second (60 times per second).

17.5.2 Object Properties/References

We have another problem though. We'd like to make sure the `PongBall` has its `move` function called regularly, but in our code we don't have any references to the ball object since we just added it via the

kv file inside the kv rule for the `PongGame` class. The only reference to our game is the one we return in the applications build method.

Since we're going to have to do more than just move the ball (e.g. bounce it off the walls and later the player's racket), we'll probably need an `update` method for our `PongGame` class anyway. Furthermore, given that we have a reference to the game object already, we can easily schedule its new `update` method when the application gets built:

```
1 class PongGame(Widget):
2
3     def update(self, dt):
4         # call ball.move and other stuff
5         pass
6
7 class PongApp(App):
8
9     def build(self):
10         game = PongGame()
11         Clock.schedule_interval(game.update, 1.0/60.0)
12         return game
```

However, that still doesn't change the fact that we don't have a reference to the `PongBall` child widget created by the kv rule. To fix this, we can add an `ObjectProperty` to the `PongGame` class, and hook it up to the widget created in the kv rule. Once that's done, we can easily reference the ball property inside the update method and even make it bounce off the edges:

```
1 class PongGame(Widget):
2     ball = ObjectProperty(None)
3
4     def update(self, dt):
5         self.ball.move()
6
7         # bounce off top and bottom
8         if (self.ball.y < 0) or (self.ball.top > self.height):
9             self.ball.velocity_y *= -1
10
11        # bounce off left and right
12        if (self.ball.x < 0) or (self.ball.right > self.width):
13            self.ball.velocity_x *= -1
```

Don't forget to hook it up in the kv file, by giving the child widget an id and setting the `PongGame`'s `ball` `ObjectProperty` to that id:

```
<PongGame>:
    ball: pong_ball

    # ... (canvas and Labels)

    PongBall:
        id: pong_ball
        center: self.parent.center
```

Note: At this point everything is hooked up for the ball to bounce around. If you're coding along as we go, you might be wondering why the ball isn't moving anywhere. The ball's velocity is set to 0 on both x and y. In the code listing below, a `serve_ball` method is added to the `PongGame` class and called in the app's `build` method. It sets a random x and y velocity for the ball, and also resets the position, so we can use it later to reset the ball when a player has scored a point.

Here is the entire code for this step:

main.py:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.properties import (
4     NumericProperty, ReferenceListProperty, ObjectProperty
5 )
6 from kivy.vector import Vector
7 from kivy.clock import Clock
8 from random import randint
9
10
11 class PongBall(Widget):
12     velocity_x = NumericProperty(0)
13     velocity_y = NumericProperty(0)
14     velocity = ReferenceListProperty(velocity_x, velocity_y)
15
16     def move(self):
17         self.pos = Vector(*self.velocity) + self.pos
18
19
20 class PongGame(Widget):
21     ball = ObjectProperty(None)
22
23     def serve_ball(self):
24         self.ball.center = self.center
25         self.ball.velocity = Vector(4, 0).rotate(randint(0, 360))
26
27     def update(self, dt):
28         self.ball.move()
29
30         # bounce off top and bottom
31         if (self.ball.y < 0) or (self.ball.top > self.height):
32             self.ball.velocity_y *= -1
33
34         # bounce off left and right
35         if (self.ball.x < 0) or (self.ball.right > self.width):
36             self.ball.velocity_x *= -1
37
38
39 class PongApp(App):
40     def build(self):
41         game = PongGame()
42         game.serve_ball()
43         Clock.schedule_interval(game.update, 1.0 / 60.0)
44         return game
45
46
47 if __name__ == '__main__':
48     PongApp().run()
```

pong.kv:

```
1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
```

(continues on next page)

```

6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10    <PongGame>:
11        ball: pong_ball
12
13        canvas:
14            Rectangle:
15                pos: self.center_x - 5, 0
16                size: 10, self.height
17
18            Label:
19                font_size: 70
20                center_x: root.width / 4
21                top: root.top - 50
22                text: "0"
23
24            Label:
25                font_size: 70
26                center_x: root.width * 3 / 4
27                top: root.top - 50
28                text: "0"
29
30        PongBall:
31            id: pong_ball
32            center: self.parent.center
33

```

17.6 Connect Input Events

Adding Players and reacting to touch input

Sweet, our ball is bouncing around. The only things missing now are the movable player rackets and keeping track of the score. We won't go over all the details of creating the class and kv rules again, since those concepts were already covered in the previous steps. Instead, let's focus on how to move the Player widgets in response to user input. You can get the whole code and kv rules for the `PongPaddle` class at the end of this section.

In Kivy, a widget can react to input by implementing the `on_touch_down`, the `on_touch_move` and the `on_touch_up` methods. By default, the `Widget` class implements these methods by just calling the corresponding method on all its child widgets to pass on the event until one of the children returns `True`.

Pong is pretty simple. The rackets just need to move up and down. In fact it's so simple, we don't even really need to have the player widgets handle the events themselves. We'll just implement the `on_touch_move` function for the `PongGame` class and have it set the position of the left or right player based on whether the touch occurred on the left or right side of the screen.

Check the `on_touch_move` handler:

```

1 def on_touch_move(self, touch):
2     if touch.x < self.width/3:
3         self.player1.center_y = touch.y
4     if touch.x > self.width - self.width/3:
5         self.player2.center_y = touch.y

```

We'll keep the score for each player in a *NumericProperty*. The score labels of the *PongGame* are kept updated by changing the *NumericProperty* *score*, which in turn updates the *PongGame* child labels text property. This binding occurs because Kivy *properties* automatically bind to any references in their corresponding kv files. When the ball escapes out of the sides, we'll update the score and serve the ball again by changing the *update* method in the *PongGame* class. The *PongPaddle* class also implements a *bounce_ball* method, so that the ball bounces differently based on where it hits the racket. Here is the code for the *PongPaddle* class:

```
1 class PongPaddle(Widget):
2
3     score = NumericProperty(0)
4
5     def bounce_ball(self, ball):
6         if self.collide_widget(ball):
7             speedup = 1.1
8             offset = 0.02 * Vector(0, ball.center_y-self.center_y)
9             ball.velocity = speedup * (offset - ball.velocity)
```

Note: This algorithm for ball bouncing is very simple, but will have strange behavior if the ball hits the paddle from the side or bottom... this is something you could try to fix yourself if you like.

And here it is in context. Pretty much done:

main.py:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.properties import (
4     NumericProperty, ReferenceListProperty, ObjectProperty
5 )
6 from kivy.vector import Vector
7 from kivy.clock import Clock
8
9
10 class PongPaddle(Widget):
11     score = NumericProperty(0)
12
13     def bounce_ball(self, ball):
14         if self.collide_widget(ball):
15             vx, vy = ball.velocity
16             offset = (ball.center_y - self.center_y) / (self.height / 2)
17             bounced = Vector(-1 * vx, vy)
18             vel = bounced * 1.1
19             ball.velocity = vel.x, vel.y + offset
20
21
22 class PongBall(Widget):
23     velocity_x = NumericProperty(0)
24     velocity_y = NumericProperty(0)
25     velocity = ReferenceListProperty(velocity_x, velocity_y)
26
27     def move(self):
28         self.pos = Vector(*self.velocity) + self.pos
29
30
31 class PongGame(Widget):
32     ball = ObjectProperty(None)
33     player1 = ObjectProperty(None)
```

(continues on next page)


```

34 player2 = ObjectProperty(None)
35
36 def serve_ball(self, vel=(4, 0)):
37     self.ball.center = self.center
38     self.ball.velocity = vel
39
40 def update(self, dt):
41     self.ball.move()
42
43     # bounce of paddles
44     self.player1.bounce_ball(self.ball)
45     self.player2.bounce_ball(self.ball)
46
47     # bounce ball off bottom or top
48     if (self.ball.y < self.y) or (self.ball.top > self.top):
49         self.ball.velocity_y *= -1
50
51     # went of to a side to score point?
52     if self.ball.x < self.x:
53         self.player2.score += 1
54         self.serve_ball(vel=(4, 0))
55     if self.ball.x > self.width:
56         self.player1.score += 1
57         self.serve_ball(vel=(-4, 0))
58
59 def on_touch_move(self, touch):
60     if touch.x < self.width / 3:
61         self.player1.center_y = touch.y
62     if touch.x > self.width - self.width / 3:
63         self.player2.center_y = touch.y
64
65
66 class PongApp(App):
67     def build(self):
68         game = PongGame()
69         game.serve_ball()
70         Clock.schedule_interval(game.update, 1.0 / 60.0)
71         return game
72
73
74 if __name__ == '__main__':
75     PongApp().run()

```

pong.kv:

```

1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10 <PongPaddle>:
11     size: 25, 200
12     canvas:
13         Rectangle:

```

(continues on next page)

```

14         pos: self.pos
15         size: self.size
16
17 <PongGame>:
18     ball: pong_ball
19     player1: player_left
20     player2: player_right
21
22     canvas:
23         Rectangle:
24             pos: self.center_x - 5, 0
25             size: 10, self.height
26
27     Label:
28         font_size: 70
29         center_x: root.width / 4
30         top: root.top - 50
31         text: str(root.player1.score)
32
33     Label:
34         font_size: 70
35         center_x: root.width * 3 / 4
36         top: root.top - 50
37         text: str(root.player2.score)
38
39     PongBall:
40         id: pong_ball
41         center: self.parent.center
42
43     PongPaddle:
44         id: player_left
45         x: root.x
46         center_y: root.center_y
47
48     PongPaddle:
49         id: player_right
50         x: root.width - self.width
51         center_y: root.center_y
52

```

17.7 Where To Go Now?

Have some fun

Well, the pong game is pretty much complete. If you understood all of the things that are covered in this tutorial, give yourself a pat on the back and think about how you could improve the game. Here are a few ideas of things you could do:

- Add some nicer graphics / images. (Hint: check out the *source* property on the graphics instructions like *circle* or *Rectangle*, to set an image as the texture.)
- Make the game end after a certain score. Maybe once a player has 10 points, you can display a large “PLAYER 1 WINS” label and/or add a main menu to start, pause and reset the game. (Hint: check out the *Button* and *Label* classes, and figure out how to use their *add_widget* and *remove_widget* functions to add or remove widgets dynamically.)
- Make it a 4 player Pong Game. Most tablets have Multi-Touch support, so wouldn’t it be cool to

have a player on each side and have four people play at the same time?

- Fix the simplistic collision check so hitting the ball with an end of the paddle results in a more realistic bounce.

Note: You can find the entire source code and source code files for each step in the Kivy examples directory under `tutorials/pong/`

A Simple Paint App

In the following tutorial, you will be guided through the creation of your first widget. This provides powerful and important knowledge when programming Kivy applications, as it lets you create completely new user interfaces with custom elements for your specific purpose.

18.1 Basic Considerations

When creating an application, you have to ask yourself three important questions:

- What data does my application process?
- How do I visually represent that data?
- How does the user interact with that data?

If you want to write a very simple line drawing application for example, you most likely want the user to just draw on the screen with his/her fingers. That's how the user *interacts* with your application. While doing so, your application would memorize the positions where the user's finger were, so that you can later draw lines between those positions. So the points where the fingers were would be your *data* and the lines that you draw between them would be your *visual representation*.

In Kivy, an application's user interface is composed of Widgets. Everything that you see on the screen is somehow drawn by a widget. Often you would like to be able to reuse code that you already wrote in a different context, which is why widgets typically represent one specific instance that answers the three questions above. A widget encapsulates data, defines the user's interaction with that data and draws its visual representation. You can build anything from simple to complex user interfaces by nesting widgets. There are many widgets built in, such as buttons, sliders and other common stuff. In many cases, however, you need a custom widget that is beyond the scope of what is shipped with Kivy (e.g. a medical visualization widget).

So keep these three questions in mind when you design your widgets. Try to write them in a minimal and reusable manner (i.e. a widget does exactly what its supposed to do and nothing more. If you need more, write more widgets or compose other widgets of smaller widgets. We try to adhere to the [Single Responsibility Principle](#)).

18.2 Paint Widget

We're sure one of your childhood dreams has always been creating your own multitouch paint program. Allow us to help you achieve that. In the following sections you will successively learn how to write a program like that using Kivy. Make sure that you have read and understood *Create an application*. You have? Great! Let's get started!

18.2.1 Initial Structure

Let's start by writing the very basic code structure that we need. By the way, all the different pieces of code that are used in this section are also available in the `examples/guide/firstwidget` directory that comes with Kivy, so you don't need to copy & paste it all the time. Here is the basic code skeleton that we will need:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3
4
5 class MyPaintWidget(Widget):
6     pass
7
8
9 class MyPaintApp(App):
10     def build(self):
11         return MyPaintWidget()
12
13
14 if __name__ == '__main__':
15     MyPaintApp().run()
```

This is actually really simple. Save it as `paint.py`. If you run it, you should only see a black screen. As you can see, instead of using a built-in widget such as a `Button` (see *Create an application*), we are going to write our own widget to do the drawing. We do that by creating a class that inherits from `Widget` (line 5-6) and although that class does nothing yet, we can still treat it like a normal Kivy widget (line 11). The `if __name__ ...` construct (line 14) is a Python mechanism that prevents you from executing the code in the if-statement when importing from the file, i.e. if you write `import paint`, it won't do something unexpected but just nicely provide the classes defined in the file.

Note: You may be wondering why you have to import `App` and `Widget` separately, instead of doing something like `from kivy import *`. While shorter, this would have the disadvantage of *polluting your namespace* and make the start of the application potentially much slower. It can also introduce ambiguity into class and variable naming, so is generally frowned upon in the Python community. The way we do it is faster and cleaner.

18.2.2 Adding Behaviour

Let's now add some actual behaviour to the widget, i.e. make it react to user input. Change the code like so:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3
4
```

(continues on next page)

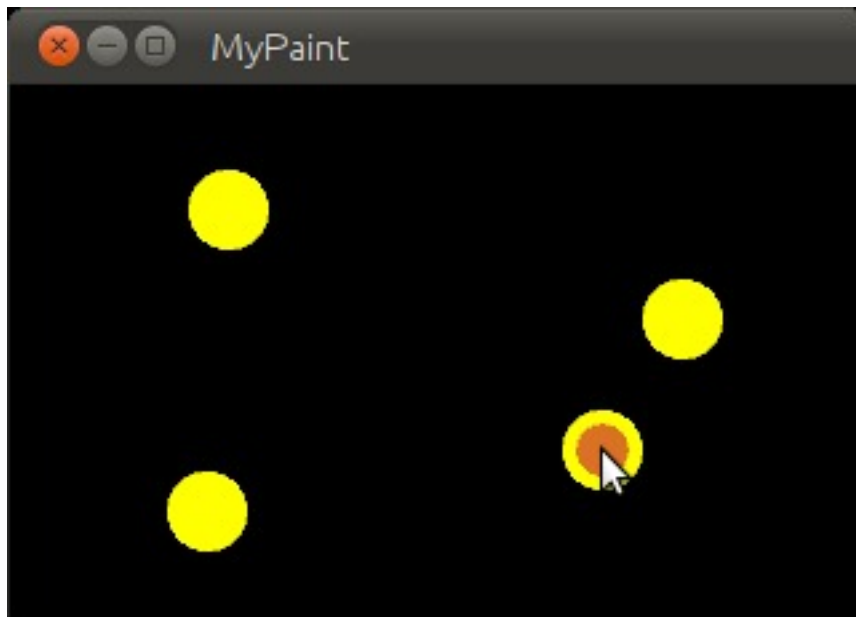
(continued from previous page)

```
5 class MyPaintWidget(Widget):
6     def on_touch_down(self, touch):
7         print(touch)
8
9
10 class MyPaintApp(App):
11     def build(self):
12         return MyPaintWidget()
13
14
15 if __name__ == '__main__':
16     MyPaintApp().run()
```

This is just to show how easy it is to react to user input. When a *MotionEvent* (i.e. a touch, click, etc.) occurs, we simply print the information about the touch object to the console. You won't see anything on the screen, but if you observe the command-line from which you are running the program, you will see a message for every touch. This also demonstrates that a widget does not have to have a visual representation.

Now that's not really an overwhelming user experience. Let's add some code that actually draws something into our window:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.graphics import Color, Ellipse
4
5
6 class MyPaintWidget(Widget):
7
8     def on_touch_down(self, touch):
9         with self.canvas:
10             Color(1, 1, 0)
11             d = 30.
12             Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d, d))
13
14
15 class MyPaintApp(App):
16
17     def build(self):
18         return MyPaintWidget()
19
20
21 if __name__ == '__main__':
22     MyPaintApp().run()
```



If you run your code with these modifications, you will see that every time you touch, there will be a small yellow circle drawn where you touched. How does it work?

- Line 9: We use Python's `with` statement with the widget's `Canvas` object. This is like an area in which the widget can draw things to represent itself on the screen. By using the `with` statement with it, all successive drawing commands that are properly indented will modify this canvas. The `with` statement also makes sure that after our drawing, internal state can be cleaned up properly.
- Line 10: You might have guessed it already: This sets the `Color` for successive drawing operations to yellow (default color format is RGB, so (1, 1, 0) is yellow). This is true until another `Color` is set. Think of this as dipping your brushes in that color, which you can then use to draw on a canvas until you dip the brushes into another color.
- Line 11: We specify the diameter for the circle that we are about to draw. Using a variable for that is preferable since we need to refer to that value multiple times and we don't want to have to change it in several places if we want the circle bigger or smaller.
- Line 12: To draw a circle, we simply draw an `Ellipse` with equal width and height. Since we want the circle to be drawn where the user touches, we pass the touch's position to the ellipse. Note that we need to shift the ellipse by $-d/2$ in the x and y directions (i.e. left and downwards) because the position specifies the bottom left corner of the ellipse's bounding box, and we want it to be centered around our touch.

That was easy, wasn't it? It gets better! Update the code to look like this:

```

1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.graphics import Color, Ellipse, Line
4
5
6 class MyPaintWidget(Widget):
7
8     def on_touch_down(self, touch):
9         with self.canvas:
10             Color(1, 1, 0)
11             d = 30.
12             Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d, d))
13             touch.ud['line'] = Line(points=(touch.x, touch.y))
14
15     def on_touch_move(self, touch):

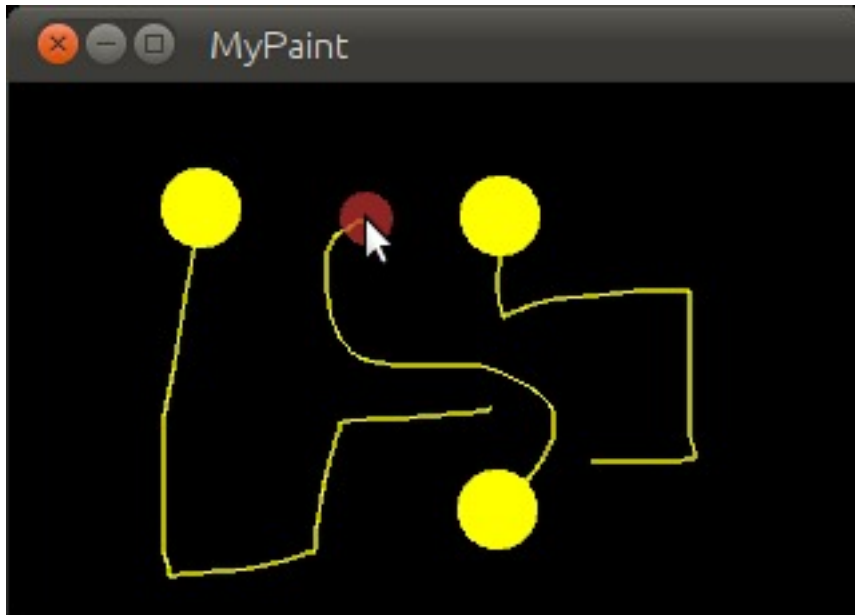
```

(continues on next page)


```

16         touch.ud['line'].points += [touch.x, touch.y]
17
18
19 class MyPaintApp(App):
20
21     def build(self):
22         return MyPaintWidget()
23
24
25 if __name__ == '__main__':
26     MyPaintApp().run()

```



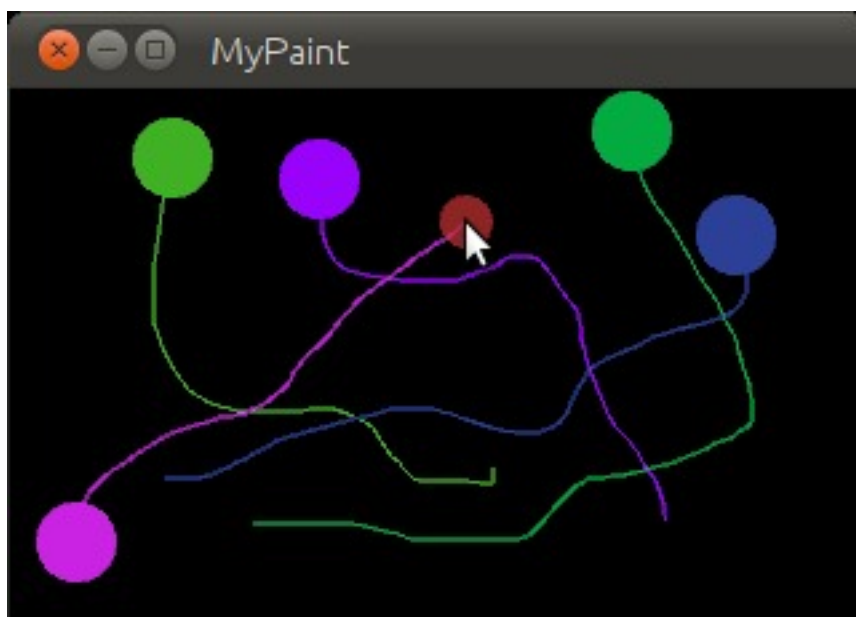
This is what has changed:

- Line 3: We now not only import the *Ellipse* drawing instruction, but also the *Line* drawing instruction. If you look at the documentation for *Line*, you will see that it accepts a *points* argument that has to be a list of 2D point coordinates, like (x1, y1, x2, y2, ..., xN, yN).
- Line 13: This is where it gets interesting. `touch.ud` is a Python dictionary (type `<dict>`) that allows us to store *custom attributes* for a touch.
- Line 13: We make use of the *Line* instruction that we imported and set a *Line* up for drawing. Since this is done in `on_touch_down`, there will be a new line for every new touch. By creating the line inside the `with` block, the canvas automatically knows about the line and will draw it. We just want to modify the line later, so we store a reference to it in the `touch.ud` dictionary under the arbitrarily chosen but aptly named key 'line'. We pass the line that we're creating the initial touch position because that's where our line will begin.
- Lines 15: We add a new method to our widget. This is similar to the `on_touch_down` method, but instead of being called when a *new* touch occurs, this method is being called when an *existing* touch (for which `on_touch_down` was already called) moves, i.e. its position changes. Note that this is the *same* *MotionEvent* object with updated attributes. This is something we found incredibly handy and you will shortly see why.
- Line 16: Remember: This is the same touch object that we got in `on_touch_down`, so we can simply access the data we stored away in the `touch.ud` dictionary! To the line we set up for this touch earlier, we now add the current position of the touch as a new point. We know that we need to extend the line because this happens in `on_touch_move`, which is only called

when the touch has moved, which is exactly why we want to update the line. Storing the line in the `touch.ud` makes it a whole lot easier for us as we don't have to maintain our own touch-to-line bookkeeping.

So far so good. This isn't exactly beautiful yet, though. It looks a bit like spaghetti bolognese. How about giving each touch its own color? Great, let's do it:

```
1 from random import random
2 from kivy.app import App
3 from kivy.uix.widget import Widget
4 from kivy.graphics import Color, Ellipse, Line
5
6
7 class MyPaintWidget(Widget):
8
9     def on_touch_down(self, touch):
10         color = (random(), random(), random())
11         with self.canvas:
12             Color(*color)
13             d = 30.
14             Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d, d))
15             touch.ud['line'] = Line(points=(touch.x, touch.y))
16
17     def on_touch_move(self, touch):
18         touch.ud['line'].points += [touch.x, touch.y]
19
20
21 class MyPaintApp(App):
22
23     def build(self):
24         return MyPaintWidget()
25
26
27 if __name__ == '__main__':
28     MyPaintApp().run()
```



Here are the changes:

- Line 1: We import Python's `random()` function that will give us random values in the range of `[0., 1.)`.

- Line 10: In this case we simply create a new tuple of 3 random float values that will represent a random RGB color. Since we do this in `on_touch_down`, every new touch will get its own color. Don't get confused by the use of **tuples**. We're just binding the tuple to `color` for use as a shortcut within this method because we're lazy.
- Line 12: As before, we set the color for the canvas. Only this time we use the random values we generated and feed them to the color class using Python's tuple unpacking syntax (since the `Color` class expects three individual color components instead of just 1. If we were to pass the tuple directly, that would be just 1 value being passed, regardless of the fact that the tuple itself contains 3 values).

This looks a lot nicer already! With a lot of skill and patience, you might even be able to create a nice little drawing!

Note: Since by default the **Color** instructions assume RGB mode and we're feeding a tuple with three random float values to it, it might very well happen that we end up with a lot of dark or even black colors if we are unlucky. That would be bad because by default the background color is dark as well, so you wouldn't be able to (easily) see the lines you draw. There is a nice trick to prevent this: Instead of creating a tuple with three random values, create a tuple like this: `(random(), 1., 1.)`. Then, when passing it to the color instruction, set the mode to HSV color space: `Color(*color, mode='hsv')`. This way you will have a smaller number of possible colors, but the colors that you get will always be equally bright: only the hue changes.

18.2.3 Bonus Points

At this point, we could say we are done. The widget does what it's supposed to do: it traces the touches and draws lines. It even draws circles at the positions where a line begins.

But what if the user wants to start a new drawing? With the current code, the only way to clear the window would be to restart the entire application. Luckily, we can do better. Let us add a *Clear* button that erases all the lines and circles that have been drawn so far. There are two options now:

- We could either create the button as a child of our widget. That would imply that if you create more than one widget, every widget gets its own button. If you're not careful, this will also allow users to draw on top of the button, which might not be what you want.
- Or we set up the button only once, initially, in our app class and when it's pressed we clear the widget.

For our simple example, it doesn't really matter that much. For larger applications you should give some thought to who does what in your app. We'll go with the second option here so that you see how you can build up your application's widget tree in your app class's **build()** method. We'll also change to the HSV color space (see preceding note):

```

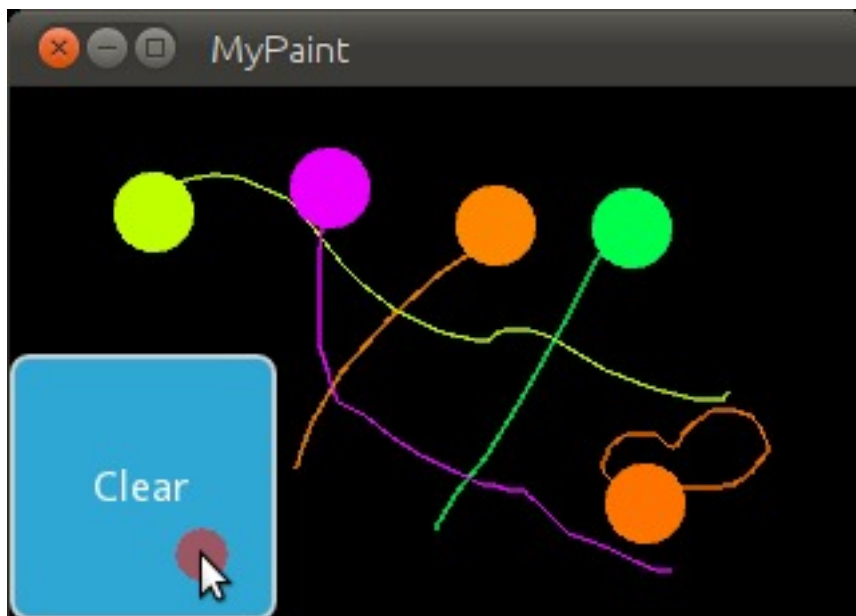
1 from random import random
2 from kivy.app import App
3 from kivy.uix.widget import Widget
4 from kivy.uix.button import Button
5 from kivy.graphics import Color, Ellipse, Line
6
7
8 class MyPaintWidget(Widget):
9
10     def on_touch_down(self, touch):
11         color = (random(), 1, 1)
12         with self.canvas:
13             Color(*color, mode='hsv')
```

(continues on next page)

```

14         d = 30.
15         Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d, d))
16         touch.ud['line'] = Line(points=(touch.x, touch.y))
17
18     def on_touch_move(self, touch):
19         touch.ud['line'].points += [touch.x, touch.y]
20
21
22 class MyPaintApp(App):
23
24     def build(self):
25         parent = Widget()
26         self.painter = MyPaintWidget()
27         clearbtn = Button(text='Clear')
28         clearbtn.bind(on_release=self.clear_canvas)
29         parent.add_widget(self.painter)
30         parent.add_widget(clearbtn)
31         return parent
32
33     def clear_canvas(self, obj):
34         self.painter.canvas.clear()
35
36
37 if __name__ == '__main__':
38     MyPaintApp().run()

```



Here's what happens:

- Line 4: We added an import statement to be able to use the `Button` class.
- Line 25: We create a dummy `Widget()` object as a parent for both our painting widget and the button we're about to add. This is just a poor-man's approach to setting up a widget tree hierarchy. We could just as well use a layout or do some other fancy stuff. Again: this widget does absolutely nothing except holding the two widgets we will now add to it as children.
- Line 26: We create our `MyPaintWidget()` as usual, only this time we don't return it directly but bind it to a variable name.
- Line 27: We create a button widget. It will have a label on it that displays the text 'Clear'.

- Line 28: We then bind the button's `on_release` event (which is fired when the button is pressed and then released) to the **callback function** `clear_canvas` defined on below on Lines 33 & 34.
- Line 29 & 30: We set up the widget hierarchy by making both the painter and the `clearbtn` children of the dummy parent widget. That means *painter* and *clearbtn* are now siblings in the usual computer science tree terminology.
- Line 33 & 34: Up to now, the button did nothing. It was there, visible, and you could press it, but nothing would happen. We change that here: we create a small, throw-away function that is going to be our **callback function** when the button is pressed. The function just clears the painter's canvas' contents, making it black again.

Note: The Kivy Widget class, by design, is kept simple. There are no general properties such as background color and border color. Instead, the examples and documentation illustrate how to easily handle such simple things yourself, as we have done here, setting the color for the canvas, and drawing the shape. From a simple start, you can move to more elaborate customization. Higher-level built-in widgets, deriving from Widget, such as Button, do have convenience properties such as `background_color`, but these vary by widget. Use the API docs to see what is offered by a widget, and subclass if you need to add more functionality.

Congratulations! You've written your first Kivy widget. Obviously this was just a quick introduction. There is much more to discover. We suggest taking a short break to let what you just learned sink in. Maybe draw some nice pictures to relax? If you feel like you've understood everything and are ready for more, we encourage you to read on.

Crash Course

The Kivy Crash Course is a series of YouTube video tutorials by Kivy core developer [inclement](#). They provide a simple walkthrough in Kivy for users who know how to code in Python and is friendly to Python beginners. After the Pong and Paint tutorials, this set of videos covers basic features and techniques that can be used to create your app quicker, keeping your code elegant and eye-friendly.

19.1 Basic Info

The Crash Course primarily consists of a series of YouTube videos, each roughly 10 minutes long. There are also articles describing some of the videos and the code used in the videos.

19.1.1 Topics covered by the Crash Course include:

- Use of the basic Kivy widgets such as the Label, Button, Scatter and TextInput
- Building an app for android with python-for-android's [old toolchain](#)
- Binding functions to events
- Using changes in variables on the go
- Smart user interface ([Kv language](#))
- Properties
- Canvas and drawing
- Label with scrolling
- Positioning and layouts
- Animation and Clock
- Accessing android API ([pyjnius](#), [plyer](#))
- Settings panel (and building your own options)

- [ScreenManager](#)

19.1.2 Links:

- [Videos](#)
- [Articles](#)
- [Code](#)

Part IV

API REFERENCE

The API reference is a lexicographic list of all the different classes, methods and features that Kivy offers.

Kivy framework

Kivy is an open source library for developing multi-touch applications. It is cross-platform (Linux/OSX/Windows/Android/iOS) and released under the terms of the [MIT License](#).

It comes with native support for many multi-touch input devices, a growing library of multi-touch aware widgets and hardware accelerated OpenGL drawing. Kivy is designed to let you focus on building custom and highly interactive applications as quickly and easily as possible.

With Kivy, you can take full advantage of the dynamic nature of Python. There are thousands of high-quality, free libraries that can be integrated in your application. At the same time, performance-critical parts are implemented using [Cython](#).

See <http://kivy.org> for more information.

kivy.require(*version*)

Require can be used to check the minimum version required to run a Kivy application. For example, you can start your application code like this:

```
import kivy
kivy.require('1.0.1')
```

If a user attempts to run your application with a version of Kivy that is older than the specified version, an Exception is raised.

The Kivy version string is built like this:

```
X.Y.Z[tag[tagrevision]]

X is the major version
Y is the minor version
Z is the bugfixes revision
```

The tag is optional, but may be one of `'.dev'`, `'.post'`, `'a'`, `'b'`, or `'rc'`. The tagrevision is the revision number of the tag.

Warning: You must not ask for a version with a tag, except -dev. Asking for a 'dev' version will just warn the user if the current Kivy version is not a -dev, but it will never raise an exception. You must not ask for a version with a tagrevision.

`kivy.parse_kivy_version(version)`

Parses the kivy version as described in `require()` into a 3-tuple of `([x, y, z], 'rc|a|b|dev|post', 'N')` where N is the tag revision. The last two elements may be None.

`kivy.kivy_configure()`

Call post-configuration of Kivy. This function must be called if you create the window yourself.

`kivy.kivy_register_post_configuration(callback)`

Register a function to be called when `kivy_configure()` is called.

Warning: Internal use only.

`kivy.kivy_options = {'audio': ('gstplayer', 'pygame', 'ffpyplayer', 'sdl2', 'avplayer')}`

Global settings options for kivy

`kivy.kivy_base_dir = '/home/docs/checkouts/readthedocs.org/user_builds/matham-kivy/envs/`

Kivy directory

`kivy.kivy_data_dir = '/home/docs/checkouts/readthedocs.org/user_builds/matham-kivy/envs/`

Kivy data directory

`kivy.kivy_shader_dir = '/home/docs/checkouts/readthedocs.org/user_builds/matham-kivy/envs/`

Kivy glsl shader directory

`kivy.kivy_icons_dir = '/home/docs/checkouts/readthedocs.org/user_builds/matham-kivy/envs/`

Kivy icons config path (don't remove the last '')

`kivy.kivy_home_dir = ''`

Kivy user-home storage directory

`kivy.kivy_config_fn = ''`

Kivy configuration filename

`kivy.kivy_usermodules_dir = ''`

Kivy user modules directory

20.1 Animation

`Animation` and `AnimationTransition` are used to animate `Widget` properties. You must specify at least a property name and target value. To use an Animation, follow these steps:

- Setup an Animation object
- Use the Animation object on a Widget

20.1.1 Simple animation

To animate a Widget's x or y position, simply specify the target x/y values where you want the widget positioned at the end of the animation:

```
anim = Animation(x=100, y=100)
anim.start(widget)
```

The animation will last for 1 second unless `duration` is specified. When `anim.start()` is called, the `Widget` will move smoothly from the current `x/y` position to `(100, 100)`.

20.1.2 Multiple properties and transitions

You can animate multiple properties and use built-in or custom transition functions using `transition` (or the `t=` shortcut). For example, to animate the position and size using the `'in_quad'` transition:

```
anim = Animation(x=50, size=(80, 80), t='in_quad')
anim.start(widget)
```

Note that the `t=` parameter can be the string name of a method in the `AnimationTransition` class or your own animation function.

20.1.3 Sequential animation

To join animations sequentially, use the `'+'` operator. The following example will animate to `x=50` over 1 second, then animate the size to `(80, 80)` over the next two seconds:

```
anim = Animation(x=50) + Animation(size=(80, 80), duration=2.)
anim.start(widget)
```

20.1.4 Parallel animation

To join animations in parallel, use the `'&'` operator. The following example will animate the position to `(80, 10)` over 1 second, whilst in parallel animating the size to `(800, 800)`:

```
anim = Animation(pos=(80, 10))
anim &= Animation(size=(800, 800), duration=2.)
anim.start(widget)
```

Keep in mind that creating overlapping animations on the same property may have unexpected results. If you want to apply multiple animations to the same property, you should either schedule them sequentially (via the `'+'` operator or using the `on_complete` callback) or cancel previous animations using the `cancel_all` method.

20.1.5 Repeating animation

New in version 1.8.0.

Note: This is currently only implemented for 'Sequence' animations.

To set an animation to repeat, simply set the `Sequence.repeat` property to `True`:

```
anim = Animation(...) + Animation(...)
anim.repeat = True
anim.start(widget)
```

For flow control of animations such as stopping and cancelling, use the methods already in place in the animation module.

class kivy.animation.**Animation**(**kw)

Bases: [kivy.event.EventDispatcher](#)

Create an animation definition that can be used to animate a Widget.

Parameters

duration or d: float, defaults to 1. Duration of the animation, in seconds.

transition or t: str or func Transition function for animate properties. It can be the name of a method from [AnimationTransition](#).

step or s: float Step in milliseconds of the animation. Defaults to 0, which means the animation is updated for every frame.

To update the animation less often, set the step value to a float. For example, if you want to animate at 30 FPS, use $s=1/30$.

Events

on_start: animation, widget Fired when the animation is started on a widget.

on_complete: animation, widget Fired when the animation is completed or stopped on a widget.

on_progress: animation, widget, progression Fired when the progression of the animation is changing.

Changed in version 1.4.0: Added s/step parameter.

Changed in version 1.10.0: The default value of the step parameter was changed from 1/60. to 0.

animated_properties

Return the properties used to animate.

cancel(widget)

Cancel the animation previously applied to a widget. Same effect as [stop](#), except the *on_complete* event will *not* be triggered!

New in version 1.4.0.

static cancel_all(widget, *largs)

Cancel all animations that concern a specific widget / list of properties. See [cancel](#).

Example:

```
anim = Animation(x=50)
anim.start(widget)

# and later
Animation.cancel_all(widget, 'x')
```

New in version 1.4.0.

cancel_property(widget, prop)

Even if an animation is running, remove a property. It will not be animated further. If it was the only / last property being animated, the animation will be canceled (see [cancel](#))

New in version 1.4.0.

duration

Return the duration of the animation.

have_properties_to_animate(widget)

Return True if a widget still has properties to animate.

New in version 1.8.0.

start(*widget*)

Start the animation on a widget.

stop(*widget*)

Stop the animation previously applied to a widget, triggering the *on_complete* event.

static stop_all(*widget*, **largs*)

Stop all animations that concern a specific widget / list of properties.

Example:

```
anim = Animation(x=50)
anim.start(widget)

# and later
Animation.stop_all(widget, 'x')
```

stop_property(*widget*, *prop*)

Even if an animation is running, remove a property. It will not be animated further. If it was the only/last property being animated, the animation will be stopped (see *stop*).

transition

Return the transition of the animation.

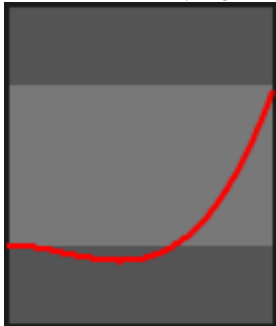
class kivy.animation.AnimationTransition

Bases: `builtins.object`

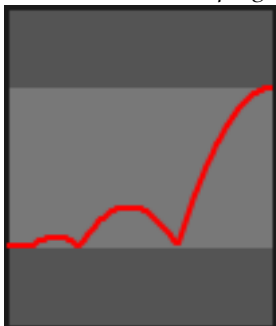
Collection of animation functions to be used with the Animation object. Easing Functions ported to Kivy from the Clutter Project <https://developer.gnome.org/clutter/stable/ClutterAlpha.html>

The *progress* parameter in each animation function is in the range 0-1.

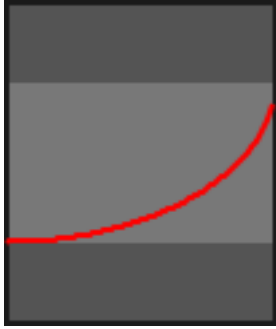
static in_back(*progress*)



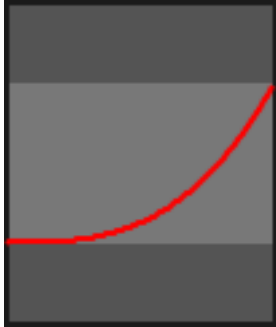
static in_bounce(*progress*)



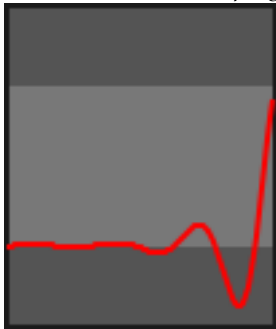
static in_circ(*progress*)



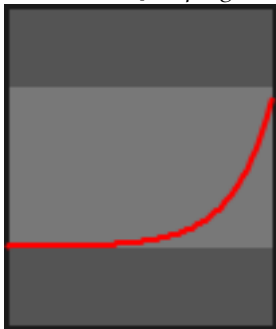
static in_cubic(*progress*)



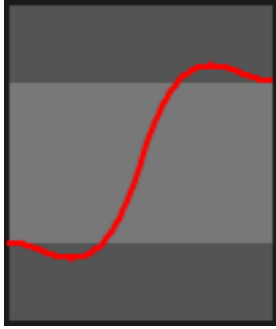
static in_elastic(*progress*)



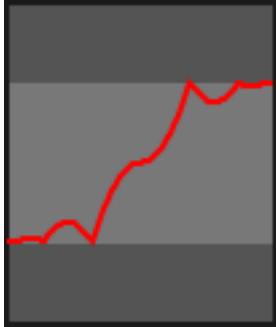
static in_expo(*progress*)



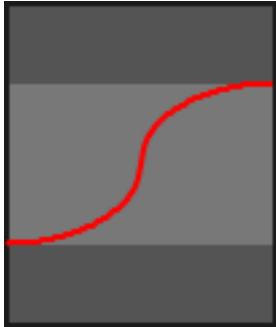
static in_out_back(*progress*)



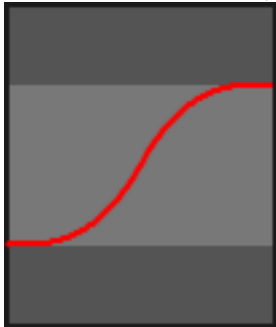
static in_out_bounce(*progress*)



static in_out_circ(*progress*)



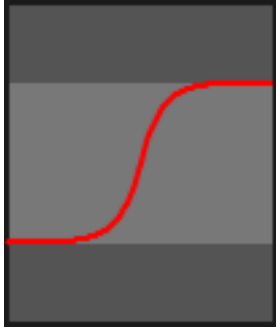
static in_out_cubic(*progress*)



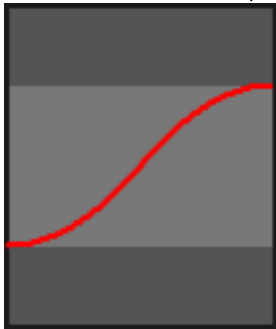
static in_out_elastic(*progress*)



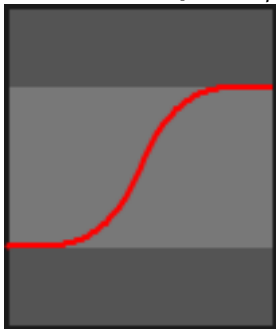
static in_out_expo(*progress*)



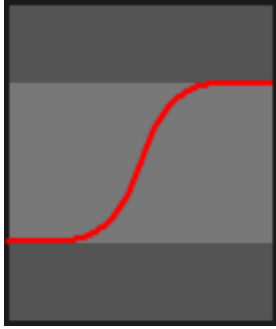
static in_out_quad(*progress*)



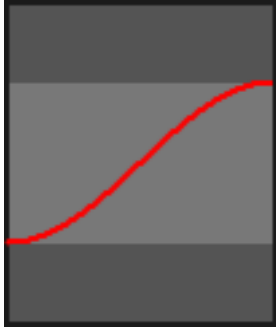
static in_out_quart(*progress*)



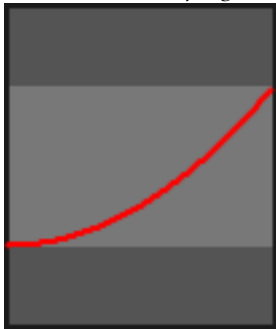
static in_out_quint(*progress*)



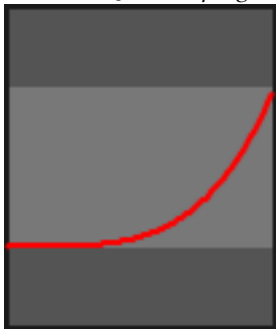
static in_out_sine(*progress*)



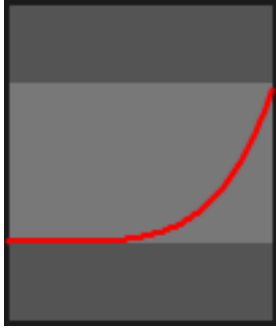
static in_quad(*progress*)



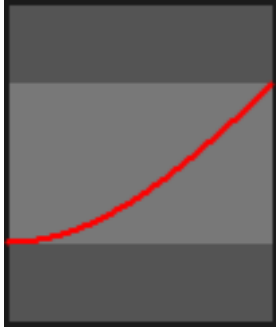
static in_quart(*progress*)



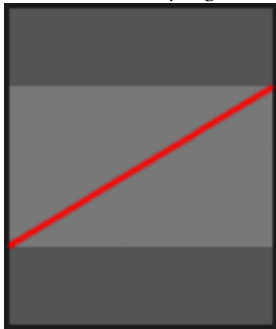
static in_quint(*progress*)



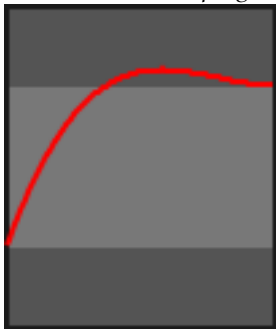
static in_sine(*progress*)



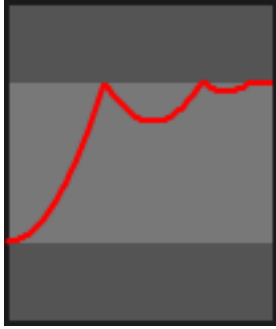
static linear(*progress*)



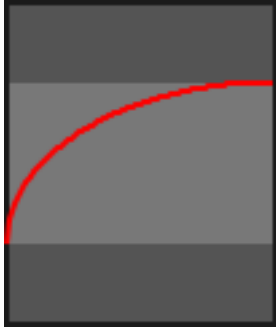
static out_back(*progress*)



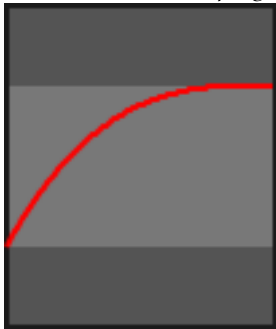
static out_bounce(*progress*)



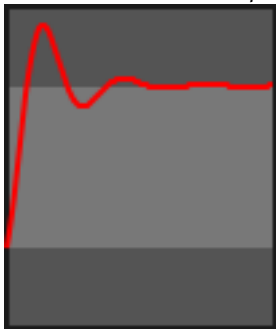
static out_circ(*progress*)



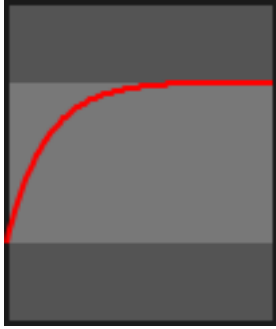
static out_cubic(*progress*)



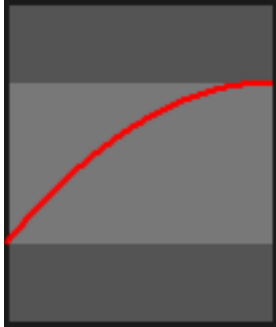
static out_elastic(*progress*)



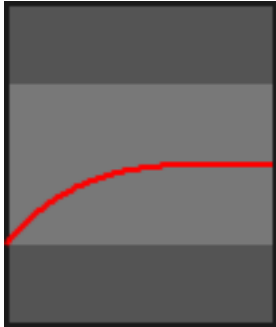
static out_expo(*progress*)



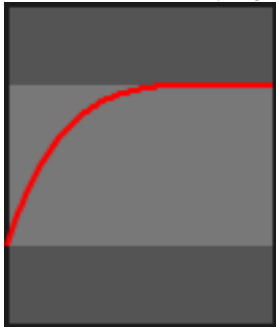
static out_quad(*progress*)



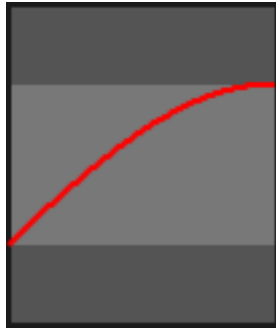
static out_quart(*progress*)



static out_quint(*progress*)



static out_sine(*progress*)



20.2 Application

The *App* class is the base for creating Kivy applications. Think of it as your main entry point into the Kivy run loop. In most cases, you subclass this class and make your own app. You create an instance of your specific app class and then, when you are ready to start the application's life cycle, you call your instance's *App.run()* method.

20.2.1 Creating an Application

Method using *build()* override

To initialize your app with a widget tree, override the *build()* method in your app class and return the widget tree you constructed.

Here's an example of a very simple application that just shows a button:

```
'''
Application example using build() + return
=====

An application can be built if you return a widget on build(), or if you set
self.root.
'''

import kivy
kivy.require('1.0.7')

from kivy.app import App
from kivy.uix.button import Button

class TestApp(App):

    def build(self):
        # return a Button() as a root widget
        return Button(text='hello world')

if __name__ == '__main__':
    TestApp().run()
```

The file is also available in the examples folder at `kivy/examples/application/app_with_build.py`.

Here, no widget tree was constructed (or if you will, a tree with only the root node).

Method using kv file

You can also use the *Kivy Language* for creating applications. The .kv can contain rules and root widget definitions at the same time. Here is the same example as the Button one in a kv file.

Contents of 'test.kv':

```
#:kivy 1.0

Button:
    text: 'Hello from test.kv'
```

Contents of 'main.py':

```
'''
Application built from a .kv file
=====

This shows how to implicitly use a .kv file for your application. You
should see a full screen button labelled "Hello from test.kv".

After Kivy instantiates a subclass of App, it implicitly searches for a .kv
file. The file test.kv is selected because the name of the subclass of App is
TestApp, which implies that kivy should try to load "test.kv". That file
contains a root Widget.
'''

import kivy
kivy.require('1.0.7')

from kivy.app import App

class TestApp(App):
    pass

if __name__ == '__main__':
    TestApp().run()
```

See kivy/examples/application/app_with_kv.py.

The relationship between main.py and test.kv is explained in *App.load_kv()*.

20.2.2 Application configuration

Use the configuration file

Your application might need its own configuration file. The *App* class handles 'ini' files automatically if you add the section key-value pair to the *App.build_config()* method using the *config* parameter (an instance of *ConfigParser*):

```
class TestApp(App):
    def build_config(self, config):
        config.setdefault('section1', {
            'key1': 'value1',
            'key2': '42'
        })
```


As soon as you add one section to the config, a file is created on the disk (see [get_application_config](#) for its location) and named based your class name. “TestApp” will give a config file named “test.ini” with the content:

```
[section1]
key1 = value1
key2 = 42
```

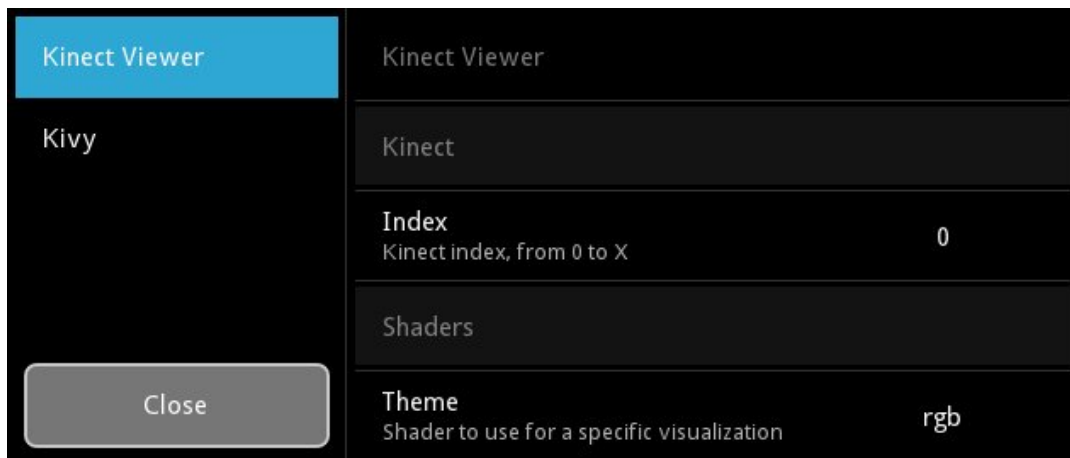
The “test.ini” will be automatically loaded at runtime and you can access the configuration in your [App.build\(\)](#) method:

```
class TestApp(App):
    def build_config(self, config):
        config.setdefault('section1', {
            'key1': 'value1',
            'key2': '42'
        })

    def build(self):
        config = self.config
        return Label(text='key1 is %s and key2 is %d' % (
            config.get('section1', 'key1'),
            config.getint('section1', 'key2')))
```

Create a settings panel

Your application can have a settings panel to let your user configure some of your config tokens. Here is an example done in the KinectViewer example (available in the examples directory):



You can add your own panels of settings by extending the [App.build_settings\(\)](#) method. Check the [Settings](#) about how to create a panel, because you need a JSON file / data first.

Let's take as an example the previous snippet of TestApp with custom config. We could create a JSON like this:

```
[
  { "type": "title",
    "title": "Test application" },
  { "type": "options",
    "title": "My first key",
    "desc": "Description of my first key",
    "section": "section1",
```

(continues on next page)

```

        "key": "key1",
        "options": ["value1", "value2", "another value"] },

    { "type": "numeric",
      "title": "My second key",
      "desc": "Description of my second key",
      "section": "section1",
      "key": "key2" }
]

```

Then, we can create a panel using this JSON to automatically create all the options and link them to our *App.config* ConfigParser instance:

```

class TestApp(App):
    # ...
    def build_settings(self, settings):
        jsondata = """... put the json data here ..."""
        settings.add_json_panel('Test application',
                                self.config, data=jsondata)

```

That's all! Now you can press F1 (default keystroke) to toggle the settings panel or press the “settings” key on your android device. You can manually call *App.open_settings()* and *App.close_settings()* if you want to handle this manually. Every change in the panel is automatically saved in the config file.

You can also use *App.build_settings()* to modify properties of the settings panel. For instance, the default panel has a sidebar for switching between json panels whose width defaults to 200dp. If you'd prefer this to be narrower, you could add:

```
settings.interface.menu.width = dp(100)
```

to your *build_settings()* method.

You might want to know when a config value has been changed by the user in order to adapt or reload your UI. You can then overload the *on_config_change()* method:

```

class TestApp(App):
    # ...
    def on_config_change(self, config, section, key, value):
        if config is self.config:
            token = (section, key)
            if token == ('section1', 'key1'):
                print('Our key1 has been changed to', value)
            elif token == ('section1', 'key2'):
                print('Our key2 has been changed to', value)

```

The Kivy configuration panel is added by default to the settings instance. If you don't want this panel, you can declare your Application as follows:

```

class TestApp(App):
    use_kivy_settings = False
    # ...

```

This only removes the Kivy panel but does not stop the settings instance from appearing. If you want to prevent the settings instance from appearing altogether, you can do this:

```

class TestApp(App):
    def open_settings(self, *largs):
        pass

```

New in version 1.0.7.

20.2.3 Profiling with on_start and on_stop

It is often useful to profile python code in order to discover locations to optimise. The standard library profilers (<http://docs.python.org/2/library/profile.html>) provides multiple options for profiling code. For profiling the entire program, the natural approaches of using profile as a module or profile's run method does not work with Kivy. It is however, possible to use `App.on_start()` and `App.on_stop()` methods:

```
import cProfile

class MyApp(App):
    def on_start(self):
        self.profile = cProfile.Profile()
        self.profile.enable()

    def on_stop(self):
        self.profile.disable()
        self.profile.dump_stats('myapp.profile')
```

This will create a file called *myapp.profile* when you exit your app.

20.2.4 Customising layout

You can choose different settings widget layouts by setting `App.settings_cls`. By default, this is a `Settings` class which provides the pictured sidebar layout, but you could set it to any of the other layouts provided in `kivy.uix.settings` or create your own. See the module documentation for `kivy.uix.settings` for more information.

You can customise how the settings panel is displayed by overriding `App.display_settings()` which is called before displaying the settings panel on the screen. By default, it simply draws the panel on top of the window, but you could modify it to (for instance) show the settings in a `Popup` or add it to your app's `ScreenManager` if you are using one. If you do so, you should also modify `App.close_settings()` to exit the panel appropriately. For instance, to have the settings panel appear in a popup you can do:

```
def display_settings(self, settings):
    try:
        p = self.settings_popup
    except AttributeError:
        self.settings_popup = Popup(content=settings,
                                    title='Settings',
                                    size_hint=(0.8, 0.8))

        p = self.settings_popup
    if p.content is not settings:
        p.content = settings
    p.open()

def close_settings(self, *args):
    try:
        p = self.settings_popup
        p.dismiss()
    except AttributeError:
        pass # Settings popup doesn't exist
```

Finally, if you want to replace the current settings panel widget, you can remove the internal references to it using `App.destroy_settings()`. If you have modified `App.display_settings()`, you should be careful to detect if the settings panel has been replaced.

20.2.5 Pause mode

New in version 1.1.0.

On tablets and phones, the user can switch at any moment to another application. By default, your application will close and the `App.on_stop()` event will be fired.

If you support Pause mode, when switching to another application, your application will wait indefinitely until the user switches back to your application. There is an issue with OpenGL on Android devices: it is not guaranteed that the OpenGL ES Context will be restored when your app resumes. The mechanism for restoring all the OpenGL data is not yet implemented in Kivy.

The currently implemented Pause mechanism is:

1. Kivy checks every frame if Pause mode is activated by the Operating System due to the user switching to another application, a phone shutdown or any other reason.
2. `App.on_pause()` is called:
3. If False is returned, then `App.on_stop()` is called.
4. If True is returned (default case), the application will sleep until the OS resumes our App.
5. When the app is resumed, `App.on_resume()` is called.
6. If our app memory has been reclaimed by the OS, then nothing will be called.

Here is a simple example of how `on_pause()` should be used:

```
class TestApp(App):  
  
    def on_pause(self):  
        # Here you can save data if needed  
        return True  
  
    def on_resume(self):  
        # Here you can check if any data needs replacing (usually nothing)  
        pass
```

Warning: Both `on_pause` and `on_stop` must save important data because after `on_pause` is called, `on_resume` may not be called at all.

20.2.6 Asynchronous app

In addition to running an app normally, Kivy can be run within an async event loop such as provided by the standard library `asyncio` package or the `trio` package (highly recommended).

Background

Normally, when a Kivy app is run, it blocks the thread that runs it until the app exits. Internally, at each clock iteration it executes all the app callbacks, handles graphics and input, and idles by sleeping for any remaining time.

To be able to run asynchronously, the Kivy app may not sleep, but instead must release control of the running context to the asynchronous event loop running the Kivy app. We do this when idling by calling the appropriate functions of the `async` package being used instead of sleeping.

Async configuration

To run a Kivy app asynchronously, either the `async_runTouchApp()` or `App.async_run()` coroutine must be scheduled to run in the event loop of the `async` library being used.

The environmental variable `KIVY_EVENTLOOP` or the `async_lib` parameter in `async_runTouchApp()` and `App.async_run()` set the `async` library that Kivy uses internally when the app is run with `async_runTouchApp()` and `App.async_run()`. It can be set to one of “`asyncio`” when the standard library `asyncio` is used, or “`trio`” if the `trio` library is used. If the environment variable is not set and `async_lib` is not provided, the stdlib `asyncio` is used.

`init_async_lib()` can also be directly called to set the `async` library to use, but it may only be called before the app has begun running with `async_runTouchApp()` or `App.async_run()`.

To run the app asynchronously, one schedules `async_runTouchApp()` or `App.async_run()` to run within the given library’s `async` event loop as in the examples shown below. Kivy is then treated as just another coroutine that the given library runs in its event loop. Internally, Kivy will use the specified `async` library’s API, so `KIVY_EVENTLOOP` or `async_lib` must match the `async` library that is running Kivy.

For a fuller basic and more advanced examples, see the demo apps in `examples/async`.

Asyncio example ~~~~~

```
import asyncio

from kivy.app import async_runTouchApp
from kivy.uix.label import Label

loop = asyncio.get_event_loop()
loop.run_until_complete(
    async_runTouchApp(Label(text='Hello, World!'), async_lib='asyncio'))
loop.close()
```

Trio example ~~~~~

```
import trio

from kivy.app import async_runTouchApp
from kivy.uix.label import Label

trio.run(async_runTouchApp, Label(text='Hello, World!'), async_lib='trio')
```

20.2.7 Interacting with Kivy app from other coroutines

It is fully safe to interact with any kivy object from other coroutines running within the same `async` event loop. This is because they are all running from the same thread and the other coroutines are only executed when Kivy is idling.

Similarly, the kivy callbacks may safely interact with objects from other coroutines running in the same event loop. Normal single threaded rules apply to both case.

New in version 2.0.0.

class `kivy.app.App`(***kwargs*)
Bases: `kivy.event.EventDispatcher`

Application class, see module documentation for more information.

Events

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

on_stop: Fired when the application stops.

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

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

Changed in version 1.7.0: Parameter `kv_file` added.

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

async_run(*async_lib=None*)

Identical to `run()`, but is a coroutine and can be scheduled in a running async event loop.

See `kivy.app` for example usage.

New in version 2.0.0.

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.

build_config(*config*)

New in version 1.0.7.

This method is called before the application is initialized to construct your `ConfigParser` object. This is where you can put any default section / key / value for your config. If anything is set, the configuration will be automatically saved in the file returned by `get_application_config()`.

Parameters

config: `ConfigParser` Use this to add default section / key / value items

build_settings(*settings*)

New in version 1.0.7.

This method is called when the user (or you) want to show the application settings. It is called once when the settings panel is first opened, after which the panel is cached. It may be called again if the cached settings panel is removed by `destroy_settings()`.

You can use this method to add settings panels and to customise the settings widget e.g. by changing the sidebar width. See the module documentation for full details.

Parameters

settings: `Settings` Settings instance for adding panels

close_settings(**largs*)

Close the previously opened settings panel.

Returns True if the settings has been closed.

config = None

Returns an instance of the *ConfigParser* for the application configuration. You can use this to query some config tokens in the *build()* method.

create_settings()

Create the settings panel. This method will normally be called only one time per application life-time and the result is cached internally, but it may be called again if the cached panel is removed by *destroy_settings()*.

By default, it will build a settings panel according to *settings_cls*, call *build_settings()*, add a Kivy panel if *use_kivy_settings* is True, and bind to *on_close*/*on_config_change*.

If you want to plug your own way of doing settings, without the Kivy panel or close/config change events, this is the method you want to overload.

New in version 1.8.0.

destroy_settings()

New in version 1.8.0.

Dereferences the current settings panel if one exists. This means that when *App.open_settings()* is next run, a new panel will be created and displayed. It doesn't affect any of the contents of the panel, but lets you (for instance) refresh the settings panel layout if you have changed the settings widget in response to a screen size change.

If you have modified *open_settings()* or *display_settings()*, you should be careful to correctly detect if the previous settings widget has been destroyed.

directory

New in version 1.0.7.

Return the directory where the application lives.

display_settings(settings)

New in version 1.8.0.

Display the settings panel. By default, the panel is drawn directly on top of the window. You can define other behaviour by overriding this method, such as adding it to a *ScreenManager* or *Popup*.

You should return True if the display is successful, otherwise False.

Parameters

settings: Settings You can modify this object in order to modify the settings display.

get_application_config(defaultpath='%s/(appname)s.ini')

Return the filename of your application configuration. Depending on the platform, the application file will be stored in different locations:

- on iOS: <appdir>/Documents/.<appname>.ini
- on Android: <user_data_dir>/.<appname>.ini
- otherwise: <appdir>/<appname>.ini

When you are distributing your application on Desktops, please note that if the application is meant to be installed system-wide, the user might not have write-access to the application directory. If you want to store user settings, you should overload this method and change the default behavior to save the configuration file in the user directory.

```
class TestApp(App):
    def get_application_config(self):
        return super(TestApp, self).get_application_config(
            '~/.{appname}s.ini')
```

Some notes:

- The tilde '~' will be expanded to the user directory.
- %(appdir)s will be replaced with the application *directory*
- %(appname)s will be replaced with the application *name*

New in version 1.0.7.

Changed in version 1.4.0: Customized the defaultpath for iOS and Android platforms. Added a defaultpath parameter for desktop OS's (not applicable to iOS and Android.)

Changed in version 1.11.0: Changed the Android version to make use of the *user_data_dir* and added a missing dot to the iOS config file name.

get_application_icon()

Return the icon of the application.

get_application_name()

Return the name of the application.

static get_running_app()

Return the currently running application instance.

New in version 1.1.0.

icon

Icon of your application. The icon can be located in the same directory as your main file. You can set this as follows:

```
class MyApp(App):
    def build(self):
        self.icon = 'myicon.png'
```

New in version 1.0.5.

Changed in version 1.8.0: *icon* is now a *StringProperty*. Don't set the icon in the class as previously stated in the documentation.

Note: For Kivy prior to 1.8.0, you need to set this as follows:

```
class MyApp(App):
    icon = 'customicon.png'
```

Recommended 256x256 or 1024x1024? for GNU/Linux and Mac OSX 32x32 for Windows7 or less. <= 256x256 for windows 8 256x256 does work (on Windows 8 at least), but is scaled down and doesn't look as good as a 32x32 icon.

kv_directory

Path of the directory where application kv is stored, defaults to None

New in version 1.8.0.

If a kv_directory is set, it will be used to get the initial kv file. By default, the file is assumed to be in the same directory as the current App definition file.

kv_file

Filename of the Kv file to load, defaults to None.

New in version 1.8.0.

If a kv_file is set, it will be loaded when the application starts. The loading of the “default” kv file will be prevented.

load_config()

(internal) This function is used for returning a ConfigParser with the application configuration. It's doing 3 things:

1. Creating an instance of a ConfigParser
2. Loading the default configuration by calling *build_config()*, then
3. If it exists, it loads the application configuration file, otherwise it creates one.

Returns *ConfigParser* instance

load_kv(filename=None)

This method is invoked the first time the app is being run if no widget tree has been constructed before for this app. This method then looks for a matching kv file in the same directory as the file that contains the application class.

For example, say you have a file named main.py that contains:

```
class ShowcaseApp(App):  
    pass
```

This method will search for a file named *showcase.kv* in the directory that contains main.py. The name of the kv file has to be the lowercase name of the class, without the 'App' postfix at the end if it exists.

You can define rules and a root widget in your kv file:

```
<ClassName>: # this is a rule  
...  
  
ClassName: # this is a root widget  
...
```

There must be only one root widget. See the *Kivy Language* documentation for more information on how to create kv files. If your kv file contains a root widget, it will be used as self.root, the root widget for the application.

Note: This function is called from *run()*, therefore, any widget whose styling is defined in this kv file and is created before *run()* is called (e.g. in *__init__*), won't have its styling applied. Note that *build()* is called after *load_kv* has been called.

name

New in version 1.0.7.

Return the name of the application based on the class name.

on_config_change(config, section, key, value)

Event handler fired when a configuration token has been changed by the settings page.

Changed in version 1.10.1: Added corresponding on_config_change event.

on_pause()

Event handler called when Pause mode is requested. You should return True if your app can go into Pause mode, otherwise return False and your application will be stopped.

You cannot control when the application is going to go into this mode. It's determined by the Operating System and mostly used for mobile devices (android/ios) and for resizing.

The default return value is True.

New in version 1.1.0.

Changed in version 1.10.0: The default return value is now True.

on_resume()

Event handler called when your application is resuming from the Pause mode.

New in version 1.1.0.

Warning: When resuming, the OpenGL Context might have been damaged / freed. This is where you can reconstruct some of your OpenGL state e.g. FBO content.

on_start()

Event handler for the *on_start* event which is fired after initialization (after *build()* has been called) but before the application has started running.

on_stop()

Event handler for the *on_stop* event which is fired when the application has finished running (i.e. the window is about to be closed).

open_settings(*args)

Open the application settings panel. It will be created the very first time, or recreated if the previously cached panel has been removed by *destroy_settings()*. The settings panel will be displayed with the *display_settings()* method, which by default adds the settings panel to the Window attached to your application. You should override that method if you want to display the settings panel differently.

Returns True if the settings has been opened.

options = None

Options passed to the *__init__* of the App

root = None

The *root* widget returned by the *build()* method or by the *load_kv()* method if the kv file contains a root widget.

root_window

New in version 1.9.0.

Returns the root window instance used by *run()*.

run()

Launches the app in standalone mode.

settings_cls

New in version 1.8.0.

The class used to construct the settings panel and the instance passed to *build_config()*. You should use either *Settings* or one of the provided subclasses with different layouts (*SettingsWithSidebar*, *SettingsWithSpinner*, *SettingsWithTabbedPanel*, *SettingsWithNoMenu*). You can also create your own Settings subclass. See the documentation of *Settings* for more information.

`settings_cls` is an *ObjectProperty* and defaults to *SettingsWithSpinner* which displays settings panels with a spinner to switch between them. If you set a string, the *Factory* will be used to resolve the class.

stop(*largs)

Stop the application.

If you use this method, the whole application will stop by issuing a call to *stopTouchApp()*.

title

Title of your application. You can set this as follows:

```
class MyApp(App):
    def build(self):
        self.title = 'Hello world'
```

New in version 1.0.5.

Changed in version 1.8.0: *title* is now a *StringProperty*. Don't set the title in the class as previously stated in the documentation.

Note: For Kivy < 1.8.0, you can set this as follows:

```
class MyApp(App):
    title = 'Custom title'
```

If you want to dynamically change the title, you can do:

```
from kivy.base import EventLoop
EventLoop.window.title = 'New title'
```

use_kivy_settings = True

New in version 1.0.7.

If True, the application settings will also include the Kivy settings. If you don't want the user to change any kivy settings from your settings UI, change this to False.

user_data_dir

New in version 1.7.0.

Returns the path to the directory in the users file system which the application can use to store additional data.

Different platforms have different conventions with regards to where the user can store data such as preferences, saved games and settings. This function implements these conventions. The `<app_name>` directory is created when the property is called, unless it already exists.

On iOS, `~/Documents/<app_name>` is returned (which is inside the app's sandbox).

On Windows, `%APPDATA%/<app_name>` is returned.

On OS X, `~/Library/Application Support/<app_name>` is returned.

On Linux, `$XDG_CONFIG_HOME/<app_name>` is returned.

On Android, *Context.GetFilesDir* is returned.

Changed in version 1.11.0: On Android, this function previously returned `/sd-card/<app_name>`. This folder became read-only by default in Android API 26 and the `user_data_dir` has therefore been moved to a writeable location.

kivy.app.runTouchApp(*widget=None, slave=False*)

Static main function that starts the application loop. You can access some magic via the following arguments:

See [kivy.app](#) for example usage.

Parameters

<empty> To make dispatching work, you need at least one input listener. If not, application will leave. (MTWindow act as an input listener)

widget If you pass only a widget, a MTWindow will be created and your widget will be added to the window as the root widget.

slave No event dispatching is done. This will be your job.

widget + slave No event dispatching is done. This will be your job but we try to get the window (must be created by you beforehand) and add the widget to it. Very useful for embedding Kivy in another toolkit. (like Qt, check kivy-designed)

kivy.app.async_runTouchApp(*widget=None, slave=False, async_lib=None*)

Identical to [runTouchApp\(\)](#) but instead it is a coroutine that can be run in an existing async event loop.

async_lib is the async library to use. See [kivy.app](#) for details and example usage.

New in version 2.0.0.

kivy.app.stopTouchApp()

Stop the current application by leaving the main loop.

See [kivy.app](#) for example usage.

20.3 Asynchronous data loader

This is the Asynchronous Loader. You can use it to load an image and use it, even if data are not yet available. You must specify a default loading image when using the loader:

```
from kivy.loader import Loader
image = Loader.image('mysprite.png')
```

You can also load an image from a url:

```
image = Loader.image('http://mysite.com/test.png')
```

If you want to change the default loading image, you can do:

```
Loader.loading_image = Image('another_loading.png')
```

20.3.1 Tweaking the asynchronous loader

New in version 1.6.0.

You can tweak the loader to provide a better user experience or more performance, depending of the images you are going to load. Take a look at the parameters:

- **Loader.num_workers** - define the number of threads to start for loading images.
- **Loader.max_upload_per_frame** - define the maximum image uploads in GPU to do per frame.

class kivy.loader.LoaderBase

Bases: builtins.object

Common base for the Loader and specific implementations. By default, the Loader will be the best available loader implementation.

The `_update()` function is called every 1 / 25.s or each frame if we have less than 25 FPS.

error_image

Image used for error. You can change it by doing:

```
Loader.error_image = 'error.png'
```

Changed in version 1.6.0: Not readonly anymore.

image(filename, load_callback=None, post_callback=None, **kwargs)

Load a image using the Loader. A ProxyImage is returned with a loading image. You can use it as follows:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.loader import Loader

class TestApp(App):
    def _image_loaded(self, proxyImage):
        if proxyImage.image.texture:
            self.image.texture = proxyImage.image.texture

    def build(self):
        proxyImage = Loader.image("myPic.jpg")
        proxyImage.bind(on_load=self._image_loaded)
        self.image = Image()
        return self.image

TestApp().run()
```

In order to cancel all background loading, call `Loader.stop()`.

loading_image

Image used for loading. You can change it by doing:

```
Loader.loading_image = 'loading.png'
```

Changed in version 1.6.0: Not readonly anymore.

max_upload_per_frame

The number of images to upload per frame. By default, we'll upload only 2 images to the GPU per frame. If you are uploading many small images, you can easily increase this parameter to 10 or more. If you are loading multiple full HD images, the upload time may have consequences and block the application. If you want a smooth experience, use the default.

As a matter of fact, a Full-HD RGB image will take ~6MB in memory, so it may take time. If you have activated `mipmap=True` too, then the GPU must calculate the mipmap of these big images too, in real time. Then it may be best to reduce the `max_upload_per_frame` to 1 or 2. If you want to get rid of that (or reduce it a lot), take a look at the DDS format.

New in version 1.6.0.

num_workers

Number of workers to use while loading (used only if the loader implementation supports it). This setting impacts the loader only on initialization. Once the loader is started, the setting has no impact:

```
from kivy.loader import Loader
Loader.num_workers = 4
```

The default value is 2 for giving a smooth user experience. You could increase the number of workers, then all the images will be loaded faster, but the user will not be able to use the application while loading. Prior to 1.6.0, the default number was 20, and loading many full-hd images was completely blocking the application.

New in version 1.6.0.

pause()

Pause the loader, can be useful during interactions.

New in version 1.6.0.

resume()

Resume the loader, after a *pause()*.

New in version 1.6.0.

run(*largs)

Main loop for the loader.

start()

Start the loader thread/process.

stop()

Stop the loader thread/process.

class kivy.loader.ProxyImage(arg, **kwargs)

Bases: *kivy.core.image.Image*

Image returned by the Loader.image() function.

Properties

loaded: bool, defaults to False This value may be True if the image is already cached.

Events

on_load Fired when the image is loaded or changed.

on_error Fired when the image cannot be loaded. *error*: Exception data that occurred

20.4 Atlas

New in version 1.1.0.

Atlas manages texture atlases: packing multiple textures into one. With it, you reduce the number of images loaded and speedup the application loading. This module contains both the Atlas class and command line processing for creating an atlas from a set of individual PNG files. The command line section requires the Pillow library, or the defunct Python Imaging Library (PIL), to be installed.

An Atlas is composed of 2 or more files:

- a json file (.atlas) that contains the image file names and texture locations of the atlas.
- one or multiple image files containing textures referenced by the .atlas file.

20.4.1 Definition of .atlas files

A file with `<basename>.atlas` is a json file formatted like this:

```
{
  "<basename>-<index>.png": {
    "id1": [ <x>, <y>, <width>, <height> ],
    "id2": [ <x>, <y>, <width>, <height> ],
    # ...
  },
  # ...
}
```

Example from the Kivy `data/images/defaulttheme.atlas`:

```
{
  "defaulttheme-0.png": {
    "progressbar_background": [431, 224, 59, 24],
    "image-missing": [253, 344, 48, 48],
    "filechooser_selected": [1, 207, 118, 118],
    "bubble_btn": [83, 174, 32, 32],
    # ... and more ...
  }
}
```

In this example, “defaulttheme-0.png” is a large image, with the pixels in the rectangle from (431, 224) to (431 + 59, 224 + 24) usable as `atlas://data/images/defaulttheme/progressbar_background` in any image parameter.

20.4.2 How to create an Atlas

Warning: The atlas creation requires the Pillow library (or the defunct Imaging/PIL library). This requirement will be removed in the future when the Kivy core Image is able to support loading, blitting, and saving operations.

You can directly use this module to create atlas files with this command:

```
$ python -m kivy.atlas <basename> <size> <list of images...>
```

Let’s say you have a list of images that you want to put into an Atlas. The directory is named `images` with lots of 64x64 png files inside:

```
$ ls
images
$ cd images
$ ls
bubble.png bubble-red.png button.png button-down.png
```

You can combine all the png’s into one and generate the atlas file with:

```
$ python -m kivy.atlas myatlas 256x256 *.png
Atlas created at myatlas.atlas
1 image has been created
$ ls
bubble.png bubble-red.png button.png button-down.png myatlas.atlas
myatlas-0.png
```

As you can see, we get 2 new files: `myatlas.atlas` and `myatlas-0.png`. `myatlas-0.png` is a new 256x256 .png composed of all your images. If the size you specify is not large enough to fit all of the source images, more atlas images will be created as required e.g. `myatlas-1.png`, `myatlas-2.png` etc.

Note: When using this script, the ids referenced in the atlas are the base names of the images without the extension. So, if you are going to name a file `../images/button.png`, the id for this image will be `button`.

If you need path information included, you should include `use_path` as follows:

```
$ python -m kivy.atlas -- --use_path myatlas 256 *.png
```

In which case the id for `../images/button.png` will be `images_button`

20.4.3 How to use an Atlas

Usually, you would specify the images by supplying the path:

```
a = Button(background_normal='images/button.png',
            background_down='images/button_down.png')
```

In our previous example, we have created the atlas containing both images and put them in `images/myatlas.atlas`. You can use url notation to reference them:

```
a = Button(background_normal='atlas://images/myatlas/button',
            background_down='atlas://images/myatlas/button_down')
```

In other words, the path to the images is replaced by:

```
atlas://path/to/myatlas/id
# will search for the `path/to/myatlas.atlas` and get the image `id`
```

Note: In the atlas url, there is no need to add the `.atlas` extension. It will be automatically append to the filename.

20.4.4 Manual usage of the Atlas

```
>>> from kivy.atlas import Atlas
>>> atlas = Atlas('path/to/myatlas.atlas')
>>> print(atlas.textures.keys())
['bubble', 'bubble-red', 'button', 'button-down']
>>> print(atlas['button'])
<kivy.graphics.texture.TextureRegion object at 0x2404d10>
```

class `kivy.atlas.Atlas`(*filename*)

Bases: `kivy.event.EventDispatcher`

Manage texture atlas. See module documentation for more information.

static `create`(*outname, filenames, size, padding=2, use_path=False*)

This method can be used to create an atlas manually from a set of images.

Parameters

outname: str Basename to use for .atlas creation and -<idx>.png associated images.

filenames: list List of filenames to put in the atlas.

size: int or list (width, height) Size of the atlas image. If the size is not large enough to fit all of the source images, more atlas images will be created as required.

padding: int, defaults to 2 Padding to put around each image.

Be careful. If you're using a padding < 2, you might have issues with the borders of the images. Because of the OpenGL linearization, it might use the pixels of the adjacent image.

If you're using a padding >= 2, we'll automatically generate a "border" of 1px around your image. If you look at the result, don't be scared if the image inside is not exactly the same as yours :).

use_path: bool, defaults to False If True, the relative path of the source png file names will be included in the atlas ids rather than just in the file names. Leading dots and slashes will be excluded and all other slashes in the path will be replaced with underscores. For example, if *use_path* is False (the default) and the file name is `../data/tiles/green_grass.png`, the id will be `green_grass`. If *use_path* is True, it will be `data_tiles_green_grass`.

Changed in version 1.8.0: Parameter *use_path* added

filename

Filename of the current Atlas.

filename is an *AliasProperty* and defaults to None.

original_textures

List of original atlas textures (which contain the *textures*).

original_textures is a *ListProperty* and defaults to [].

New in version 1.9.1.

textures

List of available textures within the atlas.

textures is a *DictProperty* and defaults to {}.

20.5 Cache manager

The cache manager can be used to store python objects attached to a unique key. The cache can be controlled in two ways: with a object limit or a timeout.

For example, we can create a new cache with a limit of 10 objects and a timeout of 5 seconds:

```
# register a new Cache
Cache.register('mycache', limit=10, timeout=5)

# create an object + id
key = 'objectid'
instance = Label(text=text)
Cache.append('mycache', key, instance)

# retrieve the cached object
instance = Cache.get('mycache', key)
```

If the instance is NULL, the cache may have trashed it because you've not used the label for 5 seconds and you've reach the limit.

class kivy.cache.Cache

Bases: `builtins.object`

See module documentation for more information.

static `append(category, key, obj, timeout=None)`

Add a new object to the cache.

Parameters

category: str Identifier of the category.

key: str Unique identifier of the object to store.

obj: object Object to store in cache.

timeout: double (optional) Time after which to delete the object if it has not been used. If None, no timeout is applied.

static `get(category, key, default=None)`

Get a object from the cache.

Parameters

category: str Identifier of the category.

key: str Unique identifier of the object in the store.

default: anything, defaults to None Default value to be returned if the key is not found.

static `get_lastaccess(category, key, default=None)`

Get the objects last access time in the cache.

Parameters

category: str Identifier of the category.

key: str Unique identifier of the object in the store.

default: anything, defaults to None Default value to be returned if the key is not found.

static `get_timestamp(category, key, default=None)`

Get the object timestamp in the cache.

Parameters

category: str Identifier of the category.

key: str Unique identifier of the object in the store.

default: anything, defaults to None Default value to be returned if the key is not found.

static `print_usage()`

Print the cache usage to the console.

static `register(category, limit=None, timeout=None)`

Register a new category in the cache with the specified limit.

Parameters

category: str Identifier of the category.

limit: int (optional) Maximum number of objects allowed in the cache. If None, no limit is applied.

timeout: double (optional) Time after which to delete the object if it has not been used. If None, no timeout is applied.

static remove(category, key=None)

Purge the cache.

Parameters

category: str Identifier of the category.

key: str (optional) Unique identifier of the object in the store. If this argument is not supplied, the entire category will be purged.

20.6 Clock object

The **Clock** object allows you to schedule a function call in the future; once or repeatedly at specified intervals. You can get the time elapsed between the scheduling and the calling of the callback via the *dt* argument:

```
# dt means delta-time
def my_callback(dt):
    pass

# call my_callback every 0.5 seconds
Clock.schedule_interval(my_callback, 0.5)

# call my_callback in 5 seconds
Clock.schedule_once(my_callback, 5)

# call my_callback as soon as possible (usually next frame.)
Clock.schedule_once(my_callback)
```

Note: If the callback returns False, the schedule will be canceled and won't repeat.

If you want to schedule a function to call with default arguments, you can use the **functools.partial** python module:

```
from functools import partial

def my_callback(value, key, *largs):
    pass

Clock.schedule_interval(partial(my_callback, 'my value', 'my key'), 0.5)
```

Conversely, if you want to schedule a function that doesn't accept the *dt* argument, you can use a **lambda** expression to write a short function that does accept *dt*. For Example:

```
def no_args_func():
    print("I accept no arguments, so don't schedule me in the clock")

Clock.schedule_once(lambda dt: no_args_func(), 0.5)
```

Note: You cannot unschedule an anonymous function unless you keep a reference to it. It's better to add **args* to your function definition so that it can be called with an arbitrary number of parameters.

Important: The callback is weak-referenced: you are responsible for keeping a reference to your original object/callback. If you don't keep a reference, the ClockBase will never execute your callback. For example:

```
class Foo(object):
    def start(self):
        Clock.schedule_interval(self.callback, 0.5)

    def callback(self, dt):
        print('In callback')

# A Foo object is created and the method start is called.
# Because no reference is kept to the instance returned from Foo(),
# the object will be collected by the Python Garbage Collector and
# your callback will be never called.
Foo().start()

# So you should do the following and keep a reference to the instance
# of foo until you don't need it anymore!
foo = Foo()
foo.start()
```

20.6.1 Schedule before frame

New in version 1.0.5.

Sometimes you need to schedule a callback BEFORE the next frame. Starting from 1.0.5, you can use a timeout of -1:

```
Clock.schedule_once(my_callback, 0) # call after the next frame
Clock.schedule_once(my_callback, -1) # call before the next frame
```

The Clock will execute all the callbacks with a timeout of -1 before the next frame even if you add a new callback with -1 from a running callback. However, **Clock** has an iteration limit for these callbacks: it defaults to 10.

If you schedule a callback that schedules a callback that schedules a ... etc more than 10 times, it will leave the loop and send a warning to the console, then continue after the next frame. This is implemented to prevent bugs from hanging or crashing the application.

If you need to increase the limit, set the `max_iteration` property:

```
from kivy.clock import Clock
Clock.max_iteration = 20
```

20.6.2 Triggered Events

New in version 1.0.5.

A triggered event is a way to defer a callback. It functions exactly like `schedule_once()` and `schedule_interval()` except that it doesn't immediately schedule the callback. Instead, one schedules the callback using the **ClockEvent** returned by it. This ensures that you can call the event multiple times but it won't be scheduled more than once. This is not the case with `Clock.schedule_once()`:

```

# will run the callback twice before the next frame
Clock.schedule_once(my_callback)
Clock.schedule_once(my_callback)

# will run the callback once before the next frame
event = Clock.create_trigger(my_callback)
event()
event()

# will also run the callback only once before the next frame
event = Clock.schedule_once(my_callback) # now it's already scheduled
event() # won't be scheduled again
event()

```

In addition, it is more convenient to create and bind to the triggered event than using `Clock.schedule_once()` in a function:

```

from kivy.clock import Clock
from kivy.uix.widget import Widget

class Sample(Widget):
    def __init__(self, **kwargs):
        self._trigger = Clock.create_trigger(self.cb)
        super(Sample, self).__init__(**kwargs)
        self.bind(x=self._trigger, y=self._trigger)

    def cb(self, *largs):
        pass

```

Even if `x` and `y` changes within one frame, the callback is only run once.

`CyClockBase.create_trigger()` has a `timeout` parameter that behaves exactly like `CyClockBase.schedule_once()`.

Changed in version 1.10.0: `CyClockBase.create_trigger()` now has a `interval` parameter. If `False`, the default, it'll create an event similar to `CyClockBase.schedule_once()`. Otherwise it'll create an event similar to `CyClockBase.schedule_interval()`.

20.6.3 Unsheduling

An event scheduled with `CyClockBase.schedule_once()`, `CyClockBase.schedule_interval()`, or with `CyClockBase.create_trigger()` and then triggered can be unscheduled in multiple ways. E.g:

```

def my_callback(dt):
    pass

# call my_callback every 0.5 seconds
event = Clock.schedule_interval(my_callback, 0.5)

# call my_callback in 5 seconds
event2 = Clock.schedule_once(my_callback, 5)

event_trig = Clock.create_trigger(my_callback, 5)
event_trig()

# unschedule using cancel
event.cancel()

```

(continues on next page)

```
# unschedule using Clock.unschedule
Clock.unschedule(event2)

# unschedule using Clock.unschedule with the callback
# NOT RECOMMENDED
Clock.unschedule(my_callback)
```

The best way to unschedule a callback is with `ClockEvent.cancel()`. `CyClockBase.unschedule()` is mainly an alias for that for that function. However, if the original callback itself is passed to `CyClockBase.unschedule()`, it'll unschedule all instances of that callback (provided `all` is `True`, the default, other just the first match is removed).

Calling `CyClockBase.unschedule()` on the original callback is highly discouraged because it's significantly slower than when using the event.

20.6.4 Threading and Callback Order

Beginning with 1.10.0, all the events scheduled for the same frame, e.g. all the events scheduled in the same frame with a `timeout` of 0, will be executed in the order they were scheduled.

Also, all the scheduling and canceling methods are fully thread safe and can be safely used from external threads.

As a consequence, calling `CyClockBase.unschedule()` with the original callback is now significantly slower and highly discouraged. Instead, the returned events should be used to cancel. As a tradeoff, all the other methods are now significantly faster than before.

20.6.5 Advanced Clock Details

The following section goes into the internal kivy clock details as well as the various clock options. It is meant only for advanced users.

Fundamentally, the Kivy clock attempts to execute any scheduled callback rhythmically as determined by the specified fps (frame per second, see `maxfps` in `config`). That is, ideally, given e.g. a desired fps of 30, the clock will execute the callbacks at intervals of $1 / 30$ seconds, or every 33.33 ms. All the callbacks in a frame are given the same timestamp, i.e. the `dt` passed to the callback are all the same and it's the difference in time between the start of this and the previous frame.

Because of inherent indeterminism, the frames do not actually occur exactly at intervals of the fps and `dt` may be under or over the desired fps. Also, once the timeout is "close enough" to the desired timeout, as determined internally, Kivy will execute the callback in the current frame even when the "actual time" has not elapsed the `timeout` amount.

Kivy offers now, since 1.10.0, multiple clocks with different behaviors.

Default Clock

The default clock (`default`) behaves as described above. When a callback with a timeout of zero or non-zero is scheduled, they are executed at the frame that is near the timeout, which is a function of the fps. So a timeout of zero would still result in a delay of one frame or about $1 / \text{fps}$, typically a bit less but sometimes more depending on the CPU usage of the other events scheduled for that frame.

In a test using a fps of 30, a callback with a timeout of 0, 0.001, and 0.05, resulted in a mean callback delay of 0.02487, 0.02488, and 0.05011 seconds, respectively. When tested with a fps of 600 the delay for 0.05 was similar, except the standard deviation was reduced resulting in overall better accuracy.

Interruptible Clock

The default clock suffers from the quantization problem, as frames occur only on intervals and any scheduled timeouts will not be able to occur during an interval. For example, with the timeout of 0.05, while the mean was 0.05011, its values ranged between 0.02548 - 0.07348 and a standard deviation of 0.002. Also, there's the minimum timeout of about 0.02487.

The interruptible clock (`interrupt`) will execute timeouts even during a frame. So a timeout of zero will execute as quickly as possible and similarly a non-zero timeout will be executed even during the interval.

This clock, and all the clocks described after this have an option, `ClockBaseInterruptBehavior.interupt_next_only`. When `True`, any of the behavior new behavior will only apply to the callbacks with a timeout of zero. Non-zero timeouts will behave like in the default clock. E.g. for this clock when `True`, only zero timeouts will execute during the the interval.

In a test using a fps of 30, a callback with a timeout of 0, 0.001, and 0.05, resulted in a mean callback delay of 0.00013, 0.00013, and 0.04120 seconds, respectively when `ClockBaseInterruptBehavior.interupt_next_only` was `False`. Also, compared to the default clock the standard deviation was reduced. When `ClockBaseInterruptBehavior.interupt_next_only` was `True`, the values were 0.00010, 0.02414, and 0.05034, respectively.

Free Clock

The interruptible clock may not be ideal for all cases because all the events are executed during the intervals and events are not executed anymore rhythmically as multiples of the fps. For example, there may not be any benefit for the graphics to update in a sub-interval, so the additional accuracy wastes CPU.

The Free clock (`free_all`) solves this by having `Clock.xxx_free` versions of all the Clock scheduling methods. By free, we mean the event is free from the fps because it's not fps limited. E.g. `CyClockBaseFree.create_trigger_free()` corresponds to `CyClockBase.create_trigger()`. Only when an event scheduled using the `Clock.xxx_free` methods is present will the clock interrupt and execute the events during the interval. So, if no free event is present the clock behaves like the `default` clock, otherwise it behaves like the `interrupt` clock.

In a test using a fps of 30, a callback with a timeout of 0s, 0.001s, and 0.05s, resulted in a mean callback delay of 0.00012s, 0.00017s, and 0.04121s seconds, respectively when it was a free event and 0.02403s, 0.02405s, and 0.04829s, respectively when it wasn't.

Free Only Clock

The Free clock executes all events when a free event was scheduled. This results in normal events also being execute in the middle of the interval when a free event is scheduled. For example, above, when a free event was absent, a normal event with a 0.001s timeout was delayed for 0.02405s. However, if a free event happened to be also scheduled, the normal event was only delayed 0.00014s, which may be undesirable.

The Free only clock (`free_only`) solves it by only executing free events during the interval and normal events are always executed like with the default clock. For example, in the presence of a free event, a normal event with a timeout of 0.001s still had a delay of 0.02406. So this clock, treats free and normal events independently, with normal events always being fps limited, but never the free events.

Summary

The kivy clock type to use can be set with the `kivy_clock` option the `config`. If `KIVY_CLOCK` is present in the environment it overwrites the config selection. Its possible values are as follows:

- When `kivy_clock` is `default`, the normal clock, `ClockBase`, which limits callbacks to the `maxfps` quantization - is used.
- When `kivy_clock` is `interrupt`, a interruptible clock, `ClockBaseInterrupt`, which doesn't limit any callbacks to the `maxfps` - is used. Callbacks will be executed at any time.
- When `kivy_clock` is `free_all`, a interruptible clock, `ClockBaseFreeInterruptAll`, which doesn't limit any callbacks to the `maxfps` in the presence of free events, but in their absence it limits events to the `fps` quantization interval - is used.
- When `kivy_clock` is `free_only`, a interruptible clock, `ClockBaseFreeInterruptAll`, which treats free and normal events independently; normal events are `fps` limited while free events are not - is used.

20.6.6 Async clock support

New in version 2.0.0.

Experimental async support has been added in 2.0.0. The Clock now has a `ClockBaseBehavior.async_tick()` and `ClockBaseBehavior.async_idle()` coroutine method which is used by the kivy EventLoop when the kivy EventLoop is executed in a asynchronous manner. When used, the kivy clock does not block while idling.

The async library to use is selected with the `KIVY_EVENTLOOP` environmental variable or by calling `init_async_lib()` directly. The library can be one of `"asyncio"` when the standard library `asyncio` should be used, or `"trio"` if the trio library should be used. If not set it defaults to `"asyncio"`.

See `app` for example usage.

`kivy.clock.Clock = None`

The kivy Clock instance. See module documentation for details.

class `kivy.clock.ClockEvent`(*CyClockBase clock, int loop, callback, double timeout, double starttime, cid=None, int trigger=False, **kwargs*)

Bases: `builtins.object`

This class is never created by the user; instead, kivy creates and returns an instance of this class when scheduling a callback.

An event can be triggered (scheduled) by calling it. If it's already scheduled, nothing will happen, otherwise it'll be scheduled. E.g.:

```
event = Clock.schedule_once(my_callback, .5)
event() # nothing will happen since it's already scheduled.
event.cancel() # cancel it
event() # now it's scheduled again.
```

callback

callback: object

cancel()

Cancels the callback if it was scheduled to be called. If not scheduled, nothing happens.

cid

cid: object

clock

clock: `kivy._clock.CyClockBase` The `CyClockBase` instance associated with the event.

get_callback()

Returns the callback associated with the event. Callbacks get stored with a indirect ref so that it doesn't keep objects alive. If the callback is dead, `None` is returned.

is_triggered

Returns whether the event is scheduled to have its callback executed by the kivy thread.

loop

loop: 'int' Whether this event repeats at intervals of *timeout*.

next

next: kivy._clock.ClockEvent The next *ClockEvent* in order they were scheduled.

prev

prev: kivy._clock.ClockEvent The previous *ClockEvent* in order they were scheduled.

release()

(internal method) Converts the callback into a indirect ref.

tick(double curtime)

(internal method) Processes the event for the kivy thread.

timeout

timeout: 'double' The duration after scheduling when the callback should be executed.

weak_callback

weak_callback: object

class kivy.clock.**FreeClockEvent**(*free, *args, **kwargs*)

Bases: kivy._clock.ClockEvent

CyClockBaseFree. It stores whether the event was scheduled as a free event.

free

free: 'int' Whether this event was scheduled as a free event.

class kivy.clock.**CyClockBase**(***kwargs*)

Bases: builtins.object

clock_resolution

clock_resolution: 'double' If the remaining time until the event timeout is less than *clock_resolution*,

the clock will execute the callback even if it hasn't exactly timed out.

If -1, the default, the resolution will be computed from config's maxfps. Otherwise, the provided value is used. Defaults to -1.

create_trigger(*callback, timeout=0, interval=False*)

Create a Trigger event. Check module documentation for more information.

Returns A *ClockEvent* instance. To schedule the callback of this instance, you can call it.

New in version 1.0.5.

Changed in version 1.10.0: *interval* has been added. If True, it create a event that is called every <timeout> seconds similar to *schedule_interval()*. Defaults to False.

get_events()

Returns the list of *ClockEvent* instances currently scheduled.

get_min_timeout()

Returns the remaining time since the start of the current frame for the event with the smallest timeout.

get_resolution()

Returns the minimum resolution the clock has. It's a function of *clock_resolution* and maxfps provided at the config.

max_iteration

max_iteration: 'int' The maximum number of callback iterations at the end of the frame, before the next

frame. If more iterations occur, a warning is issued.

on_schedule(event)

Function that is called internally every time an event is triggered for this clock. It takes the event as a parameter.

schedule_del_safe(callback)

Schedule a callback. Might be called from GC and cannot be cancelled.

It's unsafe to call various kinds of code, such as code with a lock, from a `__del__` or `__dealloc__` methods. Since Kivy's Clock uses a lock, it's generally unsafe to call from these methods. Instead, use this method, which is thread safe and `__del__` or `__dealloc__` safe, to schedule the callback in the kivy thread. It'll be executed in order after the normal events are processed.

The callback takes no parameters and cannot be canceled.

New in version 1.11.0.

schedule_interval(callback, timeout)

Schedule an event to be called every <timeout> seconds.

Returns A *ClockEvent* instance. As opposed to *create_trigger()* which only creates the trigger event, this method also schedules it.

schedule_once(callback, timeout=0)

Schedule an event in <timeout> seconds. If <timeout> is unspecified or 0, the callback will be called after the next frame is rendered.

Returns A *ClockEvent* instance. As opposed to *create_trigger()* which only creates the trigger event, this method also schedules it.

Changed in version 1.0.5: If the timeout is -1, the callback will be called before the next frame (at *tick_draw()*).

unsubscribe(callback, all=True)

Remove a previously scheduled event.

Parameters

callback: *ClockEvent* or a callable. If it's a *ClockEvent* instance, then the callback associated with this event will be canceled if it is scheduled.

If it's a callable, then the callable will be unscheduled if it was scheduled.

Warning: Passing the callback function rather than the returned *ClockEvent* will result in a significantly slower unscheduling.

all: bool If True and if *callback* is a callable, all instances of this callable will be unscheduled (i.e. if this callable was scheduled multiple times). Defaults to *True*.

Changed in version 1.9.0: The all parameter was added. Before, it behaved as if *all* was *True*.

class kivy.clock.CyClockBaseFree

Bases: *kivy._clock.CyClockBase*

A clock class that supports scheduling free events in addition to normal events.

Each of the *create_trigger()*, *schedule_once()*, and *schedule_interval()* methods, which create a normal event, have a corresponding method for creating a free event.

create_trigger(*callback*, *timeout*=0, *interval*=False)

create_trigger_free(*callback*, *timeout*=0, *interval*=False)

Similar to **create_trigger()**, but instead creates a free event.

get_min_free_timeout()

Returns the remaining time since the start of the current frame for the *free* event with the smallest timeout.

schedule_interval(*callback*, *timeout*)

schedule_interval_free(*callback*, *timeout*)

Similar to **schedule_interval()**, but instead creates a free event.

schedule_once(*callback*, *timeout*=0)

schedule_once_free(*callback*, *timeout*=0)

Similar to **schedule_once()**, but instead creates a free event.

class `kivy.clock.ClockBaseBehavior`(*async_lib*='asyncio', **kwargs)

Bases: `builtins.object`

The base of the kivy clock.

Parameters

async_lib: **string** The async library to use when the clock is run asynchronously. Can be one of, "asyncio" when the standard library asyncio should be used, or "trio" if the trio library should be used.

It defaults to 'asyncio' or the value in the environmental variable `KIVY_EVENTLOOP` if set. **init_async_lib()** can also be called directly to set the library.

MIN_SLEEP = **0.005**

The minimum time to sleep. If the remaining time is less than this, the event loop will continue.

async_idle()

(internal) async version of **idle()**.

async_tick()

async version of **tick()**.

frames

Number of internal frames (not necessarily drawn) from the start of the clock.

New in version 1.8.0.

frames_displayed

Number of displayed frames from the start of the clock.

frametime

Time spent between the last frame and the current frame (in seconds).

New in version 1.8.0.

get_boottime()

Get the time in seconds from the application start.

get_fps()

Get the current average FPS calculated by the clock.

get_rfps()

Get the current "real" FPS calculated by the clock. This counter reflects the real framerate displayed on the screen.

In contrast to `get_fps()`, this function returns a counter of the number of frames, not the average of frames per second.

get_time()

Get the last tick made by the clock.

idle()

(internal) waits here until the next frame.

init_async_lib(lib)

Manually sets the async library to use internally, when running in a asynchronous manner.

This can be called anytime before the kivy event loop has started, but not once the kivy App is running.

Parameters

lib: string The async library to use when the clock is run asynchronously. Can be one of, *“asyncio”* when the standard library asyncio should be used, or *“trio”* if the trio library should be used.

post_idle(ts, current)

Called after *idle()* by *tick()*.

pre_idle()

Called before *idle()* by *tick()*.

tick()

Advance the clock to the next step. Must be called every frame. The default clock has a tick() function called by the core Kivy framework.

tick_draw()

Tick the drawing counter.

time = functools.partial(<function _libc_clock_gettime_wrapper.<locals>._time>)

usleep(microseconds)

Sleeps for the number of microseconds.

class kivy.clock.ClockBaseInterruptBehavior(interupt_next_only=False, **kwargs)

Bases: *kivy.clock.ClockBaseBehavior*

A kivy clock which can be interrupted during a frame to execute events.

async_idle()

(internal) async version of *idle()*.

idle()

(internal) waits here until the next frame.

init_async_lib(lib)

Manually sets the async library to use internally, when running in a asynchronous manner.

This can be called anytime before the kivy event loop has started, but not once the kivy App is running.

Parameters

lib: string The async library to use when the clock is run asynchronously. Can be one of, *“asyncio”* when the standard library asyncio should be used, or *“trio”* if the trio library should be used.

usleep(microseconds)

Sleeps for the number of microseconds.

class kivy.clock.**ClockBaseInterruptFreeBehavior**(**kwargs)

Bases: *kivy.clock.ClockBaseInterruptBehavior*

A base class for the clock that interrupts the sleep interval for free events.

class kivy.clock.**ClockBase**(**kwargs)

Bases: *kivy.clock.ClockBaseBehavior*, kivy._clock.CyClockBase

The default kivy clock. See module for details.

usleep(microseconds)

Sleeps for the number of microseconds.

class kivy.clock.**ClockBaseInterrupt**(interrupt_next_only=False, **kwargs)

Bases: *kivy.clock.ClockBaseInterruptBehavior*, kivy._clock.CyClockBase

The interrupt kivy clock. See module for details.

class kivy.clock.**ClockBaseFreeInterruptAll**(**kwargs)

Bases: *kivy.clock.ClockBaseInterruptFreeBehavior*, kivy._clock.CyClockBaseFree

The free_all kivy clock. See module for details.

class kivy.clock.**ClockBaseFreeInterruptOnly**(**kwargs)

Bases: *kivy.clock.ClockBaseInterruptFreeBehavior*, kivy._clock.CyClockBaseFree

The free_only kivy clock. See module for details.

async_idle()

(internal) async version of *idle()*.

idle()

(internal) waits here until the next frame.

kivy.clock.**mainthread**(func)

Decorator that will schedule the call of the function for the next available frame in the mainthread. It can be useful when you use *UrlRequest* or when you do Thread programming: you cannot do any OpenGL-related work in a thread.

Please note that this method will return directly and no result can be returned:

```
@mainthread
def callback(self, *args):
    print('The request succeeded!',
          'This callback is called in the main thread.')

self.req = UrlRequest(url='http://...', on_success=callback)
```

New in version 1.8.0.

20.7 Compatibility module for Python 2.7 and >= 3.4

This module provides a set of utility types and functions for optimization and to aid in writing Python 2/3 compatible code.

kivy.compat.**PY2** = **False**

False, because we don't support Python 2 anymore.

kivy.compat.**clock**() → float

A clock with the highest available resolution on your current Operating System.

`kivy.compat.string_types`

alias of `builtins.str`

`kivy.compat.isclose()`

Determine whether two floating point numbers are close in value.

rel_tol maximum difference for being considered “close”, relative to the magnitude of the input values

abs_tol maximum difference for being considered “close”, regardless of the magnitude of the input values

Return True if a is close in value to b, and False otherwise.

For the values to be considered close, the difference between them must be smaller than at least one of the tolerances.

-inf, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and -inf are only close to themselves.

20.8 Configuration object

The *Config* object is an instance of a modified Python ConfigParser. See the *ConfigParser documentation* for more information.

Kivy has a configuration file which determines the default settings. In order to change these settings, you can alter this file manually or use the Config object. Please see the *Configure Kivy* section for more information.

20.8.1 Applying configurations

Configuration options control the initialization of the *App*. In order to avoid situations where the config settings do not work or are not applied before window creation (like setting an initial window size), *Config.set* should be used before importing any other Kivy modules. Ideally, this means setting them right at the start of your main.py script.

Alternatively, you can save these settings permanently using *Config.set* then *Config.write*. In this case, you will need to restart the app for the changes to take effect. Note that this approach will effect all Kivy apps system wide.

Please note that no underscores (_) are allowed in the section name.

20.8.2 Usage of the Config object

To read a configuration token from a particular section:

```
>>> from kivy.config import Config
>>> Config.getint('kivy', 'show_fps')
0
```

Change the configuration and save it:

```
>>> Config.set('postproc', 'retain_time', '50')
>>> Config.write()
```

For information on configuring your *App*, please see the *Application configuration* section.

Changed in version 1.7.1: The ConfigParser should work correctly with utf-8 now. The values are converted from ascii to unicode only when needed. The method get() returns utf-8 strings.

20.8.3 Changing configuration with environment variables

Since 1.11.0, it is now possible to change the configuration using environment variables. They take precedence on the loaded config.ini. The format is:

```
KCFG_<section>_<key> = <value>
```

For example:

```
KCFG_GRAPHICS_FULLSCREEN=auto ... KCFG_KIVY_LOG_LEVEL=warning ...
```

Or in your file before any kivy import:

```
import os os.environ["KCFG_KIVY_LOG_LEVEL"] = "warning"
```

If you don't want to map any environment variables, you can disable the behavior:

```
os.environ["KIVY_NO_ENV_CONFIG"] = "1"
```

20.8.4 Available configuration tokens

kivy

default_font: list Default fonts used for widgets displaying any text. It defaults to ['Roboto', 'data/fonts/Roboto-Regular.ttf', 'data/fonts/Roboto-Italic.ttf', 'data/fonts/Roboto-Bold.ttf', 'data/fonts/Roboto-BoldItalic.ttf'].

desktop: int, 0 or 1 This option controls desktop OS specific features, such as enabling drag-able scroll-bar in scroll views, disabling of bubbles in TextInput etc. 0 is disabled, 1 is enabled.

exit_on_escape: int, 0 or 1 Enables exiting kivy when escape is pressed. 0 is disabled, 1 is enabled.

pause_on_minimize: int, 0 or 1 If set to 1, the main loop is paused and the *on_pause* event is dispatched when the window is minimized. This option is intended for desktop use only. Defaults to 0.

keyboard_layout: string Identifier of the layout to use.

keyboard_mode: string Specifies the keyboard mode to use. It can be one of the following:

- '' - Let Kivy choose the best option for your current platform.
- 'system' - real keyboard.
- 'dock' - one virtual keyboard docked to a screen side.
- 'multi' - one virtual keyboard for every widget request.
- 'systemanddock' - virtual docked keyboard plus input from real keyboard.
- 'systemandmulti' - analogous.

kivy_clock: one of *default*, *interrupt*, *free_all*, *free_only* The clock type to use with kivy. See [kivy.clock](#).

log_dir: string Path of log directory.

log_enable: int, 0 or 1 Activate file logging. 0 is disabled, 1 is enabled.

log_level: string, one of 'trace', 'debug', 'info', 'warning', 'error' or 'critical' Set the minimum log level to use.

log_name: string Format string to use for the filename of log file.

log_maxfiles: int Keep log_maxfiles recent logfiles while purging the log directory. Set 'log_maxfiles' to -1 to disable logfile purging (eg keep all logfiles).

Note: You end up with 'log_maxfiles + 1' logfiles because the logger adds a new one after purging.

window_icon: string Path of the window icon. Use this if you want to replace the default pygame icon.

postproc

double_tap_distance: float Maximum distance allowed for a double tap, normalized inside the range 0 - 1000.

double_tap_time: int Time allowed for the detection of double tap, in milliseconds.

ignore: list of tuples List of regions where new touches are ignored. This configuration token can be used to resolve hotspot problems with DIY hardware. The format of the list must be:

```
ignore = [(xmin, ymin, xmax, ymax), ...]
```

All the values must be inside the range 0 - 1.

jitter_distance: int Maximum distance for jitter detection, normalized inside the range 0 - 1000.

jitter_ignore_devices: string, separated with commas List of devices to ignore from jitter detection.

retain_distance: int If the touch moves more than is indicated by retain_distance, it will not be retained. Argument should be an int between 0 and 1000.

retain_time: int Time allowed for a retain touch, in milliseconds.

triple_tap_distance: float Maximum distance allowed for a triple tap, normalized inside the range 0 - 1000.

triple_tap_time: int Time allowed for the detection of triple tap, in milliseconds.

graphics

borderless: int, one of 0 or 1 If set to 1, removes the window border/decoration. Window resizing must also be disabled to hide the resizing border.

window_state: string, one of 'visible', 'hidden', 'maximized' or 'minimized'

Sets the window state, defaults to 'visible'. This option is available only for the SDL2 window provider and it should be used on desktop OSes.

fbo: string, one of 'hardware', 'software' or 'force-hardware' Selects the FBO backend to use.

fullscreen: int or string, one of 0, 1, 'fake' or 'auto' Activate fullscreen. If set to 1, a resolution of *width* times *height* pixels will be used. If set to *auto*, your current display's resolution will be used instead. This is most likely what you want. If you want to place the window in another display, use *fake*, or set the *borderless* option from the graphics section, then adjust *width*, *height*, *top* and *left*.

height: int Height of the Window, not used if *fullscreen* is set to *auto*.

left: int Left position of the Window.

maxfps: int, defaults to 60 Maximum FPS allowed.

Warning: Setting maxfps to 0 will lead to max CPU usage.

'multisamples': int, defaults to 2 Sets the [MultiSample Anti-Aliasing \(MSAA\)](#) level. Increasing this value results in smoother graphics but at the cost of processing time.

Note: This feature is limited by device hardware support and will have no effect on devices which do not support the level of MSAA requested.

position: string, one of 'auto' or 'custom' Position of the window on your display. If *auto* is used, you have no control of the initial position: *top* and *left* are ignored.

show_cursor: int, one of 0 or 1 Set whether or not the cursor is shown on the window.

top: int Top position of the Window.

resizable: int, one of 0 or 1 If 0, the window will have a fixed size. If 1, the window will be resizable.

rotation: int, one of 0, 90, 180 or 270 Rotation of the Window.

width: int Width of the Window, not used if *fullscreen* is set to *auto*.

minimum_width: int Minimum width to restrict the window to. (sdl2 only)

minimum_height: int Minimum height to restrict the window to. (sdl2 only)

min_state_time: float, defaults to .035 Minimum time for widgets to display a given visual state. This attrib is currently used by widgets like [DropDown](#) & [ButtonBehavior](#) to make sure they display their current visual state for the given time.

allow_screensaver: int, one of 0 or 1, defaults to 1 Allow the device to show a screen saver, or to go to sleep on mobile devices. Only works for the sdl2 window provider.

input You can create new input devices using this syntax:

```
# example of input provider instance
yourid = providerid,parameters

# example for tuio provider
default = tuio,127.0.0.1:3333
mytable = tuio,192.168.0.1:3334
```

See also:

Check the providers in [kivy.input.providers](#) for the syntax to use inside the configuration file.

widgets

scroll_distance: int Default value of the [scroll_distance](#) property used by the [ScrollView](#) widget. Check the widget documentation for more information.

scroll_friction: float Default value of the [scroll_friction](#) property used by the [ScrollView](#) widget. Check the widget documentation for more information.

Deprecated since version 1.7.0: Please use [effect_cls](#) instead.

scroll_timeout: int Default value of the *scroll_timeout* property used by the *ScrollView* widget. Check the widget documentation for more information.

scroll_stoptime: int Default value of the *scroll_stoptime* property used by the *ScrollView* widget. Check the widget documentation for more information.

Deprecated since version 1.7.0: Please use *effect_cls* instead.

scroll_moves: int Default value of the *scroll_moves* property used by the *ScrollView* widget. Check the widget documentation for more information.

Deprecated since version 1.7.0: Please use *effect_cls* instead.

modules You can activate modules with this syntax:

```
modulename =
```

Anything after the = will be passed to the module as arguments. Check the specific module's documentation for a list of accepted arguments.

Changed in version 1.10.0: *min_state_time* and *allow_screensaver* have been added to the *graphics* section. *kivy_clock* has been added to the *kivy* section. *default_font* has been added to the *kivy* section.

Changed in version 1.9.0: *borderless* and *window_state* have been added to the *graphics* section. The *fake* setting of the *fullscreen* option has been deprecated, use the *borderless* option instead. *pause_on_minimize* has been added to the *kivy* section.

Changed in version 1.8.0: *systemanddock* and *systemandmulti* has been added as possible values for *keyboard_mode* in the *kivy* section. *exit_on_escape* has been added to the *kivy* section.

Changed in version 1.2.0: *resizable* has been added to *graphics* section.

Changed in version 1.1.0: *tuio* no longer listens by default. Window icons are not copied to user directory anymore. You can still set a new window icon by using the *window_icon* config setting.

Changed in version 1.0.8: *scroll_timeout*, *scroll_distance* and *scroll_friction* have been added. *list_friction*, *list_trigger_distance* and *list_friction_bound* have been removed. *keyboard_type* and *keyboard_layout* have been removed from the widget. *keyboard_mode* and *keyboard_layout* have been added to the *kivy* section.

kivy.config.Config = None

The default Kivy configuration object. This is a *ConfigParser* instance with the *name* set to 'kivy'.

```
Config = ConfigParser(name='kivy')
```

class kivy.config.ConfigParser(name="", **kwargs)

Bases: *configparser.RawConfigParser*, *builtins.object*

Enhanced *ConfigParser* class that supports the addition of default sections and default values.

By default, the kivy *ConfigParser* instance, *Config*, is named 'kivy' and the *ConfigParser* instance used by the *App.build_settings* method is named 'app'.

Parameters

name: string The name of the instance. See *name*. Defaults to ''.

Changed in version 1.9.0: Each *ConfigParser* can now be *named*. You can get the *ConfigParser* associated with a name using *get_configparser()*. In addition, you can now control the config values with *ConfigParserProperty*.

New in version 1.0.7.

add_callback(*callback*, *section=None*, *key=None*)

Add a callback to be called when a specific section or key has changed. If you don't specify a section or key, it will call the callback for all section/key changes.

Callbacks will receive 3 arguments: the section, key and value.

New in version 1.4.1.

adddefaultsection(*section*)

Add a section if the section is missing.

get(*section*, *option*, ***kwargs*)

Get an option value for a given section.

If 'vars' is provided, it must be a dictionary. The option is looked up in 'vars' (if provided), 'section', and in 'DEFAULTSECT' in that order. If the key is not found and 'fallback' is provided, it is used as a fallback value. 'None' can be provided as a 'fallback' value.

If interpolation is enabled and the optional argument 'raw' is False, all interpolations are expanded in the return values.

Arguments 'raw', 'vars', and 'fallback' are keyword only.

The section DEFAULT is special.

static get_configparser(*name*)

Returns the *ConfigParser* instance whose name is *name*, or None if not found.

Parameters

name: string The name of the *ConfigParser* instance to return.

getdefault(*section*, *option*, *defaultvalue*)

Get the value of an option in the specified section. If not found, it will return the default value.

getdefaultint(*section*, *option*, *defaultvalue*)

Get the value of an option in the specified section. If not found, it will return the default value. The value will always be returned as an integer.

New in version 1.6.0.

name

The name associated with this *ConfigParser* instance, if not ''. Defaults to ''. It can be safely changed dynamically or set to ''.

When a *ConfigParser* is given a name, that config object can be retrieved using *get_configparser()*. In addition, that config instance can also be used with a *ConfigParserProperty* instance that set its *config* value to this name.

Setting more than one *ConfigParser* with the same name will raise a *ValueError*.

read(*filename*)

Read only one filename. In contrast to the original *ConfigParser* of Python, this one is able to read only one file at a time. The last read file will be used for the *write()* method.

Changed in version 1.9.0: *read()* now calls the callbacks if read changed any values.

remove_callback(*callback*, *section=None*, *key=None*)

Removes a callback added with *add_callback()*. *remove_callback()* must be called with the same parameters as *add_callback()*.

Raises a *ValueError* if not found.

New in version 1.9.0.

set(*section, option, value*)

Functions similarly to PythonConfigParser's set method, except that the value is implicitly converted to a string.

setall(*section, keyvalues*)

Sets multiple key-value pairs in a section. keyvalues should be a dictionary containing the key-value pairs to be set.

setdefault(*section, option, value*)

Set the default value for an option in the specified section.

setdefaults(*section, keyvalues*)

Set multiple key-value defaults in a section. keyvalues should be a dictionary containing the new key-value defaults.

update_config(*filename, overwrite=False*)

Upgrade the configuration based on a new default config file. Overwrite any existing values if overwrite is True.

write()

Write the configuration to the last file opened using the *read()* method.

Return True if the write finished successfully, False otherwise.

20.9 Context

New in version 1.8.0.

Warning: This is experimental and subject to change as long as this warning notice is present.

Kivy has a few “global” instances that are used directly by many pieces of the framework: *Cache*, *Builder*, *Clock*.

TODO: document this module.

class kivy.context.**Context**(*init=False*)

Bases: *builtins.dict*

pop(*k[, d]*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

kivy.context.**register_context**(*name, cls, *args, **kwargs*)

Register a new context.

kivy.context.**get_current_context**()

Return the current context.

20.10 Event dispatcher

All objects that produce events in Kivy implement the *EventDispatcher* which provides a consistent interface for registering and manipulating event handlers.

Changed in version 1.0.9: Property discovery and methods have been moved from the *Widget* to the *EventDispatcher*.

class kivy.event.**EventDispatcher**(***kwargs*)

Bases: *kivy.event.ObjectWithUid*

See the module docstring for usage.

apply_property(**kwargs)

Adds properties at runtime to the class. The function accepts keyword arguments of the form `prop_name=prop`, where `prop` is a `Property` instance and `prop_name` is the name of the attribute of the property.

New in version 1.9.1.

Warning: This method is not recommended for common usage because you should declare the properties in your class instead of using this method.

For example:

```
>>> print(wid.property('sticks', quiet=True))
None
>>> wid.apply_property(sticks=ObjectProperty(55, max=10))
>>> print(wid.property('sticks', quiet=True))
<kivy.properties.ObjectProperty object at 0x04303130>
```

bind(**kwargs)

Bind an event type or a property to a callback.

Usage:

```
# With properties
def my_x_callback(obj, value):
    print('on object', obj, 'x changed to', value)
def my_width_callback(obj, value):
    print('on object', obj, 'width changed to', value)
self.bind(x=my_x_callback, width=my_width_callback)

# With event
def my_press_callback(obj):
    print('event on object', obj)
self.bind(on_press=my_press_callback)
```

In general, property callbacks are called with 2 arguments (the object and the property's new value) and event callbacks with one argument (the object). The example above illustrates this.

The following example demonstrates various ways of using the `bind` function in a complete application:

```
from kivy.uix.boxlayout import BoxLayout
from kivy.app import App
from kivy.uix.button import Button
from functools import partial

class DemoBox(BoxLayout):
    """
    This class demonstrates various techniques that can be used for
    ↪binding to
    ↪events. Although parts could be made more optimal, advanced Python
    ↪concepts
    are avoided for the sake of readability and clarity.
    """
    def __init__(self, **kwargs):
```

(continues on next page)

```

super(DemoBox, self).__init__(**kwargs)
self.orientation = "vertical"

# We start with binding to a normal event. The only argument
# passed to the callback is the object which we have bound to.
btn = Button(text="Normal binding to event")
btn.bind(on_press=self.on_event)

# Next, we bind to a standard property change event. This
→ typically
# passes 2 arguments: the object and the value
btn2 = Button(text="Normal binding to a property change")
btn2.bind(state=self.on_property)

# Here we use anonymous functions (a.k.a lambdas) to perform
→ binding.
# Their advantage is that you can avoid declaring new functions i.
→ e.
# they offer a concise way to "redirect" callbacks.
btn3 = Button(text="Using anonymous functions.")
btn3.bind(on_press=lambda x: self.on_event(None))

# You can also declare a function that accepts a variable number
→ of
# positional and keyword arguments and use introspection to
→ determine
# what is being passed in. This is very handy for debugging as
→ well
# as function re-use. Here, we use standard event binding to a
→ function
# that accepts optional positional and keyword arguments.
btn4 = Button(text="Use a flexible function")
btn4.bind(on_press=self.on_anything)

# Lastly, we show how to use partial functions. They are sometimes
# difficult to grasp, but provide a very flexible and powerful
→ way to
# reuse functions.
btn5 = Button(text="Using partial functions. For hardcores.")
btn5.bind(on_press=partial(self.on_anything, "1", "2", monthy=
→ "python"))

for but in [btn, btn2, btn3, btn4, btn5]:
    self.add_widget(but)

def on_event(self, obj):
    print("Typical event from", obj)

def on_property(self, obj, value):
    print("Typical property change from", obj, "to", value)

def on_anything(self, *args, **kwargs):
    print('The flexible function has *args of', str(args),
          "and **kwargs of", str(kwargs))

class DemoApp(App):
    def build(self):

```

```

    return DemoBox()

if __name__ == "__main__":
    DemoApp().run()

```

When binding a function to an event or property, a *kivy.weakmethod.WeakMethod* of the callback is saved, and when dispatching the callback is removed if the callback reference becomes invalid.

If a callback has already been bound to a given event or property, it won't be added again.

create_property(*unicode name, value=None, default_value=True, *larges, **kwargs*)

Create a new property at runtime.

New in version 1.0.9.

Changed in version 1.8.0: *value* parameter added, can be used to set the default value of the property. Also, the type of the value is used to specialize the created property.

Changed in version 1.9.0: In the past, if *value* was of type *bool*, a *NumericProperty* would be created, now a *BooleanProperty* is created.

Also, now and positional and keyword arguments are passed to the property when created.

Changed in version 2.0.0: *default_value* has been added.

Warning: This function is designed for the Kivy language, don't use it in your code. You should declare the property in your class instead of using this method.

Parameters

name: *string* Name of the property

value: *object, optional* Default value of the property. Type is also used for creating more appropriate property types. Defaults to *None*.

default_value: *bool, True by default* If *True*, *value* will be the default for the property. Otherwise, the property will be initialized with the the property type's normal default value, and subsequently set to *value*.

```

>>> mywidget = Widget()
>>> mywidget.create_property('custom')
>>> mywidget.custom = True
>>> print(mywidget.custom)
True

```

dispatch(*basestring event_type, *larges, **kwargs*)

Dispatch an event across all the handlers added in *bind*/*fbind*(). As soon as a handler returns *True*, the dispatching stops.

The function collects all the positional and keyword arguments and passes them on to the handlers.

Note: The handlers are called in reverse order than they were registered with *bind*() .

Parameters

event_type: *basestring* the event name to dispatch.

Changed in version 1.9.0: Keyword arguments collection and forwarding was added. Before, only positional arguments would be collected and forwarded.

dispatch_children(*basestring event_type, *larges, **kwargs*)

dispatch_generic(*basestring event_type, *larges, **kwargs*)

events()

Return all the events in the class. Can be used for introspection.

New in version 1.8.0.

fbind(*name, func, *larges, **kwargs*)

A method for advanced, and typically faster binding. This method is different than **bind()** and is meant for more advanced users and internal usage. It can be used as long as the following points are heeded.

1. As opposed to **bind()**, it does not check that this function and larges/kwarg has not been bound before to this name. So binding the same callback multiple times will just keep adding it.
2. Although **bind()** creates a **WeakMethod** of the callback when binding to an event or property, this method stores the callback directly, unless a keyword argument *ref* with value **True** is provided and then a **WeakMethod** is saved. This is useful when there's no risk of a memory leak by storing the callback directly.
3. This method returns a unique positive number if *name* was found and bound, and **0**, otherwise. It does not raise an exception, like **bind()** if the property *name* is not found. If not zero, the uid returned is unique to this *name* and callback and can be used with **unbind_uid()** for unbinding.

When binding a callback with larges and/or kwarg, **funbind()** must be used for unbinding. If no larges and kwarg are provided, **unbind()** may be used as well. **unbind_uid()** can be used in either case.

This method passes on any caught positional and/or keyword arguments to the callback, removing the need to call **partial**. When calling the callback the expended larges are passed on followed by instance/value (just instance for kwarg) followed by expended kwarg.

Following is an example of usage similar to the example in **bind()**:

```
class DemoBox(BoxLayout):

    def __init__(self, **kwargs):
        super(DemoBox, self).__init__(**kwargs)
        self.orientation = "vertical"

        btn = Button(text="Normal binding to event")
        btn.fbind('on_press', self.on_event)

        btn2 = Button(text="Normal binding to a property change")
        btn2.fbind('state', self.on_property)

        btn3 = Button(text="A: Using function with args.")
        btn3.fbind('on_press', self.on_event_with_args, 'right',
                  tree='birch', food='apple')

        btn4 = Button(text="Unbind A.")
        btn4.fbind('on_press', self.unbind_a, btn3)

        btn5 = Button(text="Use a flexible function")
        btn5.fbind('on_press', self.on_anything)
```

(continues on next page)


```

        btn6 = Button(text="B: Using flexible functions with args. For_
→hardcores.")
        btn6.fbind('on_press', self.on_anything, "1", "2", monthy="python
→")

        btn7 = Button(text="Force dispatch B with different params")
        btn7.fbind('on_press', btn6.dispatch, 'on_press', 6, 7, monthy=
→"other python")

        for but in [btn, btn2, btn3, btn4, btn5, btn6, btn7]:
            self.add_widget(but)

    def on_event(self, obj):
        print("Typical event from", obj)

    def on_event_with_args(self, side, obj, tree=None, food=None):
        print("Event with args", obj, side, tree, food)

    def on_property(self, obj, value):
        print("Typical property change from", obj, "to", value)

    def on_anything(self, *args, **kwargs):
        print('The flexible function has *args of', str(args),
              "and **kwargs of", str(kwargs))
        return True

    def unbind_a(self, btn, event):
        btn.funbind('on_press', self.on_event_with_args, 'right',
                    tree='birch', food='apple')

```

Note: Since the kv lang uses this method to bind, one has to implement this method, instead of `bind()` when creating a non *EventDispatcher* based class used with the kv lang. See *Observable* for an example.

New in version 1.9.0.

Changed in version 1.9.1: The *ref* keyword argument has been added.

funbind(*name, func, *largs, **kwargs*)

Similar to *fbind()*.

When unbinding, *unbind()* will unbind all callbacks that match the callback, while this method will only unbind the first.

To unbind, the same positional and keyword arguments passed to *fbind()* must be passed on to funbind.

Note: It is safe to use *funbind()* to unbind a function bound with *bind()* as long as no keyword and positional arguments are provided to *funbind()*.

New in version 1.9.0.

get_property_observers(*name, args=False*)

Returns a list of methods that are bound to the property/event passed as the *name* argument:

```
widget_instance.get_property_observers('on_release')
```

Parameters

name: str The name of the event or property.

args: bool Whether to return the bound args. To keep compatibility, only the callback functions and not their provided args will be returned in the list when *args* is *False*.

If *True*, each element in the list is a 5-tuple of (*callback*, *largs*, *kwargs*, *is_ref*, *uid*), where *is_ref* indicates whether *callback* is a weakref, and *uid* is the uid given by *fbind()*, or *None* if *bind()* was used. Defaults to *False*.

Returns The list of bound callbacks. See *args* for details.

New in version 1.8.0.

Changed in version 1.9.0: *args* has been added.

getter(*name*)

Return the getter of a property.

New in version 1.0.9.

is_event_type(*basestring event_type*)

Return *True* if the *event_type* is already registered.

New in version 1.0.4.

properties() → dict

Return all the properties in the class in a dictionary of key/property class. Can be used for introspection.

New in version 1.0.9.

property(*name*, *quiet=False*)

Get a property instance from the property name. If *quiet* is *True*, *None* is returned instead of raising an exception when *name* is not a property. Defaults to *False*.

New in version 1.0.9.

Returns A *Property* derived instance corresponding to the name.

Changed in version 1.9.0: *quiet* was added.

proxy_ref

Returns a *WeakProxy* reference to the *EventDispatcher*.

New in version 1.9.0.

Changed in version 2.0.0: Previously it just returned itself, now it actually returns a *WeakProxy*.

register_event_type(*basestring event_type*)

Register an event type with the dispatcher.

Registering event types allows the dispatcher to validate event handler names as they are attached and to search attached objects for suitable handlers. Each event type declaration must:

1. start with the prefix *on_*.
2. have a default handler in the class.

Example of creating a custom event:

```

class MyWidget(Widget):
    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        self.register_event_type('on_swipe')

    def on_swipe(self):
        pass

def on_swipe_callback(*largs):
    print('my swipe is called', largs)
w = MyWidget()
w.dispatch('on_swipe')

```

setter(*name*)

Return the setter of a property. Use: `instance.setter('name')`. The setter is a convenient callback function useful if you want to directly bind one property to another. It returns a partial function that will accept (`obj`, `value`) args and results in the property 'name' of instance being set to value.

New in version 1.0.9.

For example, to bind `number2` to `number1` in python you would do:

```

class ExampleWidget(Widget):
    number1 = NumericProperty(None)
    number2 = NumericProperty(None)

    def __init__(self, **kwargs):
        super(ExampleWidget, self).__init__(**kwargs)
        self.bind(number1=self.setter('number2'))

```

This is equivalent to kv binding:

```

<ExampleWidget>:
    number2: self.number1

```

unbind(***kwargs*)

Unbind properties from callback functions with similar usage as `bind()`.

If a callback has been bound to a given event or property multiple times, only the first occurrence will be unbound.

Note: It is safe to use `unbind()` on a function bound with `fbind()` as long as that function was originally bound without any keyword and positional arguments. Otherwise, the function will fail to be unbound and you should use `funbind()` instead.

unbind_uid(*name*, *uid*)

Uses the uid returned by `fbind()` to unbind the callback.

This method is much more efficient than `funbind()`. If `uid` evaluates to False (e.g. 0) a `ValueError` is raised. Also, only callbacks bound with `fbind()` can be unbound with this method.

Since each call to `fbind()` will generate a unique `uid`, only one callback will be removed. If `uid` is not found among the callbacks, no error is raised.

E.g.:

```

btn6 = Button(text="B: Using flexible functions with args. For hardcores.
→")
uid = btn6.fbind('on_press', self.on_anything, "1", "2", monthy="python")
if not uid:
    raise Exception('Binding failed').
...
btn6.unbind_uid('on_press', uid)

```

New in version 1.9.0.

unregister_event_types(*basestring event_type*)

Unregister an event type in the dispatcher.

class kivy.event.**ObjectWithUid**

Bases: `builtins.object`

(internal) This class assists in providing unique identifiers for class instances. It is not intended for direct usage.

class kivy.event.**Observable**

Bases: *kivy.event.ObjectWithUid*

Observable is a stub class defining the methods required for binding. *EventDispatcher* is (the) one example of a class that implements the binding interface. See *EventDispatcher* for details.

New in version 1.9.0.

bind(***kwargs*)

fbind(*name, func, *largs, **kwargs*)

See *EventDispatcher.fbind()*.

Note: To keep backward compatibility with derived classes which may have inherited from *Observable* before, the *fbind()* method was added. The default implementation of *fbind()* is to create a partial function that it passes to *bind* while saving the *uid* and *largs/kwargs*. However, *funbind()* (and *unbind_uid()*) are fairly inefficient since we have to first lookup this partial function using the *largs/kwargs* or *uid* and then call *unbind()* on the returned function. It is recommended to overwrite these methods in derived classes to bind directly for better performance.

Similarly to *EventDispatcher.fbind()*, this method returns 0 on failure and a positive unique *uid* on success. This *uid* can be used with *unbind_uid()*.

funbind(*name, func, *largs, **kwargs*)

See *fbind()* and *EventDispatcher.funbind()*.

unbind(***kwargs*)

unbind_uid(*name, uid*)

See *fbind()* and *EventDispatcher.unbind_uid()*.

20.11 Factory object

The factory can be used to automatically register any class or module and instantiate classes from it anywhere in your project. It is an implementation of the *Factory Pattern*.

The class list and available modules are automatically generated by *setup.py*.

Example for registering a class/module:

```
>>> from kivy.factory import Factory
>>> Factory.register('Widget', module='kivy.uix.widget')
>>> Factory.register('Vector', module='kivy.vector')
```

Example of using the Factory:

```
>>> from kivy.factory import Factory
>>> widget = Factory.Widget(pos=(456,456))
>>> vector = Factory.Vector(9, 2)
```

Example using a class name:

```
>>> from kivy.factory import Factory
>>> Factory.register('MyWidget', cls=MyWidget)
```

By default, the first classname you register via the factory is permanent. If you wish to change the registered class, you need to unregister the classname before you re-assign it:

```
>>> from kivy.factory import Factory
>>> Factory.register('MyWidget', cls=MyWidget)
>>> widget = Factory.MyWidget()
>>> Factory.unregister('MyWidget')
>>> Factory.register('MyWidget', cls=CustomWidget)
>>> customWidget = Factory.MyWidget()
```

kivy.factory.Factory = <kivy.factory.FactoryBase object>

Factory instance to use for getting new classes

exception kivy.factory.FactoryException

Bases: Exception

20.12 Geometry utilities

This module contains some helper functions for geometric calculations.

kivy.geometry.circumcircle(*a, b, c*)

Computes the circumcircle of a triangle defined by *a*, *b*, *c*. See: http://en.wikipedia.org/wiki/Circumscribed_circle

Parameters

- a*: iterable containing at least 2 values (for x and y)** The 1st point of the triangle.
- b*: iterable containing at least 2 values (for x and y)** The 2nd point of the triangle.
- c*: iterable containing at least 2 values (for x and y)** The 3rd point of the triangle.

Return

A tuple that defines the circle :

- The first element in the returned tuple is the center as (x, y)
- The second is the radius (float)

kivy.geometry.minimum_bounding_circle(*points*)

Returns the minimum bounding circle for a set of points.

For a description of the problem being solved, see the [Smallest Circle Problem](#).

The function uses Applet's Algorithm, the runtime is $O(h^3 \cdot n)$, where h is the number of points in the convex hull of the set of points. **But** it runs in linear time in almost all real world cases. See: <http://tinyurl.com/6e4n5yb>

Parameters

points: iterable A list of points (2 tuple with x,y coordinates)

Return

A tuple that defines the circle:

- The first element in the returned tuple is the center (x, y)
- The second the radius (float)

20.13 Gesture recognition

This class allows you to easily create new gestures and compare them:

```
from kivy.gesture import Gesture, GestureDatabase

# Create a gesture
g = Gesture()
g.add_stroke(point_list=[(1,1), (3,4), (2,1)])
g.normalize()

# Add it to the database
gdb = GestureDatabase()
gdb.add_gesture(g)

# And for the next gesture, try to find it!
g2 = Gesture()
# ...
gdb.find(g2)
```

Warning: You don't really want to do this: it's more of an example of how to construct gestures dynamically. Typically, you would need a lot more points, so it's better to record gestures in a file and reload them to compare later. Look in the examples/gestures directory for an example of how to do that.

class kivy.gesture.**Gesture**(tolerance=None)

Bases: builtins.object

A python implementation of a gesture recognition algorithm by Oleg Dopertchouk: <http://www.gamedev.net/reference/articles/article2039.asp>

Implemented by Jeiel Aranal (chemikhazi@gmail.com), released into the public domain.

add_stroke(point_list=None)

Adds a stroke to the gesture and returns the Stroke instance. Optional point_list argument is a list of the mouse points for the stroke.

dot_product(comparison_gesture)

Calculates the dot product of the gesture with another gesture.

get_rigid_rotation(dstpts)

Extract the rotation to apply to a group of points to minimize the distance to a second group

of points. The two groups of points are assumed to be centered. This is a simple version that just picks an angle based on the first point of the gesture.

get_score(*comparison_gesture*, *rotation_invariant=True*)

Returns the matching score of the gesture against another gesture.

normalize(*stroke_samples=32*)

Runs the gesture normalization algorithm and calculates the dot product with self.

class kivy.gesture.**GestureDatabase**

Bases: `builtins.object`

Class to handle a gesture database.

add_gesture(*gesture*)

Add a new gesture to the database.

find(*gesture*, *minscore=0.9*, *rotation_invariant=True*)

Find a matching gesture in the database.

gesture_to_str(*gesture*)

Convert a gesture into a unique string.

str_to_gesture(*data*)

Convert a unique string to a gesture.

class kivy.gesture.**GestureStroke**

Bases: `builtins.object`

Gestures can be made up of multiple strokes.

add_point(*x=x_pos*, *y=y_pos*)

Adds a point to the stroke.

center_stroke(*offset_x*, *offset_y*)

Centers the stroke by offsetting the points.

normalize_stroke(*sample_points=32*)

Normalizes strokes so that every stroke has a standard number of points. Returns True if stroke is normalized, False if it can't be normalized. *sample_points* controls the resolution of the stroke.

points_distance(*point1=GesturePoint*, *point2=GesturePoint*)

Returns the distance between two `GesturePoints`.

scale_stroke(*scale_factor=float*)

Scales the stroke down by *scale_factor*.

stroke_length(*point_list=None*)

Finds the length of the stroke. If a point list is given, finds the length of that list.

20.14 Interactive launcher

New in version 1.3.0.

Deprecated since version 1.10.0: The interactive launcher has been deprecated.

The *InteractiveLauncher* provides a user-friendly python shell interface to an `App` so that it can be prototyped and debugged interactively.

Note: The Kivy API intends for some functions to only be run once or before the main `EventLoop` has started. Methods that can normally be called during the course of an application will work as intended,

but specifically overriding methods such as `on_touch()` dynamically leads to trouble.

20.14.1 Creating an InteractiveLauncher

Take your existing subclass of `App` (this can be production code) and pass an instance to the *InteractiveLauncher* constructor.

```
from kivy.interactive import InteractiveLauncher
from kivy.app import App
from kivy.uix.button import Button

class MyApp(App):
    def build(self):
        return Button(text='Hello Shell')

launcher = InteractiveLauncher(MyApp())
launcher.run()
```

After pressing *enter*, the script will return. This allows the interpreter to continue running. Inspection or modification of the `App` can be done safely through the `InteractiveLauncher` instance or the provided *SafeMembrane* class instances.

Note: If you want to test this example, start Python without any file to have already an interpreter, and copy/paste all the lines. You'll still have the interpreter at the end + the kivy application running.

20.14.2 Interactive Development

IPython provides a fast way to learn the Kivy API. The `App` instance and all of its attributes, including methods and the entire widget tree, can be quickly listed by using the `'.'` operator and pressing `'tab'`. Try this code in an Ipython shell.

```
from kivy.interactive import InteractiveLauncher
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.graphics import Color, Ellipse

class MyPaintWidget(Widget):
    def on_touch_down(self, touch):
        with self.canvas:
            Color(1, 1, 0)
            d = 30.
            Ellipse(pos=(touch.x - d/2, touch.y - d/2), size=(d, d))

class TestApp(App):
    def build(self):
        return Widget()

i = InteractiveLauncher(TestApp())
i.run()
i.      # press 'tab' to list attributes of the app
i.root. # press 'tab' to list attributes of the root widget
```

(continues on next page)


```
# App is boring. Attach a new widget!
i.root.add_widget(MyPaintWidget())

i.safeIn()
# The application is now blocked.
# Click on the screen several times.
i.safeOut()
# The clicks will show up now

# Erase artwork and start over
i.root.canvas.clear()
```

Note: All of the proxies used in the module store their referent in the `_ref` attribute, which can be accessed directly if needed, such as for getting doc strings. `help()` and `type()` will access the proxy, not its referent.

20.14.3 Directly Pausing the Application

Both the *InteractiveLauncher* and *SafeMembrane* hold internal references to the `EventLoop`'s 'safe' and 'confirmed' `threading.Event` objects. You can use their safing methods to control the application manually.

SafeMembrane.safeIn() will cause the application to pause and *SafeMembrane.safeOut()* will allow a paused application to continue running. This is potentially useful for scripting actions into functions that need the screen to update etc.

Note: The pausing is implemented via the *Clocks*' `schedule_once()` method and occurs before the start of each frame.

20.14.4 Adding Attributes Dynamically

Note: This module uses threading and object proxies to encapsulate the running App. Deadlocks and memory corruption can occur if making direct references inside the thread without going through the provided proxy(s).

The *InteractiveLauncher* can have attributes added to it exactly like a normal object and if these were created from outside the membrane, they will not be threadsafe because the external references to them in the python interpreter do not go through *InteractiveLauncher*'s membrane behavior, inherited from *SafeMembrane*.

To threadsafe these external references, simply assign them to *SafeMembrane* instances of themselves like so:

```
from kivy.interactive import SafeMembrane

interactiveLauncher.attribute = myNewObject
# myNewObject is unsafe
myNewObject = SafeMembrane(myNewObject)
# myNewObject is now safe. Call at will.
myNewObject.method()
```

TODO

Unit tests, examples, and a better explanation of which methods are safe in a running application would be nice. All three would be excellent.

Could be re-written with a context-manager style i.e.

```
with safe:
    foo()
```

Any use cases besides compacting code?

class `kivy.interactive.SafeMembrane`(*ob*, **args*, ***kwargs*)

Bases: `builtins.object`

This help is for a proxy object. Did you want help on the proxy's referent instead? Try using `help(<instance>._ref)`

The SafeMembrane is a threadsafe proxy that also returns attributes as new thread-safe objects and makes thread-safe method calls, preventing thread-unsafe objects from leaking into the user's environment.

safeIn()

Provides a thread-safe entry point for interactive launching.

safeOut()

Provides a thread-safe exit point for interactive launching.

class `kivy.interactive.InteractiveLauncher`(*app=None*, **args*, ***kwargs*)

Bases: `kivy.interactive.SafeMembrane`

Proxy to an application instance that launches it in a thread and then returns and acts as a proxy to the application in the thread.

20.15 Kivy Base

This module contains the Kivy core functionality and is not intended for end users. Feel free to look through it, but bare in mind that calling any of these methods directly may result in an unpredictable behavior as the calls access directly the event loop of an application.

`kivy.base.EventLoop` = `<kivy.base.EventLoopBase object>`

EventLoop instance

class `kivy.base.EventLoopBase`

Bases: `kivy.event.EventDispatcher`

Main event loop. This loop handles the updating of input and dispatching events.

add_event_listener(*listener*)

Add a new event listener for getting touch events.

add_input_provider(*provider*, *auto_remove=False*)

Add a new input provider to listen for touch events.

add_postproc_module(*mod*)

Add a postproc input module (DoubleTap, TripleTap, DeJitter RetainTouch are defaults).

async_idle()

Identical to `idle()`, but instead used when running within an async event loop.

close()

Exit from the main loop and stop all configured input providers.

dispatch_input()

Called by *EventLoopBase.idle()* to read events from input providers, pass events to postproc, and dispatch final events.

ensure_window()

Ensure that we have a window.

exit()

Close the main loop and close the window.

idle()

This function is called after every frame. By default:

- it “ticks” the clock to the next frame.
- it reads all input and dispatches events.
- it dispatches *on_update*, *on_draw* and *on_flip* events to the window.

on_pause()

Event handler for *on_pause* which will be fired when the event loop is paused.

on_start()

Event handler for *on_start* which will be fired right after all input providers have been started.

on_stop()

Event handler for *on_stop* events which will be fired right after all input providers have been stopped.

post_dispatch_input(etype, me)

This function is called by *EventLoopBase.dispatch_input()* when we want to dispatch an input event. The event is dispatched to all listeners and if grabbed, it’s dispatched to grabbed widgets.

remove_android_splash(*args)

Remove android presplash in SDL2 bootstrap.

remove_event_listener(listener)

Remove an event listener from the list.

remove_input_provider(provider)

Remove an input provider.

remove_postproc_module(mod)

Remove a postproc module.

run()

Main loop

set_window(window)

Set the window used for the event loop.

start()

Must be called only once before *EventLoopBase.run()*. This starts all configured input providers.

stop()

Stop all input providers and call callbacks registered using *EventLoop.add_stop_callback()*.

touches

Return the list of all touches currently in down or move states.

class kivy.base.ExceptionHandler

Bases: *builtins.object*

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())
```

All exceptions will be set to PASS, and logged to the console!

handle_exception(exception)

Handle one exception, defaults to returning `ExceptionManager.RAISE`.

class kivy.base.ExceptionManagerBase

Bases: `builtins.object`

ExceptionManager manages exceptions handlers.

add_handler(cls)

Add a new exception handler to the stack.

handle_exception(inst)

Called when an exception occurred in the `runTouchApp()` main loop.

remove_handler(cls)

Remove a exception handler from the stack.

kivy.base.ExceptionManager = <kivy.base.ExceptionManagerBase object>

Instance of a `ExceptionManagerBase` implementation.

kivy.base.runTouchApp(widget=None, slave=False)

Static main function that starts the application loop. You can access some magic via the following arguments:

See `kivy.app` for example usage.

Parameters

<empty> To make dispatching work, you need at least one input listener. If not, application will leave. (MTWindow act as an input listener)

widget If you pass only a widget, a MTWindow will be created and your widget will be added to the window as the root widget.

slave No event dispatching is done. This will be your job.

widget + slave No event dispatching is done. This will be your job but we try to get the window (must be created by you beforehand) and add the widget to it. Very useful for embedding Kivy in another toolkit. (like Qt, check kivy-designed)

kivy.base.async_runTouchApp(widget=None, slave=False, async_lib=None)

Identical to `runTouchApp()` but instead it is a coroutine that can be run in an existing async event loop.

`async_lib` is the async library to use. See `kivy.app` for details and example usage.

New in version 2.0.0.

kivy.base.stopTouchApp()

Stop the current application by leaving the main loop.

See `kivy.app` for example usage.

20.16 Logger object

Different logging levels are available : trace, debug, info, warning, error and critical.

Examples of usage:

```
from kivy.logger import Logger

Logger.info('title: This is a info message.')
Logger.debug('title: This is a debug message.')

try:
    raise Exception('bleh')
except Exception:
    Logger.exception('Something happened!')
```

The message passed to the logger is split into two parts, separated by a colon (:). The first part is used as a title, and the second part is used as the message. This way, you can “categorize” your message easily.

```
Logger.info('Application: This is a test')

# will appear as

[INFO    ] [Application ] This is a test
```

20.16.1 Logger configuration

The Logger can be controlled via the Kivy configuration file:

```
[kivy]
log_level = info
log_enable = 1
log_dir = logs
log_name = kivy_%y-%m-%d_%.txt
log_maxfiles = 100
```

More information about the allowed values are described in the [kivy.config](#) module.

20.16.2 Logger history

Even if the logger is not enabled, you still have access to the last 100 messages:

```
from kivy.logger import LoggerHistory

print(LoggerHistory.history)
```

kivy.logger.Logger = <Logger kivy (WARNING)>
Kivy default logger instance

class kivy.logger.LoggerHistory(level=0)
Bases: logging.Handler
Kivy history handler

emit(message)

Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplemented-Error.

flush()

Ensure all logging output has been flushed.

This version does nothing and is intended to be implemented by subclasses.

20.17 Metrics

New in version 1.5.0.

A screen is defined by its physical size, density and resolution. These factors are essential for creating UI's with correct size everywhere.

In Kivy, all the graphics pipelines work with pixels. But using pixels as a measurement unit is problematic because sizes change according to the screen.

20.17.1 Dimensions

If you want to design your UI for different screen sizes, you will want better measurement units to work with. Kivy provides some more scalable alternatives.

Units

pt Points - 1/72 of an inch based on the physical size of the screen. Prefer to use **sp** instead of **pt**.

mm Millimeters - Based on the physical size of the screen.

cm Centimeters - Based on the physical size of the screen.

in Inches - Based on the physical size of the screen.

dp Density-independent Pixels - An abstract unit that is based on the physical density of the screen. With a **density** of 1, 1dp is equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up a factor appropriate to the screen's dpi, and the inverse for a lower dpi. The ratio of dp-to-pixels will change with the screen density, but not necessarily in direct proportion. Using the dp unit is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In others words, it provides consistency for the real-world size of your UI across different devices.

sp Scale-independent Pixels - This is like the dp unit, but it is also scaled by the user's font size preference. We recommend you use this unit when specifying font sizes, so the font size will be adjusted to both the screen density and the user's preference.

20.17.2 Examples

Here is an example of creating a label with a **sp** **font_size** and setting the height manually with a 10dp margin:

```
#:kivy 1.5.0
<MyWidget>:
    Label:
        text: 'Hello world'
        font_size: '15sp'
        size_hint_y: None
        height: self.texture_size[1] + dp(10)
```

20.17.3 Manual control of metrics

The metrics cannot be changed at runtime. Once a value has been converted to pixels, you can't retrieve the original value anymore. This stems from the fact that the DPI and density of a device cannot be changed at runtime.

We provide some environment variables to control metrics:

- `KIVY_METRICS_DENSITY`: if set, this value will be used for *density* instead of the systems one. On android, the value varies between 0.75, 1, 1.5 and 2.
- `KIVY_METRICS_FONTSIZE`: if set, this value will be used for *fontscale* instead of the systems one. On android, the value varies between 0.8 and 1.2.
- `KIVY_DPI`: if set, this value will be used for *dpi*. Please note that setting the DPI will not impact the dp/sp notation because these are based on the screen density.

For example, if you want to simulate a high-density screen (like the HTC One X):

```
KIVY_DPI=320 KIVY_METRICS_DENSITY=2 python main.py --size 1280x720
```

Or a medium-density (like Motorola Droid 2):

```
KIVY_DPI=240 KIVY_METRICS_DENSITY=1.5 python main.py --size 854x480
```

You can also simulate an alternative user preference for fontscale as follows:

```
KIVY_METRICS_FONTSIZE=1.2 python main.py
```

`kivy.metrics.Metrics = <kivy.metrics.MetricsBase object>`

Default instance of *MetricsBase*, used everywhere in the code .. versionadded:: 1.7.0

class `kivy.metrics.MetricsBase`

Bases: `builtins.object`

Class that contains the default attributes for Metrics. Don't use this class directly, but use the *Metrics* instance.

density()

Return the density of the screen. This value is 1 by default on desktops but varies on android depending on the screen.

dpi()

Return the DPI of the screen. Depending on the platform, the DPI can be taken from the Window provider (Desktop mainly) or from a platform-specific module (like android/ios).

dpi_rounded()

Return the DPI of the screen, rounded to the nearest of 120, 160, 240 or 320.

fontscale()

Return the fontscale user preference. This value is 1 by default but can vary between 0.8 and 1.2.

`kivy.metrics.pt(value)`

Convert from points to pixels

`kivy.metrics.inch(value)`

Convert from inches to pixels

`kivy.metrics.cm(value)`

Convert from centimeters to pixels

`kivy.metrics.mm(value)`

Convert from millimeters to pixels

`kivy.metrics.dp(value)`
Convert from density-independent pixels to pixels

`kivy.metrics.sp(value)`
Convert from scale-independent pixels to pixels

20.18 Multistroke gesture recognizer

New in version 1.9.0.

Warning: This is experimental and subject to change as long as this warning notice is present.

See `kivy/examples/demo/multistroke/main.py` for a complete application example.

20.18.1 Conceptual Overview

This module implements the Protractor gesture recognition algorithm.

Recognizer is the search/database API similar to *GestureDatabase*. It maintains a list of *MultistrokeGesture* objects and allows you to search for a user-input gestures among them.

ProgressTracker tracks the progress of a *Recognizer.recognize()* call. It can be used to interact with the running recognizer task, for example forcing it to stop half-way, or analyzing results as they arrive.

MultistrokeGesture represents a gesture in the gesture database (`Recognizer.db`). It is a container for *UnistrokeTemplate* objects, and implements the heap permute algorithm to automatically generate all possible stroke orders (if desired).

UnistrokeTemplate represents a single stroke path. It's typically instantiated automatically by *MultistrokeGesture*, but sometimes you may need to create them manually.

Candidate represents a user-input gesture that is used to search the gesture database for matches. It is normally instantiated automatically by calling *Recognizer.recognize()*.

20.18.2 Usage examples

See `kivy/examples/demo/multistroke/main.py` for a complete application example.

You can bind to events on *Recognizer* to track the state of all calls to *Recognizer.recognize()*. The callback function will receive an instance of *ProgressTracker* that can be used to analyze and control various aspects of the recognition process

```
from kivy.vector import Vector
from kivy.multistroke import Recognizer

gdb = Recognizer()

def search_start(gdb, pt):
    print("A search is starting with %d tasks" % (pt.tasks))

def search_stop(gdb, pt):
    # This will call max() on the result dictionary, so it's best to store
    # it instead of calling it 3 times consecutively
    best = pt.best
```

(continues on next page)

(continued from previous page)

```
print("Search ended (%s). Best is %s (score %f, distance %f)" % (
    pt.status, best['name'], best['score'], best['dist'] ))

# Bind your callbacks to track all matching operations
gdb.bind(on_search_start=search_start)
gdb.bind(on_search_complete=search_stop)

# The format below is referred to as `strokes`, a list of stroke paths.
# Note that each path shown here consists of two points, ie a straight
# line; if you plot them it looks like a T, hence the name.
gdb.add_gesture('T', [
    [Vector(30, 7), Vector(103, 7)],
    [Vector(66, 7), Vector(66, 87)]]

# Now you can search for the 'T' gesture using similar data (user input).
# This will trigger both of the callbacks bound above.
gdb.recognize([
    [Vector(45, 8), Vector(110, 12)],
    [Vector(88, 9), Vector(85, 95)]])
```

On the next *Clock* tick, the matching process starts (and, in this case, completes).

To track individual calls to *Recognizer.recognize()*, use the return value (also a *ProgressTracker* instance)

```
# Same as above, but keep track of progress using returned value
progress = gdb.recognize([
    [Vector(45, 8), Vector(110, 12)],
    [Vector(88, 9), Vector(85, 95)]]

progress.bind(on_progress=my_other_callback)
print(progress.progress) # = 0

# [ assuming a kivy.clock.Clock.tick() here ]

print(result.progress) # = 1
```

20.18.3 Algorithm details

For more information about the matching algorithm, see:

“Protractor: A fast and accurate gesture recognizer” by Yang Li <http://yangl.org/pdf/protractor-chi2010.pdf>

“\$N-Protractor” by Lisa Anthony and Jacob O. Wobbrock <http://depts.washington.edu/aimgroup/proj/dollar/ndollar-protractor.pdf>

Some of the code is derived from the JavaScript implementation here: <http://depts.washington.edu/aimgroup/proj/dollar/ndollar.html>

class kivy.multistroke.Recognizer(**kwargs)
Bases: *kivy.event.EventDispatcher*

Recognizer provides a gesture database with matching facilities.

Events

on_search_start Fired when a new search is started using this Recognizer.

on_search_complete Fired when a running search ends, for whatever reason. (use `ProgressTracker.status` to find out)

Properties

db A `ListProperty` that contains the available *MultistrokeGesture* objects.

db is a *ListProperty* and defaults to []

add_gesture(*name, strokes, **kwargs*)

Add a new gesture to the database. This will instantiate a new *MultistrokeGesture* with *strokes* and append it to `self.db`.

Note: If you already have instantiated a *MultistrokeGesture* object and wish to add it, append it to `Recognizer.db` manually.

export_gesture(*filename=None, **kwargs*)

Export a list of *MultistrokeGesture* objects. Outputs a base64-encoded string that can be decoded to a Python list with the *parse_gesture()* function or imported directly to `self.db` using *Recognizer.import_gesture()*. If *filename* is specified, the output is written to disk, otherwise returned.

This method accepts optional *Recognizer.filter()* arguments.

filter(***kwargs*)

filter() returns a subset of objects in `self.db`, according to given criteria. This is used by many other methods of the *Recognizer*; the arguments below can for example be used when calling *Recognizer.recognize()* or *Recognizer.export_gesture()*. You normally don't need to call this directly.

Arguments

name Limits the returned list to gestures where `MultistrokeGesture.name` matches given regular expression(s). If `re.match(name, MultistrokeGesture.name)` tests true, the gesture is included in the returned list. Can be a string or an array of strings

```
gdb = Recognizer()

# Will match all names that start with a capital N
# (ie Next, New, N, Nebraska etc, but not "n" or "next")
gdb.filter(name='N')

# exactly 'N'
gdb.filter(name='N$')

# Nebraska, teletubbies, France, fraggle, N, n, etc
gdb.filter(name=['[Nn]', '(?i)T', '(?i)F'])
```

priority Limits the returned list to gestures with certain `MultistrokeGesture.priority` values. If specified as an integer, only gestures with a lower priority are returned. If specified as a list (min/max)

```
# Max priority 50
gdb.filter(priority=50)

# Max priority 50 (same result as above)
gdb.filter(priority=[0, 50])
```

(continues on next page)

```
# Min priority 50, max 100
gdb.filter(priority=[50, 100])
```

When this option is used, `Recognizer.db` is automatically sorted according to priority, incurring extra cost. You can use `force_priority_sort` to override this behavior if your gestures are already sorted according to priority.

orientation_sensitive Limits the returned list to gestures that are orientation sensitive (True), gestures that are not orientation sensitive (False) or None (ignore template sensitivity, this is the default).

numstrokes Limits the returned list to gestures that have the specified number of strokes (in `MultistrokeGesture.strokes`). Can be a single integer or a list of integers.

numpoints Limits the returned list to gestures that have specific `MultistrokeGesture.numpoints` values. This is provided for flexibility, do not use it unless you understand what it does. Can be a single integer or a list of integers.

force_priority_sort Can be used to override the default sort behavior. Normally `MultistrokeGesture` objects are returned in priority order if the `priority` option is used. Setting this to True will return gestures sorted in priority order, False will return in the order gestures were added. None means decide automatically (the default).

Note: For improved performance, you can load your gesture database in priority order and set this to False when calling `Recognizer.recognize()`

db Can be set if you want to filter a different list of objects than `Recognizer.db`. You probably don't want to do this; it is used internally by `import_gesture()`.

import_gesture(*data=None, filename=None, **kwargs*)

Import a list of gestures as formatted by `export_gesture()`. One of *data* or *filename* must be specified.

This method accepts optional `Recognizer.filter()` arguments, if none are specified then all gestures in specified data are imported.

parse_gesture(*data*)

Parse data formatted by `export_gesture()`. Returns a list of `MultistrokeGesture` objects. This is used internally by `import_gesture()`, you normally don't need to call this directly.

prepare_templates(***kwargs*)

This method is used to prepare `UnistrokeTemplate` objects within the gestures in `self.db`. This is useful if you want to minimize punishment of lazy resampling by preparing all vectors in advance. If you do this before a call to `Recognizer.export_gesture()`, you will have the vectors computed when you load the data later.

This method accepts optional `Recognizer.filter()` arguments.

force_numpoints, if specified, will prepare all templates to the given number of points (instead of each template's preferred *n*; ie `UnistrokeTemplate.numpoints`). You normally don't want to do this.

recognize(*strokes, goodscore=None, timeout=0, delay=0, **kwargs*)

Search for gestures matching *strokes*. Returns a `ProgressTracker` instance.

This method accepts optional `Recognizer.filter()` arguments.

Arguments

strokes A list of stroke paths (list of lists of *Vector* objects) that will be matched against gestures in the database. Can also be a *Candidate* instance.

Warning: If you manually supply a *Candidate* that has a skip-flag, make sure that the correct filter arguments are set. Otherwise the system will attempt to load vectors that have not been computed. For example, if you set *skip_bounded* and do not set *orientation_sensitive* to False, it will raise an exception if an orientation_sensitive *UnistrokeTemplate* is encountered.

goodscore If this is set (between 0.0 - 1.0) and a gesture score is equal to or higher than the specified value, the search is immediately halted and the *on_search_complete* event is fired (+ the *on_complete* event of the associated *ProgressTracker* instance). Default is None (disabled).

timeout Specifies a timeout (in seconds) for when the search is aborted and the results returned. This option applies only when *max_gpf* is not 0. Default value is 0, meaning all gestures in the database will be tested, no matter how long it takes.

max_gpf Specifies the maximum number of *MultistrokeGesture* objects that can be processed per frame. When exceeded, will cause the search to halt and resume work in the next frame. Setting to 0 will complete the search immediately (and block the UI).

Warning: This does not limit the number of *UnistrokeTemplate* objects matched! If a single gesture has a million templates, they will all be processed in a single frame with *max_gpf*=1!

delay Sets an optional delay between each run of the recognizer loop. Normally, a run is scheduled for the next frame until the tasklist is exhausted. If you set this, there will be an additional delay between each run (specified in seconds). Default is 0, resume in the next frame.

force_numpoints forces all templates (and candidate) to be prepared to a certain number of points. This can be useful for example if you are evaluating templates for optimal n (do not use this unless you understand what it does).

transfer_gesture(*tgt*, ***kwargs*)

Transfers *MultistrokeGesture* objects from *Recognizer.db* to another *Recognizer* instance *tgt*.

This method accepts optional *Recognizer.filter()* arguments.

class *kivy.multistroke.ProgressTracker*(*candidate*, *tasks*, ***kwargs*)

Bases: *kivy.event.EventDispatcher*

Represents an ongoing (or completed) search operation. Instantiated and returned by the *Recognizer.recognize()* method when it is called. The *results* attribute is a dictionary that is updated as the recognition operation progresses.

Note: You do not need to instantiate this class.

Arguments

candidate *Candidate* object to be evaluated

tasks Total number of gestures in tasklist (to test against)

Events

on_progress Fired for every gesture that is processed

on_result Fired when a new result is added, and it is the first match for the *name* so far, or a consecutive match with better score.

on_complete Fired when the search is completed, for whatever reason. (use *ProgressTracker.status* to find out)

Attributes

results A dictionary of all results (so far). The key is the name of the gesture (ie *UnistrokeTemplate.name* usually inherited from *MultistrokeGesture*). Each item in the dictionary is a dict with the following entries:

name Name of the matched template (redundant)

score Computed score from 1.0 (perfect match) to 0.0

dist Cosine distance from candidate to template (low=closer)

gesture The *MultistrokeGesture* object that was matched

best_template Index of the best matching template (in *MultistrokeGesture.templates*)

template_results List of distances for all templates. The list index corresponds to a *UnistrokeTemplate* index in *gesture.templates*.

status

search Currently working

stop Was stopped by the user (*stop()* called)

timeout A timeout occurred (specified as *timeout=* to *recognize()*)

goodscore The search was stopped early because a gesture with a high enough score was found (specified as *goodscore=* to *recognize()*)

complete The search is complete (all gestures matching filters were tested)

best

Return the best match found by *recognize()* so far. It returns a dictionary with three keys, 'name', 'dist' and 'score' representing the template's name, distance (from candidate path) and the computed score value. This is a Python property.

progress

Returns the progress as a float, 0 is 0% done, 1 is 100%. This is a Python property.

stop()

Raises a stop flag that is checked by the search process. It will be stopped on the next clock tick (if it is still running).

class *kivy.multistroke.MultistrokeGesture*(*name, strokes=None, **kwargs*)

Bases: *builtins.object*

MultistrokeGesture represents a gesture. It maintains a set of *strokes* and generates unistroke (ie *UnistrokeTemplate*) permutations that are used for evaluating candidates against this gesture later.

Arguments

name Identifies the name of the gesture - it is returned to you in the results of a `Recognizer.recognize()` search. You can have any number of MultistrokeGesture objects with the same name; many definitions of one gesture. The same name is given to all the generated unistroke permutations. Required, no default.

strokes A list of paths that represents the gesture. A path is a list of Vector objects:

```
gesture = MultistrokeGesture('my_gesture', strokes=[
    [Vector(x1, y1), Vector(x2, y2), ..... ], # stroke 1
    [Vector(), Vector(), Vector(), Vector() ] # stroke 2
    #, [stroke 3], [stroke 4], ...
])
```

For template matching purposes, all the strokes are combined to a single list (unistroke). You should still specify the strokes individually, and set `stroke_sensitive` True (whenever possible).

Once you do this, unistroke permutations are immediately generated and stored in `self.templates` for later, unless you set the `permute` flag to False.

priority Determines when `Recognizer.recognize()` will attempt to match this template, lower priorities are evaluated first (only if a `priority_filter` is used). You should use lower priority on gestures that are more likely to match. For example, set user templates at lower number than generic templates. Default is 100.

numpoints Determines the number of points this gesture should be resampled to (for matching purposes). The default is 16.

stroke_sensitive Determines if the number of strokes (paths) in this gesture is required to be the same in the candidate (user input) gesture during matching. If this is False, candidates will always be evaluated, disregarding the number of strokes. Default is True.

orientation_sensitive Determines if this gesture is orientation sensitive. If True, aligns the indicative orientation with the one of eight base orientations that requires least rotation. Default is True.

angle_similarity This is used by the `Recognizer.recognize()` function when a candidate is evaluated against this gesture. If the angles between them are too far off, the template is considered a non-match. Default is 30.0 (degrees)

permute If False, do not use Heap Permute algorithm to generate different stroke orders when instantiated. If you set this to False, a single UnistrokeTemplate built from `strokes` is used.

add_stroke(*stroke*, *permute=False*)

Add a stroke to the `self.strokes` list. If `permute` is True, the `permute()` method is called to generate new unistroke templates

get_distance(*cand*, *tpl*, *numpoints=None*)

Compute the distance from this Candidate to a UnistrokeTemplate. Returns the Cosine distance between the stroke paths.

`numpoints` will prepare both the UnistrokeTemplate and Candidate path to `n` points (when necessary), you probably don't want to do this.

match_candidate(*cand*, ***kwargs*)

Match a given candidate against this MultistrokeGesture object. Will test against all templates and report results as a list of four items:

index 0 Best matching template's index (in `self.templates`)

index 1 Computed distance from the template to the candidate path

index 2 List of distances for all templates. The list index corresponds to a *UnistrokeTemplate* index in `self.templates`.

index 3 Counter for the number of performed matching operations, ie templates matched against the candidate

permute()

Generate all possible unistroke permutations from `self.strokes` and save the resulting list of *UnistrokeTemplate* objects in `self.templates`.

Quote from <http://faculty.washington.edu/wobbrock/pubs/gi-10.2.pdf>

We use Heap Permute [16] (p. 179) to generate all stroke orders in a multistroke gesture. Then, to generate stroke directions for each order, we treat each component stroke as a dichotomous [0,1] variable. There are 2^N combinations for N strokes, so we convert the decimal values 0 to 2^N-1 , inclusive, to binary representations and regard each bit as indicating forward (0) or reverse (1). This algorithm is often used to generate truth tables in propositional logic.

See section 4.1: “ 2^N Algorithm” of the linked paper for details.

Warning: Using heap permute for gestures with more than 3 strokes can result in very large number of templates (a 9-stroke gesture = 38 million templates). If you are dealing with these types of gestures, you should manually compose all the desired stroke orders.

class `kivy.multistroke.UnistrokeTemplate`(*name*, *points*=None, ***kwargs*)

Bases: `builtins.object`

Represents a (uni)stroke path as a list of Vectors. Normally, this class is instantiated by *Multi-strokeGesture* and not by the programmer directly. However, it is possible to manually compose *UnistrokeTemplate* objects.

Arguments

name Identifies the name of the gesture. This is normally inherited from the parent *MultistrokeGesture* object when a template is generated.

points A list of points that represents a unistroke path. This is normally one of the possible stroke order permutations from a *MultistrokeGesture*.

numpoints The number of points this template should (ideally) be resampled to before the matching process. The default is 16, but you can use a template-specific settings if that improves results.

orientation_sensitive Determines if this template is orientation sensitive (True) or fully rotation invariant (False). The default is True.

Note: You will get an exception if you set a skip-flag and then attempt to retrieve those vectors.

add_point(p)

Add a point to the unistroke/path. This invalidates all previously computed vectors.

prepare(numpoints=None)

This function prepares the *UnistrokeTemplate* for matching given a target number of points (for resample). 16 is optimal.

class kivy.multistroke.Candidate(*strokes=None, numpoints=16, **kwargs*)

Bases: builtins.object

Represents a set of unistroke paths of user input, ie data to be matched against a *UnistrokeTemplate* object using the Protractor algorithm. By default, data is precomputed to match both rotation bounded and fully invariant *UnistrokeTemplate* objects.

Arguments

strokes See *MultistrokeGesture.strokes* for format example. The Candidate strokes are simply combined to a unistroke in the order given. The idea is that this will match one of the unistroke permutations in *MultistrokeGesture.templates*.

numpoints The Candidate's default N; this is only for a fallback, it is not normally used since n is driven by the *UnistrokeTemplate* we are being compared to.

skip_bounded If True, do not generate/store rotation bounded vectors

skip_invariant If True, do not generate/store rotation invariant vectors

Note that you WILL get errors if you set a skip-flag and then attempt to retrieve the data.

add_stroke(*stroke*)

Add a stroke to the candidate; this will invalidate all previously computed vectors

get_angle_similarity(*tpl, **kwargs*)

(Internal use only) Compute the angle similarity between this Candidate and a *UnistrokeTemplate* object. Returns a number that represents the angle similarity (lower is more similar).

get_protractor_vector(*numpoints, orientation_sens*)

(Internal use only) Return vector for comparing to a *UnistrokeTemplate* with Protractor

get_start_unit_vector(*numpoints, orientation_sens*)

(Internal use only) Get the start vector for this Candidate, with the path resampled to *numpoints* points. This is the first step in the matching process. It is compared to a *UnistrokeTemplate* object's start vector to determine angle similarity.

prepare(*numpoints=None*)

Prepare the Candidate vectors. *self.strokes* is combined to a single unistroke (connected end-to-end), resampled to *numpoints* points, and then the vectors are calculated and stored in *self.db* (for use by *get_distance* and *get_angle_similarity*)

20.19 Parser utilities

Helper functions used for CSS parsing.

kivy.parser.parse_color(*text*)

Parse a string to a kivy color. Supported formats:

- *rgb(r, g, b)*
- *rgba(r, g, b, a)*
- *rgb*
- *rgba*
- *rrggbb*
- *rrggbbaa*

For hexadecimal values, you can also use:

- #rgb
- #rgba
- #rrgbb
- #rrggbaa

`kivy.parser.parse_int`
alias of `builtins.int`

`kivy.parser.parse_float`
alias of `builtins.float`

`kivy.parser.parse_string(text)`
Parse a string to a string (removing single and double quotes).

`kivy.parser.parse_bool(text)`
Parse a string to a boolean, ignoring case. “true”/“1” is True, “false”/“0” is False. Anything else throws an exception.

`kivy.parser.parse_int2(text)`
Parse a string to a list of exactly 2 integers.

```
>>> print(parse_int2("12 54"))
12, 54
```

`kivy.parser.parse_float4(text)`
Parse a string to a list of exactly 4 floats.

```
>>> parse_float4('54 87. 35 0')
54, 87., 35, 0
```

`kivy.parser.parse_filename(filename)`
Parse a filename and search for it using `resource_find()`. If found, the resource path is returned, otherwise return the unmodified filename (as specified by the caller).

20.20 Properties

The *Properties* classes are used when you create an *EventDispatcher*.

Warning: Kivy’s Properties are **not to be confused** with Python’s properties (i.e. the `@property` decorator and the `<property>` type).

Kivy’s property classes support:

Value Checking / Validation When you assign a new value to a property, the value is checked against validation constraints. For example, validation for an *OptionProperty* will make sure that the value is in a predefined list of possibilities. Validation for a *NumericProperty* will check that your value is a numeric type. This prevents many errors early on.

Observer Pattern You can specify what should happen when a property’s value changes. You can bind your own function as a callback to changes of a *Property*. If, for example, you want a piece of code to be called when a widget’s *pos* property changes, you can *bind* a function to it.

Better Memory Management The same instance of a property is shared across multiple widget instances.

20.20.1 Comparison Python vs. Kivy

Basic example

Let's compare Python and Kivy properties by creating a Python class with 'a' as a float property:

```
class MyClass(object):
    def __init__(self, a=1.0):
        super(MyClass, self).__init__()
        self.a = a
```

With Kivy, you can do:

```
class MyClass(EventDispatcher):
    a = NumericProperty(1.0)
```

Depth being tracked

Only the "top level" of a nested object is being tracked. For example:

```
my_list_prop = ListProperty([1, {'hi': 0}])
# Changing a top level element will trigger all `on_my_list_prop` callbacks
my_list_prop[0] = 4
# Changing a deeper element will be ignored by all `on_my_list_prop` callbacks
my_list_prop[1]['hi'] = 4
```

The same holds true for all container-type kivy properties.

Value checking

If you wanted to add a check for a minimum / maximum value allowed for a property, here is a possible implementation in Python:

```
class MyClass(object):
    def __init__(self, a=1):
        super(MyClass, self).__init__()
        self.a_min = 0
        self.a_max = 100
        self.a = a

    def _get_a(self):
        return self._a
    def _set_a(self, value):
        if value < self.a_min or value > self.a_max:
            raise ValueError('a out of bounds')
        self._a = value
    a = property(_get_a, _set_a)
```

The disadvantage is you have to do that work yourself. And it becomes laborious and complex if you have many properties. With Kivy, you can simplify the process:

```
class MyClass(EventDispatcher):
    a = BoundedNumericProperty(1, min=0, max=100)
```

That's all!

Error Handling

If setting a value would otherwise raise a `ValueError`, you have two options to handle the error gracefully within the property. The first option is to use an `errorvalue` parameter. An `errorvalue` is a substitute for the invalid value:

```
# simply returns 0 if the value exceeds the bounds
bnp = BoundedNumericProperty(0, min=-500, max=500, errorvalue=0)
```

The second option is to use an `errorhandler` parameter. An `errorhandler` is a callable (single argument function or lambda) which can return a valid substitute:

```
# returns the boundary value when exceeded
bnp = BoundedNumericProperty(0, min=-500, max=500,
    errorhandler=lambda x: 500 if x > 500 else -500)
```

Keyword arguments and `__init__()`

When working with inheritance, namely with the `__init__()` of an object that inherits from `EventDispatcher` e.g. a `Widget`, the properties protect you from a Python 3 object error. This error occurs when passing kwargs to the `object` instance through a `super()` call:

```
class MyClass(EventDispatcher):
    def __init__(self, **kwargs):
        super(MyClass, self).__init__(**kwargs)
        self.my_string = kwargs.get('my_string')

print(MyClass(my_string='value').my_string)
```

While this error is silenced in Python 2, it will stop the application in Python 3 with:

```
TypeError: object.__init__() takes no parameters
```

Logically, to fix that you'd either put `my_string` directly in the `__init__()` definition as a required argument or as an optional keyword argument with a default value i.e.:

```
class MyClass(EventDispatcher):
    def __init__(self, my_string, **kwargs):
        super(MyClass, self).__init__(**kwargs)
        self.my_string = my_string
```

or:

```
class MyClass(EventDispatcher):
    def __init__(self, my_string='default', **kwargs):
        super(MyClass, self).__init__(**kwargs)
        self.my_string = my_string
```

Alternatively, you could pop the key-value pair from the `kwargs` dictionary before calling `super()`:

```
class MyClass(EventDispatcher):
    def __init__(self, **kwargs):
        self.my_string = kwargs.pop('my_string')
        super(MyClass, self).__init__(**kwargs)
```

Kivy properties are more flexible and do the required `kwargs.pop()` in the background automatically (within the `super()` call to `EventDispatcher`) to prevent this distraction:

```

class MyClass(EventDispatcher):
    my_string = StringProperty('default')
    def __init__(self, **kwargs):
        super(MyClass, self).__init__(**kwargs)

print(MyClass(my_string='value').my_string)

```

Conclusion

Kivy properties are easier to use than the standard ones. See the next chapter for examples of how to use them :)

20.20.2 Observe Property changes

As we said in the beginning, Kivy's Properties implement the **Observer pattern**. That means you can **bind()** to a property and have your own function called when the value changes.

There are multiple ways to observe the changes.

Observe using bind()

You can observe a property change by using the bind() method outside of the class:

```

class MyClass(EventDispatcher):
    a = NumericProperty(1)

def callback(instance, value):
    print('My callback is call from', instance)
    print('and the a value changed to', value)

ins = MyClass()
ins.bind(a=callback)

# At this point, any change to the a property will call your callback.
ins.a = 5    # callback called
ins.a = 5    # callback not called, because the value did not change
ins.a = -1   # callback called

```

Note: Property objects live at the class level and manage the values attached to instances. Re-assigning at class level will remove the Property. For example, continuing with the code above, `MyClass.a = 5` replaces the property object with a simple int.

Observe using 'on_<propname>'

If you defined the class yourself, you can use the 'on_<propname>' callback:

```

class MyClass(EventDispatcher):
    a = NumericProperty(1)

    def on_a(self, instance, value):
        print('My property a changed to', value)

```

Warning: Be careful with 'on_<propname>'. If you are creating such a callback on a property you are inheriting, you must not forget to call the superclass function too.

20.20.3 Binding to properties of properties.

When binding to a property of a property, for example binding to a numeric property of an object saved in a object property, updating the object property to point to a new object will not re-bind the numeric property to the new object. For example:

```
<MyWidget>:
    Label:
        id: first
        text: 'First label'
    Label:
        id: second
        text: 'Second label'
    Button:
        label: first
        text: self.label.text
        on_press: self.label = second
```

When clicking on the button, although the label object property has changed to the second widget, the button text will not change because it is bound to the text property of the first label directly.

In 1.9.0, the `rebind` option has been introduced that will allow the automatic updating of the `text` when `label` is changed, provided it was enabled. See [ObjectProperty](#).

class `kivy.properties.Property`(*defaultvalue*, ***kw*)

Bases: `builtins.object`

This class handles all the basic setters and getters, None type handling, the observer list and storage initialisation. This class should not be directly instantiated.

By default, a *Property* always takes a default value:

```
class MyObject(Widget):
    hello = Property('Hello world')
```

The default value must be a value that agrees with the Property type. For example, you can't set a list to a *StringProperty* because the *StringProperty* will check the default value.

None is a special case: you can set the default value of a Property to None, but you can't set None to a property afterward. If you really want to do that, you must declare the Property with *allownone=True*:

```
class MyObject(Widget):
    hello = ObjectProperty(None, allownone=True)

# then later
a = MyObject()
a.hello = 'bleh' # working
a.hello = None # working too, because allownone is True.
```

Parameters

default: Specifies the default value for the property.

****kwargs:** If the parameters include *errorhandler*, this should be a callable which must take a single argument and return a valid substitute value.

If the parameters include *errorvalue*, this should be an object. If set, it will replace an invalid property value (overrides *errorhandler*).

If the parameters include *force_dispatch*, it should be a boolean. If True, no value comparison will be done, so the property event will be dispatched even if the new value matches the old value (by default identical values are not dispatched to avoid infinite recursion in two-way binds). Be careful, this is for advanced use only.

comparator: callable or None When not None, it's called with two values to be compared. The function returns whether they are considered the same.

deprecated: bool When True, a warning will be logged if the property is accessed or set. Defaults to False.

Changed in version 1.4.2: Parameters *errorhandler* and *errorvalue* added

Changed in version 1.9.0: Parameter *force_dispatch* added

Changed in version 1.11.0: Parameter *deprecated* added

bind(*EventDispatcher obj, observer*)

Add a new observer to be called only when the value is changed.

defaultvalue

defaultvalue: object

dispatch(*EventDispatcher obj*)

Dispatch the value change to all observers.

Changed in version 1.1.0: The method is now accessible from Python.

This can be used to force the dispatch of the property, even if the value didn't change:

```
button = Button()
# get the Property class instance
prop = button.property('text')
# dispatch this property on the button instance
prop.dispatch(button)
```

Property.fbind(EventDispatcher obj, observer, int ref, tuple largs=

Similar to *bind*, except it doesn't check if the observer already exists. It also expands and forwards *largs* and *kwargs* to the callback. *funbind* or *unbind_uid* should be called when unbinding. It returns a unique positive uid to be used with *unbind_uid*.

Property.funbind(EventDispatcher obj, observer, tuple largs=

Remove the observer from our widget observer list bound with *fbind*. It removes the first match it finds, as opposed to *unbind* which searches for all matches.

get(*EventDispatcher obj*)

Return the value of the property.

link(*EventDispatcher obj, unicode name*)

Link the instance with its real name.

Warning: Internal usage only.

When a widget is defined and uses a *Property* class, the creation of the property object happens, but the instance doesn't know anything about its name in the widget class:

```
class MyWidget(Widget):
    uid = NumericProperty(0)
```

In this example, the uid will be a NumericProperty() instance, but the property instance doesn't know its name. That's why `link()` is used in `Widget.__new__`. The link function is also used to create the storage space of the property for this specific widget instance.

link_deps(*EventDispatcher obj, unicode name*)

set(*EventDispatcher obj, value*)

Set a new value for the property.

unbind(*EventDispatcher obj, observer*)

Remove the observer from our widget observer list.

unbind_uid(*EventDispatcher obj, uid*)

Remove the observer from our widget observer list bound with fbind using the uid.

class kivy.properties.NumericProperty(*defaultvalue=0, **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: int or float, defaults to 0 Specifies the default value of the property.

```
>>> wid = Widget()
>>> wid.x = 42
>>> print(wid.x)
42
>>> wid.x = "plop"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "properties.pyx", line 93, in kivy.properties.Property.__set__
  File "properties.pyx", line 111, in kivy.properties.Property.set
  File "properties.pyx", line 159, in kivy.properties.NumericProperty.check
ValueError: NumericProperty accept only int/float
```

Changed in version 1.4.1: NumericProperty can now accept custom text and tuple value to indicate a type, like "in", "pt", "px", "cm", "mm", in the format: '10pt' or (10, 'pt').

get_format(*EventDispatcher obj*)

Return the format used for Numeric calculation. Default is px (mean the value have not been changed at all). Otherwise, it can be one of 'in', 'pt', 'cm', 'mm'.

class kivy.properties.StringProperty(*defaultvalue=u'', **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: string, defaults to "" Specifies the default value of the property.

class kivy.properties.ListProperty(*defaultvalue=0, **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: list, defaults to [] Specifies the default value of the property.

Warning: When assigning a list to a `ListProperty`, the list stored in the property is a shallow copy of the list and not the original list. This can be demonstrated with the following example:

```

>>> class MyWidget(Widget):
>>>     my_list = ListProperty([])

>>> widget = MyWidget()
>>> my_list = [1, 5, {'hi': 'hello'}]
>>> widget.my_list = my_list
>>> print(my_list is widget.my_list)
False
>>> my_list.append(10)
>>> print(my_list, widget.my_list)
[1, 5, {'hi': 'hello'}, 10] [1, 5, {'hi': 'hello'}]

```

However, changes to nested levels will affect the property as well, since the property uses a shallow copy of `my_list`.

```

>>> my_list[2]['hi'] = 'bye'
>>> print(my_list, widget.my_list)
[1, 5, {'hi': 'bye'}, 10] [1, 5, {'hi': 'bye'}]

```

link(*EventDispatcher obj, unicode name*)

set(*EventDispatcher obj, value*)

class `kivy.properties.ObjectProperty`(*defaultvalue=None, rebind=False, **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: **object type** Specifies the default value of the property.

rebind: **bool, defaults to False** Whether kv rules using this object as an intermediate attribute in a kv rule, will update the bound property when this object changes.

That is the standard behavior is that if there's a kv rule `text: self.a.b.c.d`, where `a`, `b`, and `c` are properties with `rebind False` and `d` is a `StringProperty`. Then when the rule is applied, `text` becomes bound only to `d`. If `a`, `b`, or `c` change, `text` still remains bound to `d`. Furthermore, if any of them were `None` when the rule was initially evaluated, e.g. `b` was `None`; then `text` is bound to `b` and will not become bound to `d` even when `b` is changed to not be `None`.

By setting `rebind` to `True`, however, the rule will be re-evaluated and all the properties rebound when that intermediate property changes. E.g. in the example above, whenever `b` changes or becomes not `None` if it was `None` before, `text` is evaluated again and becomes rebound to `d`. The overall result is that `text` is now bound to all the properties among `a`, `b`, or `c` that have `rebind` set to `True`.

****kwargs:** **a list of keyword arguments**

baseclass If `kwargs` includes a `baseclass` argument, this value will be used for validation: `isinstance(value, kwargs['baseclass'])`.

Warning: To mark the property as changed, you must reassign a new python object.

Changed in version 1.9.0: `rebind` has been introduced.

Changed in version 1.7.0: `baseclass` parameter added.

rebind

rebind: 'int'

class `kivy.properties.BooleanProperty`(*defaultvalue=True, **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: boolean Specifies the default value of the property.

class `kivy.properties.BoundedNumericProperty`(**largs, **kw*)

Bases: `kivy.properties.Property`

maximum bound – within a numeric range.

Parameters

default: numeric Specifies the default value of the property.

****kwargs: a list of keyword arguments** If a *min* parameter is included, this specifies the minimum numeric value that will be accepted. If a *max* parameter is included, this specifies the maximum numeric value that will be accepted.

bounds

Return min/max of the value.

New in version 1.0.9.

get_max(*EventDispatcher obj*)

Return the maximum value acceptable for the `BoundedNumericProperty` in *obj*. Return `None` if no maximum value is set. Check `get_min` for a usage example.

New in version 1.1.0.

get_min(*EventDispatcher obj*)

Return the minimum value acceptable for the `BoundedNumericProperty` in *obj*. Return `None` if no minimum value is set:

```
class MyWidget(Widget):
    number = BoundedNumericProperty(0, min=-5, max=5)

widget = MyWidget()
print(widget.property('number').get_min(widget))
# will output -5
```

New in version 1.1.0.

set_max(*EventDispatcher obj, value*)

Change the maximum value acceptable for the `BoundedNumericProperty`, only for the *obj* instance. Set to `None` if you want to disable it. Check `set_min` for a usage example.

Warning: Changing the bounds doesn't revalidate the current value.

New in version 1.1.0.

set_min(*EventDispatcher obj, value*)

Change the minimum value acceptable for the `BoundedNumericProperty`, only for the *obj* instance. Set to `None` if you want to disable it:

```
class MyWidget(Widget):
    number = BoundedNumericProperty(0, min=-5, max=5)
```

(continues on next page)

(continued from previous page)

```
widget = MyWidget()
# change the minimum to -10
widget.property('number').set_min(widget, -10)
# or disable the minimum check
widget.property('number').set_min(widget, None)
```

Warning: Changing the bounds doesn't revalidate the current value.

New in version 1.1.0.

class kivy.properties.**OptionProperty**(*largs, **kw)

Bases: *kivy.properties.Property*

options.

If the string set in the property is not in the list of valid options (passed at property creation time), a `ValueError` exception will be raised.

Parameters

default: any valid type in the list of options Specifies the default value of the property.

****kwargs:** a list of keyword arguments Should include an *options* parameter specifying a list (not tuple) of valid options.

For example:

```
class MyWidget(Widget):
    state = OptionProperty("None", options=["On", "Off", "None"])
```

options

Return the options available.

New in version 1.0.9.

class kivy.properties.**ReferenceListProperty**(*largs, **kw)

Bases: *kivy.properties.Property*

For example, if *x* and *y* are *NumericProperty*s, we can create a *ReferenceListProperty* for the *pos*. If you change the value of *pos*, it will automatically change the values of *x* and *y* accordingly. If you read the value of *pos*, it will return a tuple with the values of *x* and *y*.

For example:

```
class MyWidget(EventDispatcher):
    x = NumericProperty(0)
    y = NumericProperty(0)
    pos = ReferenceListProperty(x, y)
```

get(*EventDispatcher obj*)

link(*EventDispatcher obj, unicode name*)

link_deps(*EventDispatcher obj, unicode name*)

set(*EventDispatcher obj, _value*)

setitem(*EventDispatcher obj, key, value*)

trigger_change(*EventDispatcher obj, value*)

class kivy.properties.**AliasProperty**(*getter*, *setter=None*, *rebind=False*, ***kwargs*)

Bases: [kivy.properties.Property](#)

If you don't find a Property class that fits to your needs, you can make your own by creating custom Python getter and setter methods.

Example from `kivy/uix/widget.py` where *x* and *width* are instances of [NumericProperty](#):

```
def get_right(self):
    return self.x + self.width
def set_right(self, value):
    self.x = value - self.width
right = AliasProperty(get_right, set_right, bind=['x', 'width'])
```

If *x* were a non Kivy property then you have to return *True* from setter to dispatch new value of *right*:

```
def set_right(self, value):
    self.x = value - self.width
    return True
```

Usually *bind* list should contain all Kivy properties used in getter method. If you return *True* it will cause a dispatch which one should do when the property value has changed, but keep in mind that the property could already have dispatched the changed value if a kivy property the alias property is bound was set in the setter, causing a second dispatch if the setter returns *True*.

If you want to cache the value returned by getter then pass *cache=True*. This way getter will only be called if new value is set or one of the binded properties changes. In both cases new value of alias property will be cached again.

To make property readonly pass *None* as setter. This way *AttributeError* will be raised on every set attempt:

```
right = AliasProperty(get_right, None, bind=['x', 'width'], cache=True)
```

Parameters

getter: function Function to use as a property getter.

setter: function Function to use as a property setter. Callbacks bound to the alias property won't be called when the property is set (e.g. *right = 10*), unless the setter returns *True*.

bind: list/tuple Properties to observe for changes as property name strings. Changing values of this properties will dispatch value of the alias property.

cache: boolean If *True*, the value will be cached until one of the binded elements changes or if setter returns *True*.

rebind: bool, defaults to False See [ObjectProperty](#) for details.

Changed in version 1.9.0: *rebind* has been introduced.

Changed in version 1.4.0: Parameter *cache* added.

dispatch(*EventDispatcher obj*)

get(*EventDispatcher obj*)

link_deps(*EventDispatcher obj, unicode name*)

rebind

rebind: 'int'

set(*EventDispatcher obj, value*)

trigger_change(*EventDispatcher obj, value*)

class `kivy.properties.DictProperty`(*defaultvalue=0, rebind=False, **kw*)

Bases: `kivy.properties.Property`

Parameters

defaultvalue: dict, defaults to {} Specifies the default value of the property.

rebind: bool, defaults to False See `ObjectProperty` for details.

Changed in version 1.9.0: *rebind* has been introduced.

Warning: Similar to `ListProperty`, when assigning a dict to a `DictProperty`, the dict stored in the property is a shallow copy of the dict and not the original dict. See `ListProperty` for details.

link(*EventDispatcher obj, unicode name*)

rebind

rebind: 'int'

set(*EventDispatcher obj, value*)

class `kivy.properties.VariableListProperty`(*defaultvalue=None, length=4, **kw*)

Bases: `kivy.properties.Property`

list items and to expand them to the desired list size.

For example, `GridLayout`'s padding used to just accept one numeric value which was applied equally to the left, top, right and bottom of the `GridLayout`. Now padding can be given one, two or four values, which are expanded into a length four list [left, top, right, bottom] and stored in the property.

Parameters

default: a default list of values Specifies the default values for the list.

length: int, one of 2 or 4. Specifies the length of the final list. The *default* list will be expanded to match a list of this length.

****kwargs:** a list of keyword arguments Not currently used.

Keeping in mind that the *default* list is expanded to a list of length 4, here are some examples of how `VariableListProperty`'s are handled.

- `VariableListProperty([1])` represents [1, 1, 1, 1].
- `VariableListProperty([1, 2])` represents [1, 2, 1, 2].
- `VariableListProperty(['1px', (2, 'px'), 3, 4.0])` represents [1, 2, 3, 4.0].
- `VariableListProperty(5)` represents [5, 5, 5, 5].
- `VariableListProperty(3, length=2)` represents [3, 3].

New in version 1.7.0.

length

length: 'int'

link(*EventDispatcher obj, unicode name*)

```
class kivy.properties.ConfigParserProperty(defaultvalue, section, key, config, **kw)
    Bases: kivy.properties.Property
```

of a *ConfigParser* as well as to bind the ConfigParser values to other properties.

A ConfigParser is composed of sections, where each section has a number of keys and values associated with these keys. ConfigParserProperty lets you automatically listen to and change the values of specified keys based on other kivy properties.

For example, say we want to have a TextInput automatically write its value, represented as an int, in the *info* section of a ConfigParser. Also, the textinputs should update its values from the ConfigParser's fields. Finally, their values should be displayed in a label. In py:

```
class Info(Label):

    number = ConfigParserProperty(0, 'info', 'number', 'example',
                                  val_type=int, errorvalue=41)

    def __init__(self, **kw):
        super(Info, self).__init__(**kw)
        config = ConfigParser(name='example')
```

The above code creates a property that is connected to the *number* key in the *info* section of the ConfigParser named *example*. Initially, this ConfigParser doesn't exist. Then, in *__init__*, a ConfigParser is created with name *example*, which is then automatically linked with this property. then in kv:

```
BoxLayout:
    TextInput:
        id: number
        text: str(info.number)
    Info:
        id: info
        number: number.text
        text: 'Number: {}'.format(self.number)
```

You'll notice that we have to do *text: str(info.number)*, this is because the value of this property is always an int, because we specified *int* as the *val_type*. However, we can assign anything to the property, e.g. *number: number.text* which assigns a string, because it is instantly converted with the *val_type* callback.

Note: If a file has been opened for this ConfigParser using *read()*, then *write()* will be called every property change, keeping the file updated.

Warning: It is recommend that the config parser object be assigned to the property after the kv tree has been constructed (e.g. schedule on next frame from init). This is because the kv tree and its properties, when constructed, are evaluated on its own order, therefore, any initial values in the parser might be overwritten by objects it's bound to. So in the example above, the TextInput might be initially empty, and if *number: number.text* is evaluated before *text: str(info.number)*, the config value will be overwritten with the (empty) text value.

Parameters

default: object type Specifies the default value for the key. If the parser associated with this property doesn't have this section or key, it'll be created with the current value, which is the default value initially.

section: string type The section in the ConfigParser where the key / value will be written. Must be provided. If the section doesn't exist, it'll be created.

key: string type The key in section *section* where the value will be written to. Must be provided. If the key doesn't exist, it'll be created and the current value written to it, otherwise its value will be used.

config: string or *ConfigParser* instance. The ConfigParser instance to associate with this property if not None. If it's a string, the ConfigParser instance whose *name* is the value of *config* will be used. If no such parser exists yet, whenever a ConfigParser with this name is created, it will automatically be linked to this property.

Whenever a ConfigParser becomes linked with a property, if the section or key doesn't exist, the current property value will be used to create that key, otherwise, the existing key value will be used for the property value; overwriting its current value. You can change the ConfigParser associated with this property if a string was used here, by changing the *name* of an existing or new ConfigParser instance. Or through *set_config()*.

****kwargs: a list of keyword arguments**

val_type: a callable object The key values are saved in the ConfigParser as strings. When the ConfigParser value is read internally and assigned to the property or when the user changes the property value directly, if *val_type* is not None, it will be called with the new value as input and it should return the value converted to the proper type accepted by this property. For example, if the property represents ints, *val_type* can simply be *int*.

If the *val_type* callback raises a *ValueError*, *errorvalue* or *errorhandler* will be used if provided. Tip: the *getboolean* function of the ConfigParser might also be useful here to convert to a boolean type.

verify: a callable object Can be used to restrict the allowable values of the property. For every value assigned to the property, if this is specified, *verify* is called with the new value, and if it returns *True* the value is accepted, otherwise, *errorvalue* or *errorhandler* will be used if provided or a *ValueError* is raised.

New in version 1.9.0.

link_deps (*EventDispatcher obj*, *unicode name*)

set (*EventDispatcher obj*, *value*)

set_config (*config*)

Sets the ConfigParser object to be used by this property. Normally, the ConfigParser is set when initializing the Property using the *config* parameter.

Parameters

config: A *ConfigParser* instance. The instance to use for listening to and saving property value changes. If None, it disconnects the currently used *ConfigParser*.

```
class MyWidget(Widget):
    username = ConfigParserProperty('', 'info', 'name', None)

widget = MyWidget()
widget.property('username').set_config(ConfigParser())
```

20.21 Resources management

Resource management can be a pain if you have multiple paths and projects. Kivy offers 2 functions for searching for specific resources across a list of paths.

20.21.1 Resource lookup

When Kivy looks for a resource e.g. an image or a kv file, it searches through a predetermined set of folders. You can modify this folder list using the `resource_add_path()` and `resource_remove_path()` functions.

20.21.2 Customizing Kivy

These functions can also be helpful if you want to replace standard Kivy resources with your own. For example, if you wish to customize or re-style Kivy, you can force your `style.kv` or `data/defaulttheme-0.png` files to be used in preference to the defaults simply by adding the path to your preferred alternatives via the `resource_add_path()` method.

As almost all Kivy resources are looked up using the `resource_find()`, so you can use this approach to add fonts and keyboard layouts and to replace images and icons.

`kivy.resources.resource_find(filename)`

Search for a resource in the list of paths. Use `resource_add_path` to add a custom path to the search.

`kivy.resources.resource_add_path(path)`

Add a custom path to search in.

`kivy.resources.resource_remove_path(path)`

Remove a search path.

New in version 1.0.8.

20.22 Support

Activate other frameworks/toolkits inside the kivy event loop.

`kivy.support.install_gobject_iteration()`

Import and install gobject context iteration inside our event loop. This is used as soon as gobject is used (like gstreamer).

`kivy.support.install_twisted_reactor(**kwargs)`

Installs a threaded twisted reactor, which will schedule one reactor iteration before the next frame only when twisted needs to do some work.

Any arguments or keyword arguments passed to this function will be passed on the the threaded-select reactors `interleave` function. These are the arguments one would usually pass to twisted's `reactor.startRunning`.

Unlike the default twisted reactor, the installed reactor will not handle any signals unless you set the `'installSignalHandlers'` keyword argument to 1 explicitly. This is done to allow kivy to handle the signals as usual unless you specifically want the twisted reactor to handle the signals (e.g. `SIGINT`).

Note: Twisted is not included in iOS build by default. To use it on iOS, put the twisted distribution (and zope.interface dependency) in your application directory.

kivy.support.uninstall_twisted_reactor()

Uninstalls the Kivy's threaded Twisted Reactor. No more Twisted tasks will run after this got called. Use this to clean the *twisted.internet.reactor*.

New in version 1.9.0.

kivy.support.install_android()

Install hooks for the android platform.

- Automatically sleep when the device is paused.
- Automatically kill the application when the return key is pressed.

20.23 Utils

The Utils module provides a selection of general utility functions and classes that may be useful for various applications. These include maths, color, algebraic and platform functions.

Changed in version 1.6.0: The OrderedDict class has been removed. Use collections.OrderedDict instead.

kivy.utils.intersection(set1, set2)

Return the intersection of 2 lists.

kivy.utils.difference(set1, set2)

Return the difference between 2 lists.

kivy.utils.strtotuple(s)

Convert a tuple string into a tuple with some security checks. Designed to be used with the eval() function:

```
a = (12, 54, 68)
b = str(a)          # return '(12, 54, 68)'
c = strtotuple(b)   # return (12, 54, 68)
```

kivy.utils.get_color_from_hex(s)

Transform a hex string color to a kivy *Color*.

kivy.utils.get_hex_from_color(color)

Transform a kivy *Color* to a hex value:

```
>>> get_hex_from_color((0, 1, 0))
'#00ff00'
>>> get_hex_from_color((.25, .77, .90, .5))
'#3fc4e57f'
```

New in version 1.5.0.

kivy.utils.get_random_color(alpha=1.0)

Returns a random color (4 tuple).

Parameters

alpha: float, defaults to 1.0 If alpha == 'random', a random alpha value is generated.

kivy.utils.is_color_transparent(c)

Return True if the alpha channel is 0.

kivy.utils.boundary(value, minvalue, maxvalue)

Limit a value between a minvalue and maxvalue.

kivy.utils.deprecated(func=None, msg="")

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted the first time the function is used.

class kivy.utils.SafeList

Bases: `builtins.list`

List with a `clear()` method.

Warning: Usage of the `iterate()` function will decrease your performance.

clear()

Remove all items from list.

kivy.utils.interpolate(value_from, value_to, step=10)

Interpolate between two values. This can be useful for smoothing some transitions. For example:

```
# instead of setting directly
self.pos = pos

# use interpolate, and you'll have a nicer transition
self.pos = interpolate(self.pos, new_pos)
```

Warning: These interpolations work only on lists/tuples/doubles with the same dimensions. No test is done to check the dimensions are the same.

class kivy.utils.QueryDict

Bases: `builtins.dict`

QueryDict is a `dict()` that can be queried with `dot`.

```
d = QueryDict()
# create a key named toto, with the value 1
d.toto = 1
# it's the same as
d['toto'] = 1
```

New in version 1.0.4.

kivy.utils.platform = 'linux'

A string identifying the current operating system. It is one of: `'win'`, `'linux'`, `'android'`, `'macosx'`, `'ios'` or `'unknown'`. You can use it as follows:

```
from kivy.utils import platform
if platform == 'linux':
    do_linux_things()
```

New in version 1.3.0.

Changed in version 1.8.0: `platform` is now a variable instead of a function.

kivy.utils.escape_markup(text)

Escape markup characters found in the text. Intended to be used when markup text is activated on the Label:

```
untrusted_text = escape_markup('Look at the example [1]')
text = '[color=ff0000]' + untrusted_text + '[/color]'
w = Label(text=text, markup=True)
```

New in version 1.3.0.

class kivy.utils.reify(func)

Bases: builtins.object

Put the result of a method which uses this (non-data) descriptor decorator in the instance dict after the first call, effectively replacing the decorator with an instance variable.

It acts like @property, except that the function is only ever called once; after that, the value is cached as a regular attribute. This gives you lazy attribute creation on objects that are meant to be immutable.

Taken from the [Pyramid project](#).

To use this as a decorator:

```
@reify
def lazy(self):
    ...
    return hard_to_compute_int
first_time = self.lazy # lazy is reify obj, reify.__get__() runs
second_time = self.lazy # lazy is hard_to_compute_int
```

kivy.utils.rgba(s, *args)

Return a Kivy color (4 value from 0-1 range) from either a hex string or a list of 0-255 values.

New in version 1.10.0.

20.24 Vector

The **Vector** represents a 2D vector (x, y). Our implementation is built on top of a Python list.

An example of constructing a Vector:

```
>>> # Construct a point at 82,34
>>> v = Vector(82, 34)
>>> v[0]
82
>>> v.x
82
>>> v[1]
34
>>> v.y
34

>>> # Construct by giving a list of 2 values
>>> pos = (93, 45)
>>> v = Vector(pos)
>>> v[0]
93
>>> v.x
93
```

(continues on next page)

(continued from previous page)

```
>>> v[1]
45
>>> v.y
45
```

20.24.1 Optimized usage

Most of the time, you can use a list for arguments instead of using a Vector. For example, if you want to calculate the distance between 2 points:

```
a = (10, 10)
b = (87, 34)

# optimized method
print('distance between a and b:', Vector(a).distance(b))

# non-optimized method
va = Vector(a)
vb = Vector(b)
print('distance between a and b:', va.distance(vb))
```

20.24.2 Vector operators

The *Vector* supports some numeric operators such as +, -, /:

```
>>> Vector(1, 1) + Vector(9, 5)
[10, 6]

>>> Vector(9, 5) - Vector(5, 5)
[4, 0]

>>> Vector(10, 10) / Vector(2., 4.)
[5.0, 2.5]

>>> Vector(10, 10) / 5.
[2.0, 2.0]
```

You can also use in-place operators:

```
>>> v = Vector(1, 1)
>>> v += 2
>>> v
[3, 3]
>>> v *= 5
[15, 15]
>>> v /= 2.
[7.5, 7.5]
```

class kivy.vector.**Vector**(*args)

Bases: builtins.list

Vector class. See module documentation for more information.

angle(a)

Computes the angle between a and b, and returns the angle in degrees.

```
>>> Vector(100, 0).angle((0, 100))
-90.0
>>> Vector(87, 23).angle((-77, 10))
-157.7920283010705
```

distance(*to*)

Returns the distance between two points.

```
>>> Vector(10, 10).distance((5, 10))
5.
>>> a = (90, 33)
>>> b = (76, 34)
>>> Vector(a).distance(b)
14.035668847618199
```

distance2(*to*)

Returns the distance between two points squared.

```
>>> Vector(10, 10).distance2((5, 10))
25
```

dot(*a*)

Computes the dot product of *a* and *b*.

```
>>> Vector(2, 4).dot((2, 2))
12
```

static in_bbox(*point*, *a*, *b*)

Return True if *point* is in the bounding box defined by *a* and *b*.

```
>>> bmin = (0, 0)
>>> bmax = (100, 100)
>>> Vector.in_bbox((50, 50), bmin, bmax)
True
>>> Vector.in_bbox((647, -10), bmin, bmax)
False
```

length()

Returns the length of a vector.

```
>>> Vector(10, 10).length()
14.142135623730951
>>> pos = (10, 10)
>>> Vector(pos).length()
14.142135623730951
```

length2()

Returns the length of a vector squared.

```
>>> Vector(10, 10).length2()
200
>>> pos = (10, 10)
>>> Vector(pos).length2()
200
```

static line_intersection(*v1*, *v2*, *v3*, *v4*)

Finds the intersection point between the lines (1)*v1*->*v2* and (2)*v3*->*v4* and returns it as a vector object.

```
>>> a = (98, 28)
>>> b = (72, 33)
>>> c = (10, -5)
>>> d = (20, 88)
>>> Vector.line_intersection(a, b, c, d)
[15.25931928687196, 43.911669367909241]
```

Warning: This is a line intersection method, not a segment intersection.

For math see: http://en.wikipedia.org/wiki/Line-line_intersection

normalize()

Returns a new vector that has the same direction as vec, but has a length of one.

```
>>> v = Vector(88, 33).normalize()
>>> v
[0.93632917756904444, 0.3511234415883917]
>>> v.length()
1.0
```

rotate(*angle*)

Rotate the vector with an angle in degrees.

```
>>> v = Vector(100, 0)
>>> v.rotate(45)
[70.71067811865476, 70.71067811865474]
```

static segment_intersection(*v1, v2, v3, v4*)

Finds the intersection point between segments (1)v1->v2 and (2)v3->v4 and returns it as a vector object.

```
>>> a = (98, 28)
>>> b = (72, 33)
>>> c = (10, -5)
>>> d = (20, 88)
>>> Vector.segment_intersection(a, b, c, d)
None
```

```
>>> a = (0, 0)
>>> b = (10, 10)
>>> c = (0, 10)
>>> d = (10, 0)
>>> Vector.segment_intersection(a, b, c, d)
[5, 5]
```

x

x represents the first element in the list.

```
>>> v = Vector(12, 23)
>>> v[0]
12
>>> v.x
12
```

y

y represents the second element in the list.

```
>>> v = Vector(12, 23)
>>> v[1]
23
>>> v.y
23
```

20.25 Weak Method

The *WeakMethod* is used by the *Clock* class to allow references to a bound method that permits the associated object to be garbage collected. Please refer to *examples/core/clock_method.py* for more information.

This WeakMethod class is taken from the recipe <http://code.activestate.com/recipes/81253/>, based on the nicodemus version. Many thanks nicodemus!

```
class kivy.weakmethod.WeakMethod(method)
```

Bases: `builtins.object`

Implementation of a *weakref* for functions and bound methods.

```
is_dead()
```

Returns True if the referenced callable was a bound method and the instance no longer exists. Otherwise, return False.

20.26 Weak Proxy

In order to allow garbage collection, the weak proxy provides *weak references* to objects. It effectively enhances the *weakref.proxy* by adding comparison support.

```
class kivy.weakproxy.WeakProxy(obj, destructor=None)
```

Bases: `builtins.object`

Replacement for *weakref.proxy* to support comparisons

Core Abstraction

This module defines the abstraction layers for our core providers and their implementations. For further information, please refer to *Architectural Overview* and the *Core Providers and Input Providers* section of the documentation.

In most cases, you shouldn't directly use a library that's already covered by the core abstraction. Always try to use our providers first. In case we are missing a feature or method, please let us know by opening a new Bug report instead of relying on your library.

Warning: These are **not** widgets! These are just abstractions of the respective functionality. For example, you cannot add a core image to your window. You have to use the image **widget** class instead. If you're really looking for widgets, please refer to *kivy.uix* instead.

exception kivy.core.CoreCriticalException

Bases: Exception

21.1 Audio

Load an audio sound and play it with:

```
from kivy.core.audio import SoundLoader

sound = SoundLoader.load('mytest.wav')
if sound:
    print("Sound found at %s" % sound.source)
    print("Sound is %.3f seconds" % sound.length)
    sound.play()
```

You should not use the Sound class directly. The class returned by *SoundLoader.load()* will be the best sound provider for that particular file type, so it might return different Sound classes depending the file type.

21.1.1 Event dispatching and state changes

Audio is often processed in parallel to your code. This means you often need to enter the Kivy *eventloop* in order to allow events and state changes to be dispatched correctly.

You seldom need to worry about this as Kivy apps typically always require this event loop for the GUI to remain responsive, but it is good to keep this in mind when debugging or running in a *REPL* (Read-eval-print loop).

Changed in version 1.10.0: The pygst and gi providers have been removed.

Changed in version 1.8.0: There are now 2 distinct Gstreamer implementations: one using Gi/Gst working for both Python 2+3 with Gstreamer 1.0, and one using PyGST working only for Python 2 + Gstreamer 0.10.

Note: The core audio library does not support recording audio. If you require this functionality, please refer to the *audiostream* extension.

class kivy.core.audio.Sound

Bases: *kivy.event.EventDispatcher*

Represents a sound to play. This class is abstract, and cannot be used directly.

Use SoundLoader to load a sound.

Events

on_play: **None** Fired when the sound is played.

on_stop: **None** Fired when the sound is stopped.

filename

Deprecated since version 1.3.0: Use *source* instead.

get_pos()

Returns the current position of the audio file. Returns 0 if not playing.

New in version 1.4.1.

length

Get length of the sound (in seconds).

load()

Load the file into memory.

loop

Set to True if the sound should automatically loop when it finishes.

New in version 1.8.0.

loop is a *BooleanProperty* and defaults to False.

pitch

Pitch of a sound. 2 is an octave higher, .5 one below. This is only implemented for SDL2 audio provider yet.

New in version 1.10.0.

pitch is a *NumericProperty* and defaults to 1.

play()

Play the file.

seek(position)

Go to the <position> (in seconds).

Note: Most sound providers cannot seek when the audio is stopped. Play then seek.

source

Filename / source of your audio file.

New in version 1.3.0.

source is a *StringProperty* that defaults to None and is read-only. Use the *SoundLoader.load()* for loading audio.

state

State of the sound, one of 'stop' or 'play'.

New in version 1.3.0.

state is a read-only *OptionProperty*.

status

Deprecated since version 1.3.0: Use *state* instead.

stop()

Stop playback.

unload()

Unload the file from memory.

volume

Volume, in the range 0-1. 1 means full volume, 0 means mute.

New in version 1.3.0.

volume is a *NumericProperty* and defaults to 1.

class kivy.core.audio.**SoundLoader**

Bases: *builtins.object*

Load a sound, using the best loader for the given file type.

static load(*filename*)

Load a sound, and return a *Sound()* instance.

static register(*classobj*)

Register a new class to load the sound.

21.2 Camera

Core class for acquiring the camera and converting its input into a *Texture*.

Changed in version 1.10.0: The *pygst* and *videocapture* providers have been removed.

Changed in version 1.8.0: There is now 2 distinct *Gstreamer* implementation: one using *Gi/Gst* working for both Python 2+3 with *Gstreamer* 1.0, and one using *PyGST* working only for Python 2 + *Gstreamer* 0.10.

class kivy.core.camera.**CameraBase**(***kwargs*)

Bases: *kivy.event.EventDispatcher*

Abstract Camera Widget class.

Concrete camera classes must implement initialization and frame capturing to a buffer that can be uploaded to the gpu.

Parameters

index: int Source index of the camera.

size: tuple (int, int) Size at which the image is drawn. If no size is specified, it defaults to the resolution of the camera image.

resolution: tuple (int, int) Resolution to try to request from the camera. Used in the gstreamer pipeline by forcing the appsink caps to this resolution. If the camera doesn't support the resolution, a negotiation error might be thrown.

Events

on_load Fired when the camera is loaded and the texture has become available.

on_texture Fired each time the camera texture is updated.

index

Source index of the camera

init_camera()

Initialise the camera (internal)

resolution

Resolution of camera capture (width, height)

start()

Start the camera acquire

stop()

Release the camera

texture

Return the camera texture with the latest capture

21.3 Clipboard

Core class for accessing the Clipboard. If we are not able to access the system clipboard, a fake one will be used.

Usage example:

```
#:import Clipboard kivy.core.clipboard.Clipboard

Button:
    on_release:
        self.text = Clipboard.paste()
        Clipboard.copy('Data')
```

21.4 OpenGL

Select and use the best OpenGL library available. Depending on your system, the core provider can select an OpenGL ES or a 'classic' desktop OpenGL library.

21.5 Image

Core classes for loading images and converting them to a *Texture*. The raw image data can be kept in memory for further access.

Changed in version 1.11.0: Add support for argb and abgr image data

21.5.1 In-memory image loading

New in version 1.9.0: Official support for in-memory loading. Not all the providers support it, but currently SDL2, pygame, pil and imageio work.

To load an image with a filename, you would usually do:

```
from kivy.core.image import Image as CoreImage
im = CoreImage("image.png")
```

You can also load the image data directly from a memory block. Instead of passing the filename, you'll need to pass the data as a BytesIO object together with an "ext" parameter. Both are mandatory:

```
import io
from kivy.core.image import Image as CoreImage
data = io.BytesIO(open("image.png", "rb").read())
im = CoreImage(data, ext="png")
```

By default, the image will not be cached as our internal cache requires a filename. If you want caching, add a filename that represents your file (it will be used only for caching):

```
import io
from kivy.core.image import Image as CoreImage
data = io.BytesIO(open("image.png", "rb").read())
im = CoreImage(data, ext="png", filename="image.png")
```

21.5.2 Saving an image

A CoreImage can be saved to a file:

```
from kivy.core.image import Image as CoreImage
image = CoreImage(...)
image.save("/tmp/test.png")
```

Or you can get the bytes (new in 1.11.0):

```
import io
from kivy.core.image import Image as CoreImage
data = io.BytesIO()
image = CoreImage(...)
image.save(data, fmt="png")
png_bytes = data.read()
```

class kivy.core.image.Image(*arg*, ***kwargs*)

Bases: *kivy.event.EventDispatcher*

Load an image and store the size and texture.

Changed in version 1.0.7: *mipmap* attribute has been added. The *texture_mipmap* and *texture_rectangle* have been deleted.

Changed in version 1.0.8: An Image widget can change its texture. A new event 'on_texture' has been introduced. New methods for handling sequenced animation have been added.

Parameters

arg: can be a string (str), Texture, BytesIO or Image object A string path to the image file or data URI to be loaded; or a Texture object, which will be wrapped in an Image object; or a BytesIO object containing raw image data; or an already existing image object, in which case, a real copy of the given image object will be returned.

keep_data: bool, defaults to False Keep the image data when the texture is created.

mipmap: bool, defaults to False Create mipmap for the texture.

anim_delay: float, defaults to .25 Delay in seconds between each animation frame.
Lower values means faster animation.

ext: str, only with BytesIO arg File extension to use in determining how to load raw image data.

filename: str, only with BytesIO arg Filename to use in the image cache for raw image data.

anim_available

Return True if this Image instance has animation available.

New in version 1.0.8.

anim_delay

Delay between each animation frame. A lower value means faster animation.

New in version 1.0.8.

anim_index

Return the index number of the image currently in the texture.

New in version 1.0.8.

anim_reset(allow_anim)

Reset an animation if available.

New in version 1.0.8.

Parameters

allow_anim: bool Indicate whether the animation should restart playing or not.

Usage:

```
# start/reset animation
image.anim_reset(True)

# or stop the animation
image.anim_reset(False)
```

You can change the animation speed whilst it is playing:

```
# Set to 20 FPS
image.anim_delay = 1 / 20.
```

filename

Get/set the filename of image

height

Image height

image

Get/set the data image object

static load(filename, **kwargs)

Load an image

Parameters

filename: str Filename of the image.

keep_data: bool, defaults to False Keep the image data when the texture is created.

load_memory(*data, ext, filename=__inline__*)
(internal) Method to load an image from raw data.

nocache

Indicate whether the texture will not be stored in the cache or not.

New in version 1.6.0.

on_texture(**largs*)

This event is fired when the texture reference or content has changed. It is normally used for sequenced images.

New in version 1.0.8.

read_pixel(*x, y*)

For a given local x/y position, return the pixel color at that position.

Warning: This function can only be used with images loaded with the `keep_data=True` keyword. For example:

```
m = Image.load('image.png', keep_data=True)
color = m.read_pixel(150, 150)
```

Parameters

x: `int` Local x coordinate of the pixel in question.

y: `int` Local y coordinate of the pixel in question.

remove_from_cache()

Remove the Image from cache. This facilitates re-loading of images from disk in case the image content has changed.

New in version 1.3.0.

Usage:

```
im = CoreImage('1.jpg')
# -- do something --
im.remove_from_cache()
im = CoreImage('1.jpg')
# this time image will be re-loaded from disk
```

save(*filename, flipped=False, fmt=None*)

Save image texture to file.

The filename should have the '.png' extension because the texture data read from the GPU is in the RGBA format. '.jpg' might work but has not been heavily tested so some providers might break when using it. Any other extensions are not officially supported.

The flipped parameter flips the saved image vertically, and defaults to False.

Example:

```
# Save an core image object
from kivy.core.image import Image
img = Image('hello.png')
img.save('hello2.png')

# Save a texture
texture = Texture.create(...)
```

(continues on next page)

```
img = Image(texture)
img.save('hello3.png')
```

New in version 1.7.0.

Changed in version 1.8.0: Parameter *flipped* added to flip the image before saving, default to False.

Changed in version 1.11.0: Parameter *fnt* added to force the output format of the file File-name can now be a BytesIO object.

size

Image size (width, height)

texture

Texture of the image

width

Image width

class `kivy.core.image.ImageData`(*width, height, fnt, data, source=None, flip_vertical=True, source_image=None, rowlength=0*)

Bases: `builtins.object`

Container for images and mipmap images. The container will always have at least the mipmap level 0.

add_mipmap(*level, width, height, data, rowlength*)

Add a image for a specific mipmap level.

New in version 1.0.7.

data

Image data. (If the image is mipmapped, it will use the level 0)

flip_vertical

Indicate if the texture will need to be vertically flipped

fnt

Decoded image format, one of a available texture format

get_mipmap(*level*)

Get the mipmap image at a specific level if it exists

New in version 1.0.7.

height

Image height in pixels. (If the image is mipmapped, it will use the level 0)

iterate_mipmaps()

Iterate over all mipmap images available.

New in version 1.0.7.

mipmaps

Data for each mipmap.

rowlength

Image rowlength. (If the image is mipmapped, it will use the level 0)

New in version 1.9.0.

size

Image (width, height) in pixels. (If the image is mipmapped, it will use the level 0)

source

Image source, if available

width

Image width in pixels. (If the image is mipmapped, it will use the level 0)

21.6 Spelling

Provides abstracted access to a range of spellchecking backends as well as word suggestions. The API is inspired by *enchant* but other backends can be added that implement the same API.

Spelling currently requires *python-enchant* for all platforms except OSX, where a native implementation exists.

```
>>> from kivy.core.spelling import Spelling
>>> s = Spelling()
>>> s.list_languages()
['en', 'en_CA', 'en_GB', 'en_US']
>>> s.select_language('en_US')
>>> s.suggest('hello')
[u'hole', u'help', u'helot', u'hello', u'halo', u'hero', u'hell', u'held',
 u'helm', u'he-lo']
```

class kivy.core.spelling.SpellingBase(*language=None*)

Bases: `builtins.object`

Base class for all spelling providers. Supports some abstract methods for checking words and getting suggestions.

check(*word*)

If *word* is a valid word in *self._language* (the currently active language), returns True. If the word shouldn't be checked, returns None (e.g. for ""). If it is not a valid word in *self._language*, return False.

Parameters

word: **str** The word to check.

list_languages()

Return a list of all supported languages. E.g. ['en', 'en_GB', 'en_US', 'de', ...]

select_language(*language*)

From the set of registered languages, select the first language for *language*.

Parameters

language: **str** Language identifier. Needs to be one of the options returned by `list_languages()`. Sets the language used for spell checking and word suggestions.

suggest(*fragment*)

For a given *fragment* (i.e. part of a word or a word by itself), provide corrections (*fragment* may be misspelled) or completions as a list of strings.

Parameters

fragment: **str** The word fragment to get suggestions/corrections for. E.g. 'foo' might become 'of', 'food' or 'foot'.

exception kivy.core.spelling.NoSuchLangError

Bases: `Exception`

Exception to be raised when a specific language could not be found.

exception `kivy.core.spelling.NoLanguageSelectedError`

Bases: `Exception`

Exception to be raised when a language-using method is called but no language was selected prior to the call.

21.7 Text

An abstraction of text creation. Depending of the selected backend, the accuracy of text rendering may vary.

Changed in version 1.10.1: `LabelBase.find_base_direction()` added.

Changed in version 1.5.0: `LabelBase.line_height` added.

Changed in version 1.0.7: The `LabelBase` does not generate any texture if the text has a width ≤ 1 .

This is the backend layer for rendering text with different text providers, you should only be using this directly if your needs aren't fulfilled by the `Label`.

Usage example:

```
from kivy.core.text import Label as CoreLabel

...
...
my_label = CoreLabel()
my_label.text = 'hello'
# the label is usually not drawn until needed, so force it to draw.
my_label.refresh()
# Now access the texture of the label and use it wherever and
# however you may please.
hello_texture = my_label.texture
```

21.7.1 Font Context Manager

A font context is a namespace where multiple fonts are loaded; if a font is missing a glyph needed to render text, it can fall back to a different font in the same context. The font context manager can be used to query and manipulate the state of font contexts when using the Pango text provider (no other provider currently implements it).

New in version 1.11.0.

Warning: This feature requires the Pango text provider.

Font contexts can be created automatically by `kivy.uix.label.Label` or `kivy.uix.textinput.TextInput`; if a non-existent context is used in one of these classes, it will be created automatically, or if a font file is specified without a context (this creates an isolated context, without support for fallback).

Usage example:

```
from kivy.uix.label import Label
from kivy.core.text import FontContextManager as FCM

# Create a font context containing system fonts + one custom TTF
FCM.create('system://myapp')
```

(continues on next page)


```
family = FCM.add_font('/path/to/file.ttf')

# These are now interchangeable ways to refer to the custom font:
lbl1 = Label(font_context='system://myapp', family_name=family)
lbl2 = Label(font_context='system://myapp', font_name='/path/to/file.ttf')

# You could also refer to a system font by family, since this is a
# system:// font context
lbl3 = Label(font_context='system://myapp', family_name='Arial')
```

```
class kivy.core.text.LabelBase(text="", font_size=12, font_name=None, bold=False,
                               italic=False, underline=False, strikethrough=False,
                               font_family=None, halign='left', valign='bottom',
                               shorten=False, text_size=None, mipmap=False,
                               color=None, line_height=1.0, strip=False,
                               strip_reflow=True, shorten_from='center', split_str='
', unicode_errors='replace', font_hinting='normal',
                               font_kerning=True, font_blended=True, out-
                               line_width=None, outline_color=None,
                               font_context=None, font_features=None,
                               base_direction=None, text_language=None, **kwargs)
```

Bases: `builtins.object`

Core text label. This is the abstract class used by different backends to render text.

Warning: The core text label can't be changed at runtime. You must recreate one.

Parameters

font_size: `int`, defaults to 12 Font size of the text

font_context: `str`, defaults to `None` Context for the specified font (see [kivy.uix.label.Label](#) for details). `None` will autocreate an isolated context named after the resolved font file.

font_name: `str`, defaults to `DEFAULT_FONT` Font name of the text

font_family: `str`, defaults to `None` Font family name to request for drawing, this can only be used with `font_context`.

bold: `bool`, defaults to `False` Activate “bold” text style

italic: `bool`, defaults to `False` Activate “italic” text style

text_size: `tuple`, defaults to `(None, None)` Add constraint to render the text (inside a bounding box). If no size is given, the label size will be set to the text size.

padding: `float`, defaults to `None` If it's a float, it will set `padding_x` and `padding_y`

padding_x: `float`, defaults to 0.0 Left/right padding

padding_y: `float`, defaults to 0.0 Top/bottom padding

halign: `str`, defaults to “left” Horizontal text alignment inside the bounding box

valign: `str`, defaults to “bottom” Vertical text alignment inside the bounding box

shorten: `bool`, defaults to `False` Indicate whether the label should attempt to shorten its textual contents as much as possible if a `size` is given. Setting this to `True` without an appropriately set size will lead to unexpected results.

***shorten_from*: str, defaults to center** The side from which we should shorten the text from, can be left, right, or center. E.g. if left, the ellipsis will appear towards the left side and it will display as much text starting from the right as possible.

***split_str*: string, defaults to ' ' (space)** The string to use to split the words by when shortening. If empty, we can split after every character filling up the line as much as possible.

***max_lines*: int, defaults to 0 (unlimited)** If set, this indicate how maximum line are allowed to render the text. Works only if a limitation on *text_size* is set.

***mipmap*: bool, defaults to False** Create a mipmap for the texture

***strip*: bool, defaults to False** Whether each row of text has its leading and trailing spaces stripped. If *halign* is *justify* it is implicitly True.

***strip_reflow*: bool, defaults to True** Whether text that has been reflowed into a second line should be stripped, even if *strip* is False. This is only in effect when *size_hint_x* is not None, because otherwise lines are never split.

***unicode_errors*: str, defaults to 'replace'** How to handle unicode decode errors. Can be 'strict', 'replace' or 'ignore'.

***outline_width*: int, defaults to None** Width in pixels for the outline.

***outline_color*: tuple, defaults to (0, 0, 0)** Color of the outline.

***font_features*: str, defaults to None** OpenType font features in CSS format (Pango only)

***base_direction*: str, defaults to None (auto)** Text direction, one of *None*, 'ltr', 'rtl', 'weak_ltr', or 'weak_rtl' (Pango only)

***text_language*: str, defaults to None (user locale)** RFC-3066 format language tag as a string (Pango only)

Changed in version 1.10.1: *font_context*, *font_family*, *font_features*, *base_direction* and *text_language* were added.

Changed in version 1.10.0: *outline_width* and *outline_color* were added.

Changed in version 1.9.0: *strip*, *strip_reflow*, *shorten_from*, *split_str*, and *unicode_errors* were added.

Changed in version 1.9.0: *padding_x* and *padding_y* has been fixed to work as expected. In the past, the text was padded by the negative of their values.

Changed in version 1.8.0: *max_lines* parameters has been added.

Changed in version 1.0.8: *size* have been deprecated and replaced with *text_size*.

Changed in version 1.0.7: The *valign* is now respected. This wasn't the case previously so you might have an issue in your application if you have not considered this.

content_height

Return the content height; i.e. the height of the text without any padding.

content_size

Return the content size (width, height)

content_width

Return the content width; i.e. the width of the text without any padding.

static find_base_direction(text)

Searches a string the first character that has a strong direction, according to the Unicode bidirectional algorithm. Returns *None* if the base direction cannot be determined, or one of 'ltr' or 'rtl'.

Note: This feature requires the Pango text provider.

fontid

Return a unique id for all font parameters

get_cached_extents()

Returns a cached version of the *get_extents()* function.

```
>>> func = self._get_cached_extents()
>>> func
<built-in method size of pygame.font.Font object at 0x01E45650>
>>> func('a line')
(36, 18)
```

Warning: This method returns a size measuring function that is valid for the font settings used at the time *get_cached_extents()* was called. Any change in the font settings will render the returned function incorrect. You should only use this if you know what you're doing.

New in version 1.9.0.

get_extents(text)

Return a tuple (width, height) indicating the size of the specified text

static get_system_fonts_dir()

Return the directories used by the system for fonts.

label

Get/Set the text

refresh()

Force re-rendering of the text

static register(name, fn_regular, fn_italic=None, fn_bold=None, fn_bolditalic=None)

Register an alias for a Font.

New in version 1.1.0.

If you're using a ttf directly, you might not be able to use the bold/italic properties of the ttf version. If the font is delivered in multiple files (one regular, one italic and one bold), then you need to register these files and use the alias instead.

All the *fn_regular*/*fn_italic*/*fn_bold* parameters are resolved with *kivy.resources.resource_find()*. If *fn_italic*/*fn_bold* are None, *fn_regular* will be used instead.

render(real=False)

Return a tuple (width, height) to create the image with the user constraints. (width, height) includes the padding.

shorten(text, margin=2)

Shortens the text to fit into a single line by the width specified by *text_size* [0]. If *text_size* [0] is None, it returns text text unchanged.

split_str and *shorten_from* determines how the text is shortened.

Params *text* str, the text to be shortened. *margin* int, the amount of space to leave between the margins and the text. This is in addition to *padding_x*.

Returns the text shortened to fit into a single line.

text

Get/Set the text

text_size

Get/set the (width, height) of the ' ' constrained rendering box

usersize

(deprecated) Use text_size instead.

21.7.2 Text layout

An internal module for laying out text according to options and constraints. This is not part of the API and may change at any time.

`kivy.core.text.text_layout.layout_text(text, list lines, tuple size, tuple text_size, dict options, get_extents, int append_down, int complete)`

Lays out text into a series of [LayoutWord](#) and [LayoutLine](#) instances according to the options specified.

The function is designed to be called many times, each time new text is appended to the last line (or first line if appending upwards), unless a newline is present in the text. Each text appended is described by its own options which can change between successive calls. If the text is constrained, we stop as soon as the constraint is reached.

Parameters

text: **string or bytes** the text to be broken down into lines. If lines is not empty, the text is added to the last line (or first line if *append_down* is False) until a newline is reached which creates a new line in *lines*. See [LayoutLine](#).

lines: **list** a list of [LayoutLine](#) instances, each describing a line of the text. Calls to *layout_text()* append or create new [LayoutLine](#) instances in *lines*.

size: **2-tuple of ints** the size of the laid out text so far. Upon first call it should probably be (0, 0), afterwards it should be the (w, h) returned by this function in a previous call. When size reaches the constraining size, *text_size*, we stop adding lines and return True for the clipped parameter. size includes the x and y padding.

text_size: **2-tuple of ints or None.** the size constraint on the laid out text. If either element is None, the text is not constrained in that dimension. For example, (None, 200) will constrain the height, including padding to 200, while the width is unconstrained. The first line, and the first character of a line is always returned, even if it exceeds the constraint. The value be changed between different calls.

options: **dict** the label options of this *text*. The options are saved with each word allowing different words to have different options from successive calls.

Note, *options* must include a *space_width* key with a value indicating the width of a space for that set of options.

get_extents: **callable** a function called with a string, which returns a tuple containing the width, height of the string.

append_down: **bool** Whether successive calls to the function appends lines before or after the existing lines. If True, they are appended to the last line and below it. If False, it's appended at the first line and above. For example, if False, everything after the last newline in *text* is appended to the first line in *lines*. Everything before the last newline is inserted at the start of lines in same order as text; that is we do not invert the line order.

This allows laying out from top to bottom until the constrained is reached, or from bottom to top until the constrained is reached.

complete: bool whether this text complete lines. It use is that normally is strip in *options* is True, all leading and trailing spaces are removed from each line except from the last line (or first line if *append_down* is False) which only removes leading spaces. That's because further text can still be appended to the last line so we cannot strip them. If *complete* is True, it indicates no further text is coming and all lines will be stripped.

The function can also be called with *text* set to the empty string and *complete* set to True in order for the last (first) line to be stripped.

Returns 3-tuple, (w, h, clipped). w and h is the width and height of the text in lines so far and includes padding. This can be larger than *text_size*, e.g. if not even a single fitted, the first line would still be returned. *clipped* is True if not all the text has been added to lines because w, h reached the constrained size.

Following is a simple example with no padding and no stripping:

```
>>> from kivy.core.text import Label
>>> from kivy.core.text.text_layout import layout_text

>>> l = Label()
>>> lines = []
>>> # layout text with width constraint by 50, but no height constraint
>>> w, h, clipped = layout_text('heres some text\nah, another line',
... lines, (0, 0), (50, None), l.options, l.get_cached_extents(), True,
... False)
>>> w, h, clipped
(46, 90, False)
# now add text from bottom up, and constrain width only be 100
>>> w, h, clipped = layout_text('\nyay, more text\n', lines, (w, h),
... (100, None), l.options, l.get_cached_extents(), False, True)
>>> w, h, clipped
(77, 120, 0)
>>> for line in lines:
...     print('line w: {}, line h: {}'.format(line.w, line.h))
...     for word in line.words:
...         print('w: {}, h: {}, text: {}'.format(word.lw, word.lh,
... [word.text]))
line w: 0, line h: 15
line w: 77, line h: 15
w: 77, h: 15, text: ['yay, more text']
line w: 31, line h: 15
w: 31, h: 15, text: ['heres']
line w: 34, line h: 15
w: 34, h: 15, text: [' some']
line w: 24, line h: 15
w: 24, h: 15, text: [' text']
line w: 17, line h: 15
w: 17, h: 15, text: ['ah,']
line w: 46, line h: 15
w: 46, h: 15, text: [' another']
line w: 23, line h: 15
w: 23, h: 15, text: [' line']
```

class kivy.core.text.text_layout.LayoutWord
Bases: builtins.object

Formally describes a word contained in a line. The name word simply means a chunk of text and

can be used to describe any text.

A word has some width, height and is rendered according to options saved in *options*. See *LayoutLine* for its usage.

Parameters

options: dict the label options dictionary for this word.

lw: int the width of the text in pixels.

lh: int the height of the text in pixels.

text: string the text of the word.

lh

lh: 'int'

lw

lw: 'int'

options

options: dict

text

text: object

class `kivy.core.text.text_layout.LayoutLine`

Bases: `builtins.object`

Formally describes a line of text. A line of text is composed of many *LayoutWord* instances, each with it's own text, size and options.

A *LayoutLine* instance does not always imply that the words contained in the line ended with a newline. That is only the case if *is_last_line* is True. For example a single real line of text can be split across multiple *LayoutLine* instances if the whole line doesn't fit in the constrained width.

Parameters

x: int the location in a texture from where the left side of this line is began drawn.

y: int the location in a texture from where the bottom of this line is drawn.

w: int the width of the line. This is the sum of the individual widths of its *LayoutWord* instances. Does not include any padding.

h: int the height of the line. This is the maximum of the individual heights of its *LayoutWord* instances multiplied by the *line_height* of these instance. So this is larger then the word height.

is_last_line: bool whether this line was the last line in a paragraph. When True, it implies that the line was followed by a newline. Newlines should not be included in the text of words, but is implicit by setting this to True.

line_wrap: bool whether this line is continued from a previous line which didn't fit into a constrained width and was therefore split across multiple *LayoutLine* instances. *line_wrap* can be True or False independently of *is_last_line*.

words: python list a list that contains only *LayoutWord* instances describing the text of the line.

h

h: 'int'

is_last_line

is_last_line: 'int'

line_wrap
 line_wrap: 'int'

w
 w: 'int'

words
 words: list

x
 x: 'int'

y
 y: 'int'

21.7.3 Text Markup

New in version 1.1.0.

Changed in version 1.10.1: Added *font_context*, *font_features* and *text_language* (Pango only)

We provide a simple text-markup for inline text styling. The syntax look the same as the **BBCode**.

A tag is defined as `[tag]`, and should have a corresponding `[/tag]` closing tag. For example:

```
[b>Hello [color=ff0000]world[/color][b]
```

The following tags are available:

[b][b] Activate bold text

[i][i] Activate italic text

[u][u] Underlined text

[s][s] Strikethrough text

[font=<str>][font] Change the font (note: this refers to a TTF file or registered alias)

[font_context=<str>][font_context] Change context for the font, use string value "none" for isolated context.

[font_family=<str>][font_family] Font family to request for drawing. This is only valid when using a font context, and takes precedence over *[font]*. See *kivy.uix.label.Label* for details.

[font_features=<str>][font_features] OpenType font features, in CSS format, this is passed straight through to Pango. The effects of requesting a feature depends on loaded fonts, library versions, etc. Pango only, requires v1.38 or later.

[size=<size>][size] Change the font size. <size> should be an integer, optionally with a unit (i.e. 16sp)

[color=#<color>][color] Change the text color

[ref=<str>][ref] Add an interactive zone. The reference + all the word box inside the reference will be available in *MarkupLabel.refs*

[anchor=<str>] Put an anchor in the text. You can get the position of your anchor within the text with *MarkupLabel.anchors*

[sub][sub] Display the text at a subscript position relative to the text before it.

[sup][sup] Display the text at a superscript position relative to the text before it.

[text_language=<str>][/text_language**]** Language of the text, this is an RFC-3066 format language tag (as string), for example “en_US”, “zh_CN”, “fr” or “ja”. This can impact font selection, metrics and rendering. For example, the same bytes of text can look different for *ur* and *ar* languages, though both use Arabic script. Use the string ‘none’ to revert to locale detection. Pango only.

If you need to escape the markup from the current text, use `kivy.utils.escape_markup()`.

class kivy.core.text.markup.**MarkupLabel**(**largs, **kwargs*)

Bases: `kivy.core.text.LabelBase`

Markup text label.

See module documentation for more informations.

anchors

Get the position of all the [**anchor**=...]:

```
{ 'anchorA': (x, y), 'anchorB': (x, y), ... }
```

markup

Return the text with all the markup splitted:

```
>>> MarkupLabel('[b>Hello world[/b]').markup
>>> ('[b]', 'Hello world', '[/b]')
```

refs

Get the bounding box of all the [**ref**=...]:

```
{ 'refA': ((x1, y1, x2, y2), (x1, y1, x2, y2)), ... }
```

render(*real=False*)

Return a tuple (width, height) to create the image with the user constraints. (width, height) includes the padding.

shorten_post(*lines, w, h, margin=2*)

Shortens the text to a single line according to the label options.

This function operates on a text that has already been laid out because for markup, parts of text can have different size and options.

If `text_size[0]` is None, the lines are returned unchanged. Otherwise, the lines are converted to a single line fitting within the constrained width, `text_size[0]`.

Params *lines*: list of *LayoutLine* instances describing the text. *w*: int, the width of the text in lines, including padding. *h*: int, the height of the text in lines, including padding. *margin* int, the additional space left on the sides. This is in addition to `padding_x`.

Returns 3-tuple of (xw, h, lines), where *w*, and *h* is similar to the input and contains the resulting width / height of the text, including padding. *lines*, is a list containing a single *LayoutLine*, which contains the words for the line.

21.8 Video

Core class for reading video files and managing the video *Texture*.

Changed in version 1.10.0: The pyglet, pygst and gi providers have been removed.

Changed in version 1.8.0: There are now 2 distinct Gstreamer implementations: one using Gi/Gst working for both Python 2+3 with Gstreamer 1.0, and one using PyGST working only for Python 2 + Gstreamer 0.10.

Note: Recording is not supported.

class `kivy.core.video.VideoBase(**kwargs)`

Bases: `kivy.event.EventDispatcher`

VideoBase, a class used to implement a video reader.

Parameters

filename: `str` Filename of the video. Can be a file or an URI.

eos: `str`, defaults to `'pause'` Action to take when EOS is hit. Can be one of `'pause'`, `'stop'` or `'loop'`.

Changed in version 1.4.0: added `'pause'`

async: `bool`, defaults to `True` Load the video asynchronously (may be not supported by all providers).

autoplay: `bool`, defaults to `False` Auto play the video on init.

Events

on_eos Fired when EOS is hit.

on_load Fired when the video is loaded and the texture is available.

on_frame Fired when a new frame is written to the texture.

duration

Get the video duration (in seconds)

filename

Get/set the filename/uri of the current video

load()

Load the video from the current filename

pause()

Pause the video

New in version 1.4.0.

play()

Play the video

position

Get/set the position in the video (in seconds)

seek(percent, precise=True)

Move on percent position

state

Get the video playing status

stop()

Stop the video playing

texture

Get the video texture

unload()

Unload the actual video

volume

Get/set the volume in the video (1.0 = 100%)

21.9 Window

Core class for creating the default Kivy window. Kivy supports only one window per application: please don't try to create more than one.

class `kivy.core.window.Keyboard`(**kwargs)

Bases: `kivy.event.EventDispatcher`

Keyboard interface that is returned by `WindowBase.request_keyboard()`. When you request a keyboard, you'll get an instance of this class. Whatever the keyboard input is (system or virtual keyboard), you'll receive events through this instance.

Events

on_key_down: `keycode, text, modifiers` Fired when a new key is pressed down

on_key_up: `keycode` Fired when a key is released (up)

Here is an example of how to request a Keyboard in accordance with the current configuration:

```
import kivy
kivy.require('1.0.8')

from kivy.core.window import Window
from kivy.uix.widget import Widget

class MyKeyboardListener(Widget):

    def __init__(self, **kwargs):
        super(MyKeyboardListener, self).__init__(**kwargs)
        self._keyboard = Window.request_keyboard(
            self._keyboard_closed, self, 'text')
        if self._keyboard.widget:
            # If it exists, this widget is a VKeyboard object which you can_
            ↪use
            # to change the keyboard layout.
            pass
        self._keyboard.bind(on_key_down=self._on_keyboard_down)

    def _keyboard_closed(self):
        print('My keyboard have been closed!')
        self._keyboard.unbind(on_key_down=self._on_keyboard_down)
        self._keyboard = None

    def _on_keyboard_down(self, keyboard, keycode, text, modifiers):
        print('The key', keycode, 'have been pressed')
        print(' - text is %r' % text)
        print(' - modifiers are %r' % modifiers)

        # Keycode is composed of an integer + a string
        # If we hit escape, release the keyboard
        if keycode[1] == 'escape':
            keyboard.release()
```

(continues on next page)

```

    # Return True to accept the key. Otherwise, it will be used by
    # the system.
    return True

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MyKeyboardListener())

```

callback = None

Callback that will be called when the keyboard is released

keycode_to_string(value)

Convert a keycode number to a string according to the `Keyboard.keycodes`. If the value is not found in the keycodes, it will return "".

release()

Call this method to release the current keyboard. This will ensure that the keyboard is no longer attached to your callback.

string_to_keycode(value)

Convert a string to a keycode number according to the `Keyboard.keycodes`. If the value is not found in the keycodes, it will return -1.

target = None

Target that have requested the keyboard

widget = None

VKeyboard widget, if allowed by the configuration

window = None

Window which the keyboard is attached too

class `kivy.core.window.WindowBase(**kwargs)`

Bases: `kivy.event.EventDispatcher`

WindowBase is an abstract window widget for any window implementation.

Parameters

borderless: str, one of ('0', '1') Set the window border state. Check the `config` documentation for a more detailed explanation on the values.

fullscreen: str, one of ('0', '1', 'auto', 'fake') Make the window fullscreen. Check the `config` documentation for a more detailed explanation on the values.

width: int Width of the window.

height: int Height of the window.

minimum_width: int Minimum width of the window (only works for sdl2 window provider).

minimum_height: int Minimum height of the window (only works for sdl2 window provider).

allow_screensaver: bool Allow the device to show a screen saver, or to go to sleep on mobile devices. Defaults to True. Only works for sdl2 window provider.

Events

on_motion: etype, motionevent Fired when a new `MotionEvent` is dispatched

on_touch_down: Fired when a new touch event is initiated.

on_touch_move: Fired when an existing touch event changes location.

on_touch_up: Fired when an existing touch event is terminated.

on_draw: Fired when the Window is being drawn.

on_flip: Fired when the Window GL surface is being flipped.

on_rotate: **rotation** Fired when the Window is being rotated.

on_close: Fired when the Window is closed.

on_request_close: Fired when the event loop wants to close the window, or if the escape key is pressed and *exit_on_escape* is *True*. If a function bound to this event returns *True*, the window will not be closed. If the event is triggered because of the keyboard escape key, the keyword argument *source* is dispatched along with a value of *keyboard* to the bound functions.

New in version 1.9.0.

on_cursor_enter: Fired when the cursor enters the window.

New in version 1.9.1.

on_cursor_leave: Fired when the cursor leaves the window.

New in version 1.9.1.

on_minimize: Fired when the window is minimized.

New in version 1.10.0.

on_maximize: Fired when the window is maximized.

New in version 1.10.0.

on_restore: Fired when the window is restored.

New in version 1.10.0.

on_hide: Fired when the window is hidden.

New in version 1.10.0.

on_show: Fired when when the window is shown.

New in version 1.10.0.

on_keyboard: **key, scancode, codepoint, modifier** Fired when the keyboard is used for input.

Changed in version 1.3.0: The *unicode* parameter has been deprecated in favor of codepoint, and will be removed completely in future versions.

on_key_down: **key, scancode, codepoint, modifier** Fired when a key pressed.

Changed in version 1.3.0: The *unicode* parameter has been deprecated in favor of codepoint, and will be removed completely in future versions.

on_key_up: **key, scancode, codepoint** Fired when a key is released.

Changed in version 1.3.0: The *unicode* parameter has been deprecated in favor of codepoint, and will be removed completely in future versions.

on_dropfile: **str** Fired when a file is dropped on the application.

Note: This event doesn't work for apps with elevated permissions, because the OS API calls are filtered. Check issue [#4999](#) for pointers to workarounds.

on_memorywarning: Fired when the platform have memory issue (iOS / Android mostly) You can listen to this one, and clean whatever you can.

New in version 1.9.0.

on_textedit(self, text): Fired when inputting with IME. The string inputting with IME is set as the parameter of this event.

New in version 1.10.1.

add_widget(*widget*, *canvas=None*)
Add a widget to a window

allow_screensaver

Whether the screen saver is enabled, or on mobile devices whether the device is allowed to go to sleep while the app is open.

New in version 1.10.0.

allow_screensaver is a *BooleanProperty* and defaults to True.

borderless

When set to True, this property removes the window border/decoration. Check the *config* documentation for a more detailed explanation on the values.

New in version 1.9.0.

borderless is a *BooleanProperty* and defaults to False.

center

Center of the rotated window.

New in version 1.0.9.

center is an *AliasProperty*.

children

List of the children of this window.

children is a *ListProperty* instance and defaults to an empty list.

Use *add_widget()* and *remove_widget()* to manipulate the list of children. Don't manipulate the list directly unless you know what you are doing.

clear()

Clear the window with the background color

clearcolor

Color used to clear the window.

```
from kivy.core.window import Window

# red background color
Window.clearcolor = (1, 0, 0, 1)

# don't clear background at all
Window.clearcolor = None
```

Changed in version 1.7.2: The clearcolor default value is now: (0, 0, 0, 1).

New in version 1.0.9.

clearcolor is an *AliasProperty* and defaults to (0, 0, 0, 1).

close()

Close the window

create_window(*largs)

Will create the main window and configure it.

Warning: This method is called automatically at runtime. If you call it, it will recreate a `RenderContext` and `Canvas`. This means you'll have a new graphics tree, and the old one will be unusable.

This method exist to permit the creation of a new OpenGL context AFTER closing the first one. (Like using `runTouchApp()` and `stopTouchApp()`).

This method has only been tested in a unittest environment and is not suitable for Applications.

Again, don't use this method unless you know exactly what you are doing!

dpi()

Return the DPI of the screen. If the implementation doesn't support any DPI lookup, it will just return 96.

Warning: This value is not cross-platform. Use `kivy.base.EventLoop.dpi` instead.

flip()

Flip between buffers

focus

Check whether or not the window currently has focus.

New in version 1.9.1.

focus is a read-only *AliasProperty* and defaults to `True`.

fullscreen

This property sets the fullscreen mode of the window. Available options are: `True`, `False`, `'auto'` and `'fake'`. Check the *config* documentation for more detailed explanations on these values.

fullscreen is an *OptionProperty* and defaults to `False`.

New in version 1.2.0.

Note: The `'fake'` option has been deprecated, use the *borderless* property instead.

get_gl_backend_name()

Returns the gl backend that will or is used with this window.

gl_backends_allowed = []

A list of Kivy gl backend names, which if not empty, will be the exclusive list of gl backends that can be used with this window.

gl_backends_ignored = []

A list of Kivy gl backend names that may not be used with this window.

grab_mouse()

Grab mouse - so won't leave window

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

height

Rotated window height.

height is a read-only *AliasProperty*.

hide()

Hides the window. This method should be used on desktop platforms only.

New in version 1.9.0.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

icon

A path to the window icon.

New in version 1.1.2.

icon is a *StringProperty*.

keyboard_anim_args = {'d': 0.5, 't': 'in_out_quart'}

The attributes for animating softkeyboard/IME. *t* = *transition*, *d* = *duration*. This value will have no effect on desktops.

New in version 1.10.0.

keyboard_anim_args is a dict and defaults to {'t': 'in_out_quart', 'd': .5}.

keyboard_height

Returns the height of the softkeyboard/IME on mobile platforms. Will return 0 if not on mobile platform or if IME is not active.

Note: This property returns 0 with SDL2 on Android, but setting `Window.softinput_mode` does work.

New in version 1.9.0.

keyboard_height is a read-only *AliasProperty* and defaults to 0.

keyboard_padding

The padding to have between the softkeyboard/IME & target or bottom of window. Will have no effect on desktops.

New in version 1.10.0.

keyboard_padding is a *NumericProperty* and defaults to 0.

left

Left position of the window.

Note: It's an SDL2 property with `[0, 0]` in the top-left corner.

Changed in version 1.10.0: *left* is now an *AliasProperty*

New in version 1.9.1.

left is an *AliasProperty* and defaults to the position set in *Config*.

mainloop()

Called by the EventLoop every frame after it idles.

maximize()

Maximizes the window. This method should be used on desktop platforms only.

New in version 1.9.0.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

minimize()

Minimizes the window. This method should be used on desktop platforms only.

New in version 1.9.0.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

minimum_height

The minimum height to restrict the window to.

New in version 1.9.1.

minimum_height is a *NumericProperty* and defaults to 0.

minimum_width

The minimum width to restrict the window to.

New in version 1.9.1.

minimum_width is a *NumericProperty* and defaults to 0.

modifiers

List of keyboard modifiers currently active.

New in version 1.0.9.

modifiers is an *AliasProperty*.

mouse_pos

2d position of the mouse within the window.

New in version 1.2.0.

mouse_pos is an *ObjectProperty* and defaults to [0, 0].

on_close(*largs)

Event called when the window is closed.

on_cursor_enter(*largs)

Event called when the cursor enters the window.

New in version 1.9.1.

Note: This feature requires the SDL2 window provider.

on_cursor_leave(*largs)

Event called when the cursor leaves the window.

New in version 1.9.1.

Note: This feature requires the SDL2 window provider.

on_dropfile(*filename*)

Event called when a file is dropped on the application.

Warning: This event currently works with sdl2 window provider, on pygame window provider and OS X with a patched version of pygame. This event is left in place for further evolution (ios, android etc.)

New in version 1.2.0.

on_flip()

Flip between buffers (event)

on_hide(**largs*)

Event called when the window is hidden.

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

on_joy_axis(*stickid, axisid, value*)

Event called when a joystick has a stick or other axis moved.

New in version 1.9.0.

on_joy_ball(*stickid, ballid, xvalue, yvalue*)

Event called when a joystick has a ball moved.

New in version 1.9.0.

on_joy_button_down(*stickid, buttonid*)

Event called when a joystick has a button pressed.

New in version 1.9.0.

on_joy_button_up(*stickid, buttonid*)

Event called when a joystick has a button released.

New in version 1.9.0.

on_joy_hat(*stickid, hatid, value*)

Event called when a joystick has a hat/dpad moved.

New in version 1.9.0.

on_key_down(*key, scancode=None, codepoint=None, modifier=None, **kwargs*)

Event called when a key is down (same arguments as on_keyboard)

on_key_up(*key, scancode=None, codepoint=None, modifier=None, **kwargs*)

Event called when a key is released (same arguments as on_keyboard).

on_keyboard(*key, scancode=None, codepoint=None, modifier=None, **kwargs*)

Event called when keyboard is used.

Warning: Some providers may omit *scancode*, *codepoint* and/or *modifier*.

on_maximize(*largs)

Event called when the window is maximized.

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

on_memorywarning()

Event called when the platform have memory issue. Your goal is to clear the cache in your app as much as you can, release unused widgets, do garbage collection etc.

Currently, this event is fired only from the SDL2 provider, for iOS and Android.

New in version 1.9.0.

on_minimize(*largs)

Event called when the window is minimized.

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

on_motion(etype, me)

Event called when a Motion Event is received.

Parameters

etype: str One of 'begin', 'update', 'end'

me: *MotionEvent* The Motion Event currently dispatched.

on_mouse_down(x, y, button, modifiers)

Event called when the mouse is used (pressed/released).

on_mouse_move(x, y, modifiers)

Event called when the mouse is moved with buttons pressed.

on_mouse_up(x, y, button, modifiers)

Event called when the mouse is moved with buttons pressed.

on_request_close(*largs, **kwargs)

Event called before we close the window. If a bound function returns *True*, the window will not be closed. If the the event is triggered because of the keyboard escape key, the keyword argument *source* is dispatched along with a value of *keyboard* to the bound functions.

Warning: When the bound function returns *True* the window will not be closed, so use with care because the user would not be able to close the program, even if the red X is clicked.

on_resize(width, height)

Event called when the window is resized.

on_restore(*largs)

Event called when the window is restored.

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

on_rotate(*rotation*)

Event called when the screen has been rotated.

on_show(**largs*)

Event called when the window is shown.

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

on_textedit(*text*)

Event called when inputting with IME. The string inputting with IME is set as the parameter of this event.

New in version 1.10.1.

on_textinput(*text*)

Event called when text: i.e. alpha numeric non control keys or set of keys is entered. As it is not guaranteed whether we get one character or multiple ones, this event supports handling multiple characters.

New in version 1.9.0.

on_touch_down(*touch*)

Event called when a touch down event is initiated.

Changed in version 1.9.0: The touch *pos* is now transformed to window coordinates before this method is called. Before, the touch *pos* coordinate would be (0, 0) when this method was called.

on_touch_move(*touch*)

Event called when a touch event moves (changes location).

Changed in version 1.9.0: The touch *pos* is now transformed to window coordinates before this method is called. Before, the touch *pos* coordinate would be (0, 0) when this method was called.

on_touch_up(*touch*)

Event called when a touch event is released (terminated).

Changed in version 1.9.0: The touch *pos* is now transformed to window coordinates before this method is called. Before, the touch *pos* coordinate would be (0, 0) when this method was called.

parent

Parent of this window.

parent is a *ObjectProperty* instance and defaults to None. When created, the parent is set to the window itself. You must take care of it if you are doing a recursive check.

raise_window()

Raise the window. This method should be used on desktop platforms only.

New in version 1.9.1.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

release_all_keyboards()

New in version 1.0.8.

This will ensure that no virtual keyboard / system keyboard is requested. All instances will be closed.

release_keyboard(*target=None*)

New in version 1.0.4.

Internal method for the widget to release the real-keyboard. Check [request_keyboard\(\)](#) to understand how it works.

remove_widget(*widget*)

Remove a widget from a window

request_keyboard(*callback, target, input_type='text'*)

New in version 1.0.4.

Internal widget method to request the keyboard. This method is rarely required by the end-user as it is handled automatically by the [TextInput](#). We expose it in case you want to handle the keyboard manually for unique input scenarios.

A widget can request the keyboard, indicating a callback to call when the keyboard is released (or taken by another widget).

Parameters

callback: func Callback that will be called when the keyboard is closed. This can be because somebody else requested the keyboard or the user closed it.

target: Widget Attach the keyboard to the specified *target*. This should be the widget that requested the keyboard. Ensure you have a different target attached to each keyboard if you're working in a multi user mode.

New in version 1.0.8.

input_type: string Choose the type of soft keyboard to request. Can be one of 'text', 'number', 'url', 'mail', 'datetime', 'tel', 'address'.

Note: *input_type* is currently only honored on mobile devices.

New in version 1.8.0.

Return An instance of [Keyboard](#) containing the callback, target, and if the configuration allows it, a [VKeyboard](#) instance attached as a *.widget* property.

Note: The behavior of this function is heavily influenced by the current *keyboard_mode*. Please see the Config's [configuration tokens](#) section for more information.

restore()

Restores the size and position of a maximized or minimized window. This method should be used on desktop platforms only.

New in version 1.9.0.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

rotation

Get/set the window content rotation. Can be one of 0, 90, 180, 270 degrees.

New in version 1.0.9.

rotation is an *AliasProperty*.

screenshot(*name*='screenshot{:04d}.png')

Save the actual displayed image to a file.

set_icon(*filename*)

Set the icon of the window.

New in version 1.0.5.

set_system_cursor(*cursor_name*)

Set type of a mouse cursor in the Window.

It can be one of 'arrow', 'ibeam', 'wait', 'crosshair', 'wait_arrow', 'size_nwse', 'size_nesw', 'size_we', 'size_ns', 'size_all', 'no', or 'hand'.

On some platforms there might not be a specific cursor supported and such an option falls back to one of the substitutable alternatives:

	Windows	MacOS	Linux X11	Linux Wayland
arrow	arrow	arrow	arrow	arrow
ibeam	ibeam	ibeam	ibeam	ibeam
wait	wait	arrow	wait	wait
crosshair	crosshair	crosshair	crosshair	hand
wait_arrow	arrow	arrow	wait	wait
size_nwse	size_nwse	size_all	size_all	hand
size_nesw	size_nesw	size_all	size_all	hand
size_we	size_we	size_we	size_we	hand
size_ns	size_ns	size_ns	size_ns	hand
size_all	size_all	size_all	size_all	hand
no	no	no	no	ibeam
hand	hand	hand	hand	hand

New in version 1.10.1.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

set_title(*title*)

Set the window title.

New in version 1.0.5.

set_vkeyboard_class(*cls*)

New in version 1.0.8.

Set the VKeyboard class to use. If set to *None*, it will use the *kivy.uix.vkeyboard.VKeyboard*.

shape_color_key

Color key of the shaped window - sets which color will be hidden from the window *shape_image* (only works for sdl2 window provider).

New in version 1.10.1.

shape_color_key is a *ListProperty* instance and defaults to [1, 1, 1, 1].

shape_cutoff

The window *shape_image* cutoff property (only works for sdl2 window provider).

New in version 1.10.1.

shape_cutoff is a *BooleanProperty* and defaults to True.

shape_image

An image for the window shape (only works for sdl2 window provider).

Warning: The image size has to be the same like the window's size!

New in version 1.10.1.

shape_image is a *StringProperty* and defaults to 'data/images/defaultshape.png'. This value is taken from *Config*.

shape_mode

Window mode for shaping (only works for sdl2 window provider).

- **can be RGB only**
 - *default* - does nothing special
 - *colorkey* - hides a color of the *shape_color_key*
- **has to contain alpha channel**
 - *binalpha* - hides an alpha channel of the *shape_image*
 - *reversebinalpha* - shows only the alpha of the *shape_image*

Note: Before actually setting the mode make sure the Window has the same size like the *shape_image*, preferably via Config before the Window is actually created.

If the *shape_image* isn't set, the default one will be used and the mode might not take the desired visual effect.

New in version 1.10.1.

shape_mode is an *AliasProperty*.

shaped

Read only property to check if the window is shapable or not (only works for sdl2 window provider).

New in version 1.10.1.

shaped is an *AliasProperty*.

show()

Shows the window. This method should be used on desktop platforms only.

New in version 1.9.0.

Note: This feature requires the SDL2 window provider and is currently only supported on desktop platforms.

show_cursor

Set whether or not the cursor is shown on the window.

New in version 1.9.1.

show_cursor is a *BooleanProperty* and defaults to True.

size

Get the rotated size of the window. If *rotation* is set, then the size will change to reflect the rotation.

New in version 1.0.9.

size is an *AliasProperty*.

softinput_mode

This specifies the behavior of window contents on display of the soft keyboard on mobile platforms. It can be one of `""`, `'pan'`, `'scale'`, `'resize'` or `'below_target'`. Their effects are listed below.

Value	Effect
<code>""</code>	The main window is left as is, allowing you to use the <i>keyboard_height</i> to manage the window contents manually.
<code>'pan'</code>	The main window pans, moving the bottom part of the window to be always on top of the keyboard.
<code>'resize'</code>	The window is resized and the contents scaled to fit the remaining space.
<code>'below_target'</code>	The window pans so that the current target TextInput widget requesting the keyboard is presented just above the soft keyboard.

softinput_mode is an *OptionProperty* and defaults to `""`.

Note: The *resize* option does not currently work with SDL2 on Android.

New in version 1.9.0.

Changed in version 1.9.1: The `'below_target'` option was added.

system_size

Real size of the window ignoring rotation.

New in version 1.0.9.

system_size is an *AliasProperty*.

toggle_fullscreen()

Toggle between fullscreen and windowed mode.

Deprecated since version 1.9.0: Use *fullscreen* instead.

top

Top position of the window.

Note: It's an SDL2 property with `[0, 0]` in the top-left corner.

Changed in version 1.10.0: *top* is now an *AliasProperty*

New in version 1.9.1.

top is an *AliasProperty* and defaults to the position set in *Config*.

ungrab_mouse()

Ungrab mouse

New in version 1.10.0.

Note: This feature requires the SDL2 window provider.

width

Rotated window width.

width is a read-only *AliasProperty*.

Kivy module for binary dependencies.

Binary dependencies such as gstreamer is installed as a namespace module of kivy.deps. These modules are responsible for making sure that the binaries are available to kivy.

Effects

New in version 1.7.0.

Everything starts with the *KineticEffect*, the base class for computing velocity out of a movement. This base class is used to implement the *ScrollEffect*, a base class used for our *ScrollView* widget effect. We have multiple implementations:

- *ScrollEffect*: base class used for implementing an effect. It only calculates the scrolling and the overscroll.
- *DampedScrollEffect*: uses the overscroll information to allow the user to drag more than expected. Once the user stops the drag, the position is returned to one of the bounds.
- *OpacityScrollEffect*: uses the overscroll information to reduce the opacity of the scrollview widget. When the user stops the drag, the opacity is set back to 1.

23.1 Damped scroll effect

New in version 1.7.0.

This damped scroll effect will use the *overscroll* to calculate the scroll value, and slows going back to the upper or lower limit.

class `kivy.effects.dampedscroll.DampedScrollEffect(**kwargs)`

Bases: *kivy.effects.scroll.ScrollEffect*

DampedScrollEffect class. See the module documentation for more information.

edge_damping

Edge damping.

edge_damping is a *NumericProperty* and defaults to 0.25

min_overscroll

An overscroll less than this amount will be normalized to 0.

New in version 1.8.0.

min_overscroll is a *NumericProperty* and defaults to .5.

round_value

If True, when the motion stops, *value* is rounded to the nearest integer.

New in version 1.8.0.

round_value is a *BooleanProperty* and defaults to True.

spring_constant

Spring constant.

spring_constant is a *NumericProperty* and defaults to 2.0

update_velocity(*dt*)

(internal) Update the velocity according to the framerate and friction.

23.2 Kinetic effect

New in version 1.7.0.

The *KineticEffect* is the base class that is used to compute the velocity out of a movement. When the movement is finished, the effect will compute the position of the movement according to the velocity, and reduce the velocity with a friction. The movement stop until the velocity is 0.

Conceptually, the usage could be:

```
>>> effect = KineticEffect()
>>> effect.start(10)
>>> effect.update(15)
>>> effect.update(30)
>>> effect.stop(48)
```

Over the time, you will start a movement of a value, update it, and stop the movement. At this time, you'll get the movement value into *KineticEffect.value*. On the example i've typed manually, the computed velocity will be:

```
>>> effect.velocity
3.1619100231163046
```

After multiple clock interaction, the velocity will decrease according to *KineticEffect.friction*. The computed value will be stored in *KineticEffect.value*. The output of this *value* could be:

```
46.30038145219605
54.58302451968686
61.9229016256196
# ...
```

```
class kivy.effects.kinetic.KineticEffect(**kwargs)
```

Bases: *kivy.event.EventDispatcher*

Kinetic effect class. See module documentation for more information.

cancel()

Cancel a movement. This can be used in case *stop()* cannot be called. It will reset *is_manual* to False, and compute the movement if the velocity is > 0.

friction

Friction to apply on the velocity

velocity is a *NumericProperty* and defaults to 0.05.

is_manual

Indicate if a movement is in progress (True) or not (False).

velocity is a *BooleanProperty* and defaults to False.

max_history

Save up to *max_history* movement value into the history. This is used for correctly calculating the velocity according to the movement.

max_history is a *NumericProperty* and defaults to 5.

min_distance

The minimal distance for a movement to have nonzero velocity.

New in version 1.8.0.

min_distance is *NumericProperty* and defaults to 0.1.

min_velocity

Velocity below this quantity is normalized to 0. In other words, any motion whose velocity falls below this number is stopped.

New in version 1.8.0.

min_velocity is a *NumericProperty* and defaults to 0.5.

start(*val*, *t=None*)

Start the movement.

Parameters

***val*: float or int** Value of the movement

***t*: float, defaults to None** Time when the movement happen. If no time is set, it will use `time.time()`

stop(*val*, *t=None*)

Stop the movement.

See *start()* for the arguments.

update(*val*, *t=None*)

Update the movement.

See *start()* for the arguments.

update_velocity(*dt*)

(internal) Update the velocity according to the framerate and friction.

value

Value (during the movement and computed) of the effect.

velocity is a *NumericProperty* and defaults to 0.

velocity

Velocity of the movement.

velocity is a *NumericProperty* and defaults to 0.

23.3 Opacity scroll effect

Based on the `DampedScrollEffect`, this one will also decrease the opacity of the target widget during the overscroll.

class kivy.effects.opacityscroll.**OpacityScrollEffect**(**kwargs)

Bases: *kivy.effects.dampedscroll.DampedScrollEffect*

OpacityScrollEffect class. Uses the overscroll information to reduce the opacity of the scrollview widget. When the user stops the drag, the opacity is set back to 1.

23.4 Scroll effect

New in version 1.7.0.

Based on the *kinetic* effect, the *ScrollEffect* will limit the movement to bounds determined by its *min* and *max* properties. If the movement exceeds these bounds, it will calculate the amount of *overscroll* and try to return to the value of one of the bounds.

This is very useful for implementing a scrolling list. We actually use this class as a base effect for our *ScrollView* widget.

class kivy.effects.scroll.**ScrollEffect**(**kwargs)

Bases: *kivy.effects.kinetic.KineticEffect*

ScrollEffect class. See the module documentation for more informations.

displacement

Cumulative distance of the movement during the interaction. This is used to determine if the movement is a drag (more than *drag_threshold*) or not.

displacement is a *NumericProperty* and defaults to 0.

drag_threshold

Minimum distance to travel before the movement is considered as a drag.

velocity is a *NumericProperty* and defaults to 20sp.

max

Maximum boundary to use for scrolling.

max is a *NumericProperty* and defaults to 0.

min

Minimum boundary to use for scrolling.

min is a *NumericProperty* and defaults to 0.

overscroll

Computed value when the user over-scrolls i.e. goes out of the bounds.

overscroll is a *NumericProperty* and defaults to 0.

reset(pos)

(internal) Reset the value and the velocity to the *pos*. Mostly used when the bounds are checked.

scroll

Computed value for scrolling. This value is different from *kivy.effects.kinetic.KineticEffect.value* in that it will return to one of the min/max bounds.

scroll is a *NumericProperty* and defaults to 0.

start(val, t=None)

Start the movement.

Parameters

val: float or int Value of the movement

***t*: float, defaults to None** Time when the movement happen. If no time is set, it will use `time.time()`

stop(*val*, *t=None*)

Stop the movement.

See *start()* for the arguments.

target_widget

Widget to attach to this effect. Even if this class doesn't make changes to the *target_widget* by default, subclasses can use it to change the graphics or apply custom transformations.

target_widget is a *ObjectProperty* and defaults to None.

update(*val*, *t=None*)

Update the movement.

See *start()* for the arguments.

Garden

New in version 1.7.0.

Changed in version 1.11.1.

Garden is a project to centralize addons for Kivy maintained by users. You can find more information at [Kivy Garden](#). All the garden packages are centralized on the [kivy-garden Github](#) repository.

Warning: The garden flower widgets are contributed by regular users such as yourself. The kivy developers do not take any responsibility for the code hosted in the garden organization repositories - we do not actively monitor the flower repos. Please use at your own risk.

24.1 Update to garden structure

Starting with the kivy 1.11.0 release, kivy has [shifted](#) from using the garden legacy tool that installs flowers with *garden install flower* where the flower does not have a proper python package structure to flowers that can be installed with pip and uploaded to pypi. Kivy supports the legacy garden flowers side by side with the newer packages so the garden tool and legacy flowers will be able to be used indefinitely. But we will only provide support for the newer packages format in the future.

For garden package maintainers - for a guide how to migrate your garden package from the legacy structure *garden.flower* to the newer *flower* structure used with the pip, see [this guide](#).

We hope most garden packages will be converted to the newer format to help with installation.

24.2 General Usage Guidelines

To use a kivy garden flower, first check if the flower is in the legacy format. If the flower name is in the format of *garden.flower*, such as [garden.graph](#) it is a legacy flower. If it is just *flower* such as [graph](#) it is in the present format. If it is in the legacy format see [Legacy garden tool instructions](#) for how to install and use it. Otherwise, continue with the guide below.

Garden flowers can now be installed with the *pip* tool like a normal python package. Given a flower that you want to install, lets use **graph** as an example. You can install master directly from github with:

```
python -m pip install https://github.com/kivy-garden/graph/archive/master.zip
```

Look under the repository's releases tab if you'd like to install a specific release or a pre-compiled wheel, if the flower has any. Then use the url with *pip*.

Or you can automatically install it using garden's pypi server with:

```
python -m pip install kivy_garden.graph --extra-index-url https://kivy-garden.
→github.io/simple/
```

To permanently add our garden server to your pip configuration so that you don't have to specify it with *-extra-index-url*, add:

```
[global]
timeout = 60
index-url = https://kivy-garden.github.io/simple/
```

to your **pip.conf**.

If the flower maintainer has uploaded the flower to **pypi**, you can just install it with *pip install kivy_garden.flower*.

24.3 Legacy garden tool instructions

Garden is now distributed as a separate Python module, kivy-garden. You can install it with pip:

```
pip install kivy-garden
```

The garden module does not initially include any packages. You can download them with the garden tool installed by the pip package:

```
# Installing a garden package
garden install graph

# Upgrade a garden package
garden install --upgrade graph

# Uninstall a garden package
garden uninstall graph

# List all the garden packages installed
garden list

# Search new packages
garden search

# Search all the packages that contain "graph"
garden search graph

# Show the help
garden --help
```

All the garden packages are installed by default in *~/kivy/garden*.

Note: In previous versions of Kivy, garden was a tool at `kivy/tools/garden`. This no longer exists, but the `kivy-garden` module provides exactly the same functionality.

24.3.1 Packaging

If you want to include garden packages in your application, you can add `--app` to the *install* command. This will create a *libs/garden* directory in your current directory which will be used by *kivy.garden*.

For example:

```
cd myapp
garden install --app graph
```

```
kivy.garden.garden_system_dir = 'garden'
    system path where garden modules can be installed
```

Graphics

This package assembles many low level functions used for drawing. The whole graphics package is compatible with OpenGL ES 2.0 and has many rendering optimizations.

25.1 The basics

For drawing on a screen, you will need :

1. a *Canvas* object.
2. *Instruction* objects.

Each *Widget* in Kivy already has a *Canvas* by default. When you create a widget, you can create all the instructions needed for drawing. If *self* is your current widget, you can do:

```
from kivy.graphics import *
with self.canvas:
    # Add a red color
    Color(1., 0, 0)

    # Add a rectangle
    Rectangle(pos=(10, 10), size=(500, 500))
```

The instructions *Color* and *Rectangle* are automatically added to the canvas object and will be used when the window is drawn.

Note: Kivy drawing instructions are not automatically relative to the position or size of the widget. You, therefore, need to consider these factors when drawing. In order to make your drawing instructions relative to the widget, the instructions need either to be declared in the *KvLang* or bound to pos and size changes. Please see *Adding a Background to a Layout* for more detail.

25.2 GL Reloading mechanism

New in version 1.2.0.

During the lifetime of the application, the OpenGL context might be lost. This happens:

- when the window is resized on OS X or the Windows platform and you're using pygame as a window provider. This is due to SDL 1.2. In the SDL 1.2 design, it needs to recreate a GL context everytime the window is resized. This was fixed in SDL 1.3 but pygame is not yet available on it by default.
- when Android releases the app resources: when your application goes to the background, Android might reclaim your opengl context to give the resource to another app. When the user switches back to your application, a newly created gl context is given to your app.

Starting from 1.2.0, we have introduced a mechanism for reloading all the graphics resources using the GPU: Canvas, FBO, Shader, Texture, VBO, and VertexBatch:

- VBO and VertexBatch are constructed by our graphics instructions. We have all the data needed to reconstruct when reloading.
- Shader: same as VBO, we store the source and values used in the shader so we are able to recreate the vertex/fragment/program.
- Texture: if the texture has a source (an image file or atlas), the image is reloaded from the source and reuploaded to the GPU.

You should cover these cases yourself:

- Textures without a source: if you manually created a texture and manually blit data / a buffer to it, you must handle the reloading yourself. Check the *Texture* to learn how to manage that case. (The text rendering already generates the texture and handles the reloading. You don't need to reload text yourself.)
- FBO: if you added / removed / drew things multiple times on the FBO, we can't reload it. We don't keep a history of the instructions put on it. As for textures without a source, check the *Framebuffer* to learn how to manage that case.

class `kivy.graphics.Bezier(**kwargs)`
Bases: `kivy.graphics.instructions.VertexInstruction`

New in version 1.0.8.

Parameters

points: list List of points in the format (x1, y1, x2, y2...)

segments: int, defaults to 180 Define how many segments are needed for drawing the curve. The drawing will be smoother if you have many segments.

loop: bool, defaults to False Set the bezier curve to join the last point to the first.

dash_length: int Length of a segment (if dashed), defaults to 1.

dash_offset: int Distance between the end of a segment and the start of the next one, defaults to 0. Changing this makes it dashed.

dash_length

Property for getting/setting the length of the dashes in the curve.

dash_offset

Property for getting/setting the offset between the dashes in the curve.

points

Property for getting/settings the points of the triangle.

Warning: This will always reconstruct the whole graphic from the new points list. It can be very CPU intensive.

segments

Property for getting/setting the number of segments of the curve.

class `kivy.graphics.BindTexture(**kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

The BindTexture Instruction will bind a texture and enable GL_TEXTURE_2D for subsequent drawing.

Parameters

texture: `Texture` Specifies the texture to bind to the given index.

source

Set/get the source (filename) to load for the texture.

class `kivy.graphics.BorderImage(**kwargs)`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

concept of a CSS3 border-image.

Parameters

border: `list` Border information in the format (bottom, right, top, left). Each value is in pixels.

auto_scale: `string` New in version 1.9.1.

Changed in version 1.9.2: This used to be a bool and has been changed to be a string state.

Can be one of 'off', 'both', 'x_only', 'y_only', 'y_full_x_lower', 'x_full_y_lower', 'both_lower'.

Autoscale controls the behavior of the 9-slice.

By default the border values are preserved exactly, meaning that if the total size of the object is smaller than the border values you will have some 'rendering errors' where your texture appears inside out. This also makes it impossible to achieve a rounded button that scales larger than the size of its source texture. The various options for auto_scale will let you achieve some mixes of the 2 types of rendering.

'off': is the default and behaves as BorderImage did when auto_scale was False before.

'both': Scales both x and y dimension borders according to the size of the BorderImage, this disables the BorderImage making it render the same as a regular Image.

'x_only': The Y dimension functions as the default, and the X scales to the size of the BorderImage's width.

'y_only': The X dimension functions as the default, and the Y scales to the size of the BorderImage's height.

'y_full_x_lower': Y scales as in 'y_only', Y scales if the size of the scaled version would be smaller than the provided border only.

'x_full_y_lower': X scales as in 'x_only', Y scales if the size of the scaled version would be smaller than the provided border only.

'both_lower': This is what `auto_scale` did when it was True in 1.9.1 Both X and Y dimensions will be scaled if the `BorderImage` is smaller than the source.

If the `BorderImage`'s size is less than the sum of its borders, horizontally or vertically, and this property is set to True, the borders will be rescaled to accommodate for the smaller size.

auto_scale

Property for setting if the corners are automatically scaled when the `BorderImage` is too small.

border

Property for getting/setting the border of the class.

display_border

Property for getting/setting the border display size.

class `kivy.graphics.Callback`(*callback=None, **kwargs*)

Bases: `kivy.graphics.instructions.Instruction`

A Callback is an instruction that will be called when the drawing operation is performed. When adding instructions to a canvas, you can do this:

```
with self.canvas:
    Color(1, 1, 1)
    Rectangle(pos=self.pos, size=self.size)
    Callback(self.my_callback)
```

The definition of the callback must be:

```
def my_callback(self, instr):
    print('I have been called!')
```

Warning: Note that if you perform many and/or costly calls to callbacks, you might potentially slow down the rendering performance significantly.

The updating of your canvas does not occur until something new happens. From your callback, you can ask for an update:

```
with self.canvas:
    self.cb = Callback(self.my_callback)
# then later in the code
self.cb.ask_update()
```

If you use the `Callback` class to call rendering methods of another toolkit, you will have issues with the OpenGL context. The OpenGL state may have been manipulated by the other toolkit, and as soon as program flow returns to Kivy, it will just break. You can have glitches, crashes, black holes might occur, etc. To avoid that, you can activate the `reset_context` option. It will reset the OpenGL context state to make Kivy's rendering correct after the call to your callback.

Warning: The `reset_context` is not a full OpenGL reset. If you have issues regarding that, please contact us.

ask_update()

Inform the parent canvas that we'd like it to update on the next frame. This is useful when you need to trigger a redraw due to some value having changed for example.

New in version 1.0.4.

callback

Property for getting/setting func.

reset_context

Set this to True if you want to reset the OpenGL context for Kivy after the callback has been called.

class kivy.graphics.Canvas(**kwargs)

Bases: *kivy.graphics.instructions.CanvasBase*

instructions that you want to be used for drawing.

Note: The Canvas supports Python's `with` statement and its enter & exit semantics.

Usage of a canvas without the `with` statement:

```
self.canvas.add(Color(1., 1., 0))
self.canvas.add(Rectangle(size=(50, 50)))
```

Usage of a canvas with Python's `with` statement:

```
with self.canvas:
    Color(1., 1., 0)
    Rectangle(size=(50, 50))
```

add(*Instruction c*)

after

Property for getting the 'after' group.

ask_update()

Inform the canvas that we'd like it to update on the next frame. This is useful when you need to trigger a redraw due to some value having changed for example.

before

Property for getting the 'before' group.

clear()

Clears every *Instruction* in the canvas, leaving it clean.

draw()

Apply the instruction to our window.

has_after

Property to see if the *after* group has already been created.

New in version 1.7.0.

has_before

Property to see if the *before* group has already been created.

New in version 1.7.0.

opacity

Property to get/set the opacity value of the canvas.

New in version 1.4.1.

The opacity attribute controls the opacity of the canvas and its children. Be careful, it's a cumulative attribute: the value is multiplied to the current global opacity and the result is applied to the current context color.

For example: if your parent has an opacity of 0.5 and a child has an opacity of 0.2, the real opacity of the child will be $0.5 * 0.2 = 0.1$.

Then, the opacity is applied on the shader as:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

remove(*Instruction c*)

class kivy.graphics.CanvasBase

Bases: *kivy.graphics.instructions.InstructionGroup*

CanvasBase provides the context manager methods for the *Canvas*.

class kivy.graphics.Color(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

drawn after it.

This represents a color between 0 and 1, but is applied as a *multiplier* to the texture of any vertex instructions following it in a canvas. If no texture is set, the vertex instruction takes the precise color of the Color instruction.

For instance, if a Rectangle has a texture with uniform color (0.5, 0.5, 0.5, 1.0) and the preceding Color has *rgba*=(1, 0.5, 2, 1), the actual visible color will be (0.5, 0.25, 1.0, 1.0) since the Color instruction is applied as a multiplier to every *rgba* component. In this case, a Color component outside the 0-1 range gives a visible result as the intensity of the blue component is doubled.

To declare a Color in Python, you can do:

```
from kivy.graphics import Color

# create red v
c = Color(1, 0, 0)
# create blue color
c = Color(0, 1, 0)
# create blue color with 50% alpha
c = Color(0, 1, 0, .5)

# using hsv mode
c = Color(0, 1, 1, mode='hsv')
# using hsv mode + alpha
c = Color(0, 1, 1, .2, mode='hsv')
```

You can also set color components that are available as properties by passing them as keyword arguments:

```
c = Color(b=0.5) # sets the blue component only
```

In kv lang you can set the color properties directly:

```
<Rule>:
    canvas:
        # red color
        Color:
            rgb: 1, 0, 0
        # blue color
        Color:
            rgb: 0, 1, 0
        # blue color with 50% alpha
        Color:
```

(continues on next page)

```

    rgba: 0, 1, 0, .5

    # using hsv mode
    Color:
        hsv: 0, 1, 1
    # using hsv mode + alpha
    Color:
        hsv: 0, 1, 1
        a: .5

```

a

Alpha component, between 0 and 1.

b

Blue component, between 0 and 1.

g

Green component, between 0 and 1.

h

Hue component, between 0 and 1.

hsv

HSV color, list of 3 values in 0-1 range, alpha will be 1.

r

Red component, between 0 and 1.

rgb

RGB color, list of 3 values in 0-1 range. The alpha will be 1.

rgba

RGBA color, list of 4 values in 0-1 range.

s

Saturation component, between 0 and 1.

v

Value component, between 0 and 1.

class `kivy.graphics.ContextInstruction(**kwargs)`Bases: `kivy.graphics.instructions.Instruction`

that don't have a direct visual representation, but instead modify the current Canvas' state, e.g. texture binding, setting color parameters, matrix manipulation and so on.

class `kivy.graphics.Ellipse(*args, **kwargs)`Bases: `kivy.graphics.vertex_instructions.Rectangle`

Changed in version 1.0.7: Added `angle_start` and `angle_end`.

Parameters

segments: int, defaults to 180 Define how many segments are needed for drawing the ellipse. The drawing will be smoother if you have many segments.

angle_start: int, defaults to 0 Specifies the starting angle, in degrees, of the disk portion.

angle_end: int, defaults to 360 Specifies the ending angle, in degrees, of the disk portion.

angle_end

End angle of the ellipse in degrees, defaults to 360.

angle_start

Start angle of the ellipse in degrees, defaults to 0.

segments

Property for getting/setting the number of segments of the ellipse.

class `kivy.graphics.Fbo(*args, **kwargs)`

Bases: `kivy.graphics.instructions.RenderContext`

“with” statement.

Parameters

clear_color: tuple, defaults to (0, 0, 0, 0) Define the default color for clearing the framebuffer

size: tuple, defaults to (1024, 1024) Default size of the framebuffer

push_viewport: bool, defaults to True If True, the OpenGL viewport will be set to the framebuffer size, and will be automatically restored when the framebuffer released.

with_depthbuffer: bool, defaults to False If True, the framebuffer will be allocated with a Z buffer.

with_stencilbuffer: bool, defaults to False New in version 1.9.0.

If True, the framebuffer will be allocated with a stencil buffer.

texture: `Texture`, defaults to None If None, a default texture will be created.

Note: Using both of `with_stencilbuffer` and `with_depthbuffer` is not supported in kivy 1.9.0

add_reload_observer(callback)

Add a callback to be called after the whole graphics context has been reloaded. This is where you can reupload your custom data in GPU.

New in version 1.2.0.

Parameters

callback: func(context) -> return None The first parameter will be the context itself

bind()

Bind the FBO to the current opengl context. *Bind* mean that you enable the Framebuffer, and all the drawing operations will act inside the Framebuffer, until `release()` is called.

The bind/release operations are automatically called when you add graphics objects into it. If you want to manipulate a Framebuffer yourself, you can use it like this:

```
self.fbo = FBO()
self.fbo.bind()
# do any drawing command
self.fbo.release()

# then, your fbo texture is available at
print(self.fbo.texture)
```

clear_buffer()

Clear the framebuffer with the `clear_color`.

You need to bind the framebuffer yourself before calling this method:

```
fbo.bind()
fbo.clear_buffer()
fbo.release()
```

clear_color

Clear color in (red, green, blue, alpha) format.

get_pixel_color(*int wx, int wy*)

Get the color of the pixel with specified window coordinates wx, wy. It returns result in RGBA format.

New in version 1.8.0.

pixels

Get the pixels texture, in RGBA format only, unsigned byte. The origin of the image is at bottom left.

New in version 1.7.0.

release()

Release the Framebuffer (unbind).

remove_reload_observer(*callback*)

Remove a callback from the observer list, previously added by [add_reload_observer\(\)](#).

New in version 1.2.0.

size

Size of the framebuffer, in (width, height) format.

If you change the size, the framebuffer content will be lost.

texture

Return the framebuffer texture

exception `kivy.graphics.GraphicsException`

Bases: `Exception`

Exception raised when a graphics error is fired.

class `kivy.graphics.Instruction`(***kwargs*)

Bases: [kivy.event.ObjectWithUid](#)

usage only, don't use it directly.

flag_update(*int do_parent=1*)

group

group: unicode

proxy_ref

Return a proxy reference to the Instruction i.e. without creating a reference of the widget. See [weakref.proxy](#) for more information.

New in version 1.7.2.

class `kivy.graphics.InstructionGroup`(***kwargs*)

Bases: [kivy.graphics.instructions.Instruction](#)

removing of graphics instructions. It can be used directly as follows:

```
blue = InstructionGroup() blue.add(Color(0, 0, 1, 0.2)) blue.add(Rectangle(pos=self.pos, size=(100, 100)))
```

```
green = InstructionGroup() green.add(Color(0, 1, 0, 0.4)) green.add(Rectangle(pos=(100, 100), size=(100, 100)))
```

```
# Here, self should be a Widget or subclass [self.canvas.add(group) for group in [blue, green]]
add(Instruction c)
    Add a new Instruction to our list.

children
    children: list

clear()
    Remove all the Instructions.

get_group(unicode groupname)
    Return an iterable for all the Instructions with a specific group name.

indexof(Instruction c)

insert(int index, Instruction c)
    Insert a new Instruction into our list at index.

length()

remove(Instruction c)
    Remove an existing Instruction from our list.

remove_group(unicode groupname)
    Remove all Instructions with a specific group name.

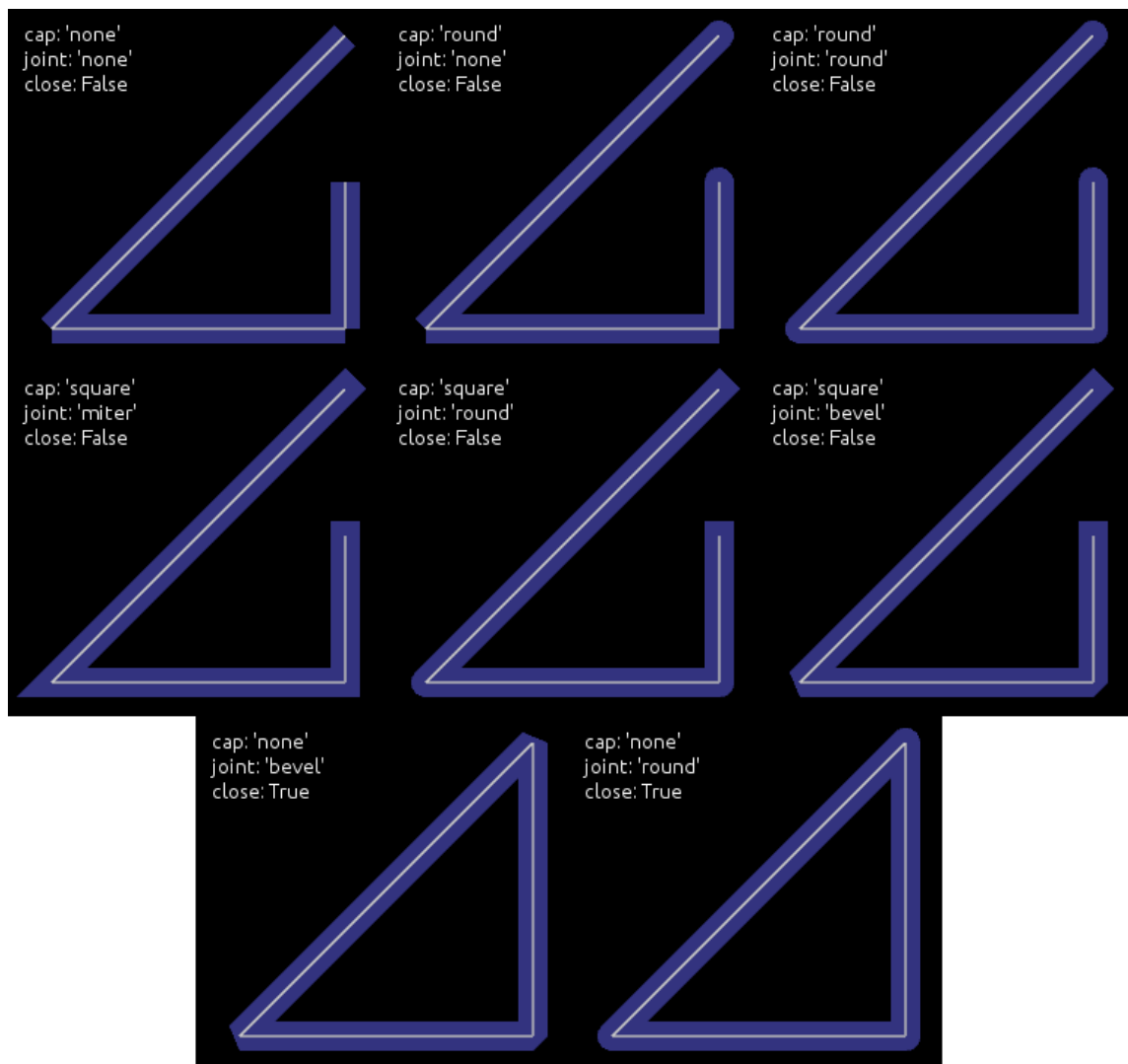
class kivy.graphics.Line(**kwargs)
    Bases: kivy.graphics.instructions.VertexInstruction

    Drawing a line can be done easily:
```

```
with self.canvas:
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

The line has 3 internal drawing modes that you should be aware of for optimal results:

1. If the *width* is 1.0, then the standard GL_LINE drawing from OpenGL will be used. *dash_length*, *dash_offset*, and *dashes* will work, while properties for cap and joint have no meaning here.
2. If the *width* is greater than 1.0, then a custom drawing method, based on triangulation, will be used. *dash_length*, *dash_offset*, and *dashes* do not work in this mode. Additionally, if the current color has an alpha less than 1.0, a stencil will be used internally to draw the line.



Parameters

points: list List of points in the format (x1, y1, x2, y2...)

dash_length: int Length of a segment (if dashed), defaults to 1.

dash_offset: int Offset between the end of a segment and the beginning of the next one, defaults to 0. Changing this makes it dashed.

dashes: list of ints List of [ON length, offset, ON length, offset, ...]. E.g. [2, 4, 1, 6, 8, 2] would create a line with the first dash length 2 then an offset of 4 then a dash length of 1 then an offset of 6 and so on. Defaults to []. Changing this makes it dashed and overrides `dash_length` and `dash_offset`.

width: float Width of the line, defaults to 1.0.

cap: str, defaults to 'round' See [cap](#) for more information.

joint: str, defaults to 'round' See [joint](#) for more information.

cap_precision: int, defaults to 10 See [cap_precision](#) for more information

joint_precision: int, defaults to 10 See [joint_precision](#) for more information
See [cap_precision](#) for more information.

joint_precision: int, defaults to 10 See [joint_precision](#) for more information.

close: bool, defaults to False If True, the line will be closed.

circle: list If set, the *points* will be set to build a circle. See *circle* for more information.

ellipse: list If set, the *points* will be set to build an ellipse. See *ellipse* for more information.

rectangle: list If set, the *points* will be set to build a rectangle. See *rectangle* for more information.

bezier: list If set, the *points* will be set to build a bezier line. See *bezier* for more information.

bezier_precision: int, defaults to 180 Precision of the Bezier drawing.

Changed in version 1.0.8: *dash_offset* and *dash_length* have been added.

Changed in version 1.4.1: *width*, *cap*, *joint*, *cap_precision*, *joint_precision*, *close*, *ellipse*, *rectangle* have been added.

Changed in version 1.4.1: *bezier*, *bezier_precision* have been added.

Changed in version 1.11.0: *dashes* have been added

bezier

Use this property to build a bezier line, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of 2n elements, n being the number of points.

Usage:

```
Line(bezier=(x1, y1, x2, y2, x3, y3))
```

New in version 1.4.2.

Note: Bezier lines calculations are inexpensive for a low number of points, but complexity is quadratic, so lines with a lot of points can be very expensive to build, use with care!

bezier_precision

Number of iteration for drawing the bezier between 2 segments, defaults to 180. The *bezier_precision* must be at least 1.

New in version 1.4.2.

cap

Determine the cap of the line, defaults to 'round'. Can be one of 'none', 'square' or 'round'

New in version 1.4.1.

cap_precision

Number of iteration for drawing the "round" cap, defaults to 10. The *cap_precision* must be at least 1.

New in version 1.4.1.

circle

Use this property to build a circle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (center_x, center_y, radius, angle_start, angle_end, segments):

- center_x and center_y represent the center of the circle
- radius represent the radius of the circle

- (optional) `angle_start` and `angle_end` are in degree. The default value is 0 and 360.
- (optional) `segments` is the precision of the ellipse. The default value is calculated from the range between angle.

Note that it's up to you to *close* the circle or not.

For example, for building a simple ellipse, in python:

```
# simple circle
Line(circle=(150, 150, 50))

# only from 90 to 180 degrees
Line(circle=(150, 150, 50, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(circle=(150, 150, 50, 90, 180, 20))
```

New in version 1.4.1.

close

If True, the line will be closed.

New in version 1.4.1.

dash_length

Property for getting/setting the length of the dashes in the curve

New in version 1.0.8.

dash_offset

Property for getting/setting the offset between the dashes in the curve

New in version 1.0.8.

dashes

Property for getting/setting dashes.

List of [ON length, offset, ON length, offset, ...]. E.g. [2, 4, 1, 6, 8, 2] would create a line with the first dash length 2 then an offset of 4 then a dash length of 1 then an offset of 6 and so on.

New in version 1.11.0.

ellipse

Use this property to build an ellipse, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (x, y, width, height, angle_start, angle_end, segments):

- x and y represent the bottom left of the ellipse
- width and height represent the size of the ellipse
- (optional) `angle_start` and `angle_end` are in degree. The default value is 0 and 360.
- (optional) `segments` is the precision of the ellipse. The default value is calculated from the range between angle.

Note that it's up to you to *close* the ellipse or not.

For example, for building a simple ellipse, in python:

```
# simple ellipse
Line(ellipse=(0, 0, 150, 150))
```

(continues on next page)

(continued from previous page)

```
# only from 90 to 180 degrees
Line(ellipse=(0, 0, 150, 150, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(ellipse=(0, 0, 150, 150, 90, 180, 20))
```

New in version 1.4.1.

joint

Determine the join of the line, defaults to 'round'. Can be one of 'none', 'round', 'bevel', 'miter'.

New in version 1.4.1.

joint_precision

Number of iteration for drawing the "round" joint, defaults to 10. The joint_precision must be at least 1.

New in version 1.4.1.

points

Property for getting/settings points of the line

Warning: This will always reconstruct the whole graphics from the new points list. It can be very CPU expensive.

rectangle

Use this property to build a rectangle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (x, y, width, height):

- x and y represent the bottom-left position of the rectangle
- width and height represent the size

The line is automatically closed.

Usage:

```
Line(rectangle=(0, 0, 200, 200))
```

New in version 1.4.1.

rounded_rectangle

Use this property to build a rectangle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of one of the following forms:

- (x, y, width, height, corner_radius)
- (x, y, width, height, corner_radius, resolution)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4, resolution)
- x and y represent the bottom-left position of the rectangle
- width and height represent the size

- `corner_radius` is the number of pixels between two borders and the center of the circle arc joining them
- `resolution` is the number of line segment that will be used to draw the circle arc at each corner (defaults to 30)

The line is automatically closed.

Usage:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40, 100))
```

New in version 1.9.0.

width

Determine the width of the line, defaults to 1.0.

New in version 1.4.1.

class `kivy.graphics.SmoothLine(**kwargs)`

Bases: `kivy.graphics.vertex_instructions.Line`

results. It has few drawbacks:

- drawing a line with `alpha` will probably not have the intended result if the line crosses itself.
- `cap`, `joint` and `dash` properties are not supported.
- it uses a custom texture with a premultiplied alpha.
- lines under 1px in width are not supported: they will look the same.

Warning: This is an unfinished work, experimental, and subject to crashes.

New in version 1.9.0.

overdraw_width

Determine the overdraw width of the line, defaults to 1.2.

premultiplied_texture()

class `kivy.graphics.MatrixInstruction(*args, **kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

matrix

Matrix property. Matrix from the transformation module. Setting the matrix using this property when a change is made is important because it will notify the context about the update.

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class `kivy.graphics.Mesh(**kwargs)`

Bases: `kivy.graphics.instructions.VertexInstruction`

In OpenGL ES 2.0 and in our graphics implementation, you cannot have more than 65535 indices.

A list of vertices is described as:

```
vertices = [x1, y1, u1, v1, x2, y2, u2, v2, ...]
            |       |       |       |
            +----- i1 -----+ +----- i2 -----+
```

If you want to draw a triangle, add 3 vertices. You can then make an indices list as follows:

```
indices = [0, 1, 2]
```

New in version 1.1.0.

Parameters

vertices: iterable List of vertices in the format (x1, y1, u1, v1, x2, y2, u2, v2...).

indices: iterable List of indices in the format (i1, i2, i3...).

mode: str Mode of the vbo. Check *mode* for more information. Defaults to 'points'.

fnt: list The format for vertices, by default, each vertex is described by 2D coordinates (x, y) and 2D texture coordinate (u, v). Each element of the list should be a tuple or list, of the form

```
(variable_name, size, type)
```

which will allow mapping vertex data to the glsl instructions.

```
[(b'v_pos', 2, 'float'), (b'v_tc', 2, 'float'),]
```

will allow using

```
attribute vec2 v_pos; attribute vec2 v_tc;
```

in glsl's vertex shader.

Changed in version 1.8.1: Before, *vertices* and *indices* would always be converted to a list, now, they are only converted to a list if they do not implement the buffer interface. So e.g. numpy arrays, python arrays etc. are used in place, without creating any additional copies. However, the buffers cannot be readonly (even though they are not changed, due to a cython limitation) and must be contiguous in memory.

Note: When passing a memoryview or a instance that implements the buffer interface, *vertices* should be a buffer of floats ('f' code in python array) and *indices* should be a buffer of unsigned short ('H' code in python array). Arrays in other formats will still have to be converted internally, negating any potential gain.

indices

Vertex indices used to specify the order when drawing the mesh.

mode

VBO Mode used for drawing vertices/indices. Can be one of 'points', 'line_strip', 'line_loop', 'lines', 'triangles', 'triangle_strip' or 'triangle_fan'.

vertices

List of x, y, u, v coordinates used to construct the Mesh. Right now, the Mesh instruction doesn't allow you to change the format of the vertices, which means it's only x, y + one texture coordinate.

```
class kivy.graphics.Point(**kwargs)
```

Bases: *kivy.graphics.instructions.VertexInstruction*

width/height of 2 times the *pointsize*.

Parameters

points: list List of points in the format (x1, y1, x2, y2...), where each pair of coordinates specifies the center of a new point.

pointsize: float, defaults to 1. The size of the point, measured from the center to the edge. A value of 1.0 therefore means the real size will be 2.0 x 2.0.

Warning: Starting from version 1.0.7, vertex instruction have a limit of 65535 vertices (indices of vertex to be accurate). 2 entries in the list (x, y) will be converted to 4 vertices. So the limit inside Point() class is 2^{15-2} .

add_point(*float x, float y*)

Add a point to the current *points* list.

If you intend to add multiple points, prefer to use this method instead of reassigning a new *points* list. Assigning a new *points* list will recalculate and reupload the whole buffer into the GPU. If you use add_point, it will only upload the changes.

points

Property for getting/settings the center points in the points list. Each pair of coordinates specifies the center of a new point.

pointsize

Property for getting/setting point size. The size is measured from the center to the edge, so a value of 1.0 means the real size will be 2.0 x 2.0.

class kivy.graphics.PopMatrix(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class kivy.graphics.PushMatrix(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class kivy.graphics.Quad(**kwargs)

Bases: *kivy.graphics.instructions.VertexInstruction*

Parameters

points: list List of point in the format (x1, y1, x2, y2, x3, y3, x4, y4).

points

Property for getting/settings points of the quad.

class kivy.graphics.Rectangle(**kwargs)

Bases: *kivy.graphics.instructions.VertexInstruction*

Parameters

pos: list Position of the rectangle, in the format (x, y).

size: list Size of the rectangle, in the format (width, height).

pos

Property for getting/settings the position of the rectangle.

size

Property for getting/settings the size of the rectangle.

class kivy.graphics.RenderContext(*args, **kwargs)

Bases: *kivy.graphics.instructions.Canvas*

- The vertex shader
- The fragment shader
- The default texture
- The state stack (color, texture, matrix...)

shader

Return the shader attached to the render context.

use_parent_frag_modelview

If True, the parent fragment modelview matrix will be used.

New in version 1.10.1: `rc = RenderContext(use_parent_frag_modelview=True)`

use_parent_modelview

If True, the parent modelview matrix will be used.

New in version 1.7.0.

Before:

```
rc['modelview_mat'] = Window.render_context['modelview_mat']
```

Now:

```
rc = RenderContext(use_parent_modelview=True)
```

use_parent_projection

If True, the parent projection matrix will be used.

New in version 1.7.0.

Before:

```
rc['projection_mat'] = Window.render_context['projection_mat']
```

Now:

```
rc = RenderContext(use_parent_projection=True)
```

class kivy.graphics.Rotate(*args, **kwargs)

Bases: `kivy.graphics.context_instructions.Transform`

on the modelview matrix. You can set the properties of the instructions afterwards with e.g.

```
rot.angle = 90
rot.axis = (0, 0, 1)
```

angle

Property for getting/setting the angle of the rotation.

axis

Property for getting/setting the axis of the rotation.

The format of the axis is (x, y, z).

origin

Origin of the rotation.

New in version 1.7.0.

The format of the origin can be either (x, y) or (x, y, z).

set(float angle, float ax, float ay, float az)

Set the angle and axis of rotation.

```
>>> rotationobject.set(90, 0, 0, 1)
```

Deprecated since version 1.7.0: The set() method doesn't use the new *origin* property.

class kivy.graphics.**Scale**(*args, **kwargs)

Bases: kivy.graphics.context_instructions.Transform

Create using one or three arguments:

```
Scale(s)          # scale all three axes the same
Scale(x, y, z)     # scale the axes independently
```

Deprecated since version 1.6.0: Deprecated single scale property in favor of x, y, z, xyz axis independent scaled factors.

origin

Origin of the scale.

New in version 1.9.0.

The format of the origin can be either (x, y) or (x, y, z).

scale

Property for getting/setting the scale.

Deprecated since version 1.6.0: Deprecated in favor of per axis scale properties x,y,z, xyz, etc.

x

Property for getting/setting the scale on the X axis.

Changed in version 1.6.0.

xyz

3 tuple scale vector in 3D in x, y, and z axis.

Changed in version 1.6.0.

y

Property for getting/setting the scale on the Y axis.

Changed in version 1.6.0.

z

Property for getting/setting the scale on Z axis.

Changed in version 1.6.0.

class kivy.graphics.**StencilPop**

Bases: *kivy.graphics.instructions.Instruction*

Pop the stencil stack. See the module documentation for more information.

class kivy.graphics.**StencilPush**

Bases: *kivy.graphics.instructions.Instruction*

Push the stencil stack. See the module documentation for more information.

class kivy.graphics.**StencilUse**(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

more information.

func_op

Determine the stencil operation to use for glStencilFunc(). Can be one of 'never', 'less', 'equal', 'lequal', 'greater', 'notequal', 'gequal' or 'always'.

By default, the operator is set to 'equal'.

New in version 1.5.0.

class kivy.graphics.StencilUnUse

Bases: *kivy.graphics.instructions.Instruction*

Use current stencil buffer to unset the mask.

class kivy.graphics.Translate(*args, **kwargs)

Bases: kivy.graphics.context_instructions.Transform

Construct by either:

```
Translate(x, y)           # translate in just the two axes
Translate(x, y, z)        # translate in all three axes
```

x

Property for getting/setting the translation on the X axis.

xy

2 tuple with translation vector in 2D for x and y axis.

xyz

3 tuple translation vector in 3D in x, y, and z axis.

y

Property for getting/setting the translation on the Y axis.

z

Property for getting/setting the translation on the Z axis.

class kivy.graphics.Triangle(**kwargs)

Bases: *kivy.graphics.instructions.VertexInstruction*

Parameters

points: list List of points in the format (x1, y1, x2, y2, x3, y3).

points

Property for getting/settings points of the triangle.

class kivy.graphics.VertexInstruction(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

that have a direct visual representation on the canvas, such as Rectangles, Triangles, Lines, Ellipse and so on.

source

This property represents the filename to load the texture from. If you want to use an image as source, do it like this:

```
with self.canvas:
    Rectangle(source='mylogo.png', pos=self.pos, size=self.size)
```

Here's the equivalent in Kivy language:

```
<MyWidget>:
    canvas:
        Rectangle:
            source: 'mylogo.png'
            pos: self.pos
            size: self.size
```

Note: The filename will be searched for using the `kivy.resources.resource_find()` function.

tex_coords

This property represents the texture coordinates used for drawing the vertex instruction. The value must be a list of 8 values.

A texture coordinate has a position (u, v), and a size (w, h). The size can be negative, and would represent the 'flipped' texture. By default, the tex_coords are:

```
[u, v, u + w, v, u + w, v + h, u, v + h]
```

You can pass your own texture coordinates if you want to achieve fancy effects.

Warning: The default values just mentioned can be negative. Depending on the image and label providers, the coordinates are flipped vertically because of the order in which the image is internally stored. Instead of flipping the image data, we are just flipping the texture coordinates to be faster.

texture

Property that represents the texture used for drawing this Instruction. You can set a new texture like this:

```
from kivy.core.image import Image

texture = Image('logo.png').texture
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=self.size)
```

Usually, you will use the `source` attribute instead of the texture.

class kivy.graphics.**ClearColor**(r, g, b, a, **kwargs)

Bases: `kivy.graphics.instructions.Instruction`

New in version 1.3.0.

Sets the clear color used to clear buffers with the glClear function or `ClearBuffers` graphics instructions.

a

Alpha component, between 0 and 1.

b

Blue component, between 0 and 1.

g

Green component, between 0 and 1.

r

Red component, between 0 and 1.

rgb

RGB color, a list of 3 values in 0-1 range where alpha will be 1.

rgba

RGBA color used for the clear color, a list of 4 values in the 0-1 range.

class kivy.graphics.**ClearBuffers**(*args, **kwargs)

Bases: `kivy.graphics.instructions.Instruction`

New in version 1.3.0.

Clear the buffers specified by the instructions buffer mask property. By default, only the coloc buffer is cleared.

clear_color

If True, the color buffer will be cleared.

clear_depth

If True, the depth buffer will be cleared.

clear_stencil

If True, the stencil buffer will be cleared.

`kivy.graphics.gl_init_resources()`

class `kivy.graphics.PushState(*args, **kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

state stack.

New in version 1.6.0.

class `kivy.graphics.ChangeState(**kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

current render context.

New in version 1.6.0.

class `kivy.graphics.PopState(*args, **kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

state stack.

New in version 1.6.0.

class `kivy.graphics.ApplyContextMatrix(**kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

target_stack by the matrix at the top of the 'source_stack'

New in version 1.6.0.

source_stack

Name of the matrix stack to use as a source. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

target_stack

Name of the matrix stack to use as a target. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class `kivy.graphics.UpdateNormalMatrix`

Bases: `kivy.graphics.instructions.ContextInstruction`

Update the normal matrix 'normal_mat' based on the current modelview matrix. This will compute 'normal_mat' uniform as: $\text{inverse}(\text{transpose}(\text{mat3}(\text{mvm})))$

New in version 1.6.0.

class `kivy.graphics.LoadIdentity(**kwargs)`

Bases: `kivy.graphics.instructions.ContextInstruction`

the instructions stack property (default='modelview_mat')

New in version 1.6.0.

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

25.3 Canvas

The *Canvas* is the root object used for drawing by a *Widget*. Check the class documentation for more information about the usage of Canvas.

class kivy.graphics.instructions.**Instruction**(**kwargs)

Bases: *kivy.event.ObjectWithUid*

usage only, don't use it directly.

flag_update(int do_parent=1)

group

group: unicode

proxy_ref

Return a proxy reference to the Instruction i.e. without creating a reference of the widget. See *weakref.proxy* for more information.

New in version 1.7.2.

class kivy.graphics.instructions.**InstructionGroup**(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

removing of graphics instructions. It can be used directly as follows:

```
blue = InstructionGroup() blue.add(Color(0, 0, 1, 0.2)) blue.add(Rectangle(pos=self.pos, size=(100, 100)))
```

```
green = InstructionGroup() green.add(Color(0, 1, 0, 0.4)) green.add(Rectangle(pos=(100, 100), size=(100, 100)))
```

Here, self should be a Widget or subclass [self.canvas.add(group) for group in [blue, green]]

add(*Instruction c*)

Add a new *Instruction* to our list.

children

children: list

clear()

Remove all the *Instructions*.

get_group(*unicode groupname*)

Return an iterable for all the *Instructions* with a specific group name.

indexof(*Instruction c*)

insert(*int index, Instruction c*)

Insert a new *Instruction* into our list at index.

length()

remove(*Instruction c*)

Remove an existing *Instruction* from our list.

remove_group(*unicode groupname*)

Remove all *Instructions* with a specific group name.

class kivy.graphics.instructions.**ContextInstruction**(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

that don't have a direct visual representation, but instead modify the current Canvas' state, e.g. texture binding, setting color parameters, matrix manipulation and so on.

class kivy.graphics.instructions.**VertexInstruction**(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

that have a direct visual representation on the canvas, such as Rectangles, Triangles, Lines, Ellipse and so on.

source

This property represents the filename to load the texture from. If you want to use an image as source, do it like this:

```
with self.canvas:
    Rectangle(source='mylogo.png', pos=self.pos, size=self.size)
```

Here's the equivalent in Kivy language:

```
<MyWidget>:
    canvas:
        Rectangle:
            source: 'mylogo.png'
            pos: self.pos
            size: self.size
```

Note: The filename will be searched for using the *kivy.resources.resource_find()* function.

tex_coords

This property represents the texture coordinates used for drawing the vertex instruction. The value must be a list of 8 values.

A texture coordinate has a position (u, v), and a size (w, h). The size can be negative, and would represent the 'flipped' texture. By default, the tex_coords are:

```
[u, v, u + w, v, u + w, v + h, u, v + h]
```

You can pass your own texture coordinates if you want to achieve fancy effects.

Warning: The default values just mentioned can be negative. Depending on the image and label providers, the coordinates are flipped vertically because of the order in which the image is internally stored. Instead of flipping the image data, we are just flipping the texture coordinates to be faster.

texture

Property that represents the texture used for drawing this Instruction. You can set a new texture like this:

```
from kivy.core.image import Image

texture = Image('logo.png').texture
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=self.size)
```

Usually, you will use the *source* attribute instead of the texture.

class kivy.graphics.instructions.**Canvas**(**kwargs)

Bases: *kivy.graphics.instructions.CanvasBase*

instructions that you want to be used for drawing.

Note: The Canvas supports Python's `with` statement and its enter & exit semantics.

Usage of a canvas without the `with` statement:

```
self.canvas.add(Color(1., 1., 0))
self.canvas.add(Rectangle(size=(50, 50)))
```

Usage of a canvas with Python's `with` statement:

```
with self.canvas:
    Color(1., 1., 0)
    Rectangle(size=(50, 50))
```

add(*Instruction c*)

after

Property for getting the 'after' group.

ask_update()

Inform the canvas that we'd like it to update on the next frame. This is useful when you need to trigger a redraw due to some value having changed for example.

before

Property for getting the 'before' group.

clear()

Clears every *Instruction* in the canvas, leaving it clean.

draw()

Apply the instruction to our window.

has_after

Property to see if the *after* group has already been created.

New in version 1.7.0.

has_before

Property to see if the *before* group has already been created.

New in version 1.7.0.

opacity

Property to get/set the opacity value of the canvas.

New in version 1.4.1.

The opacity attribute controls the opacity of the canvas and its children. Be careful, it's a cumulative attribute: the value is multiplied to the current global opacity and the result is applied to the current context color.

For example: if your parent has an opacity of 0.5 and a child has an opacity of 0.2, the real opacity of the child will be $0.5 * 0.2 = 0.1$.

Then, the opacity is applied on the shader as:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

remove(*Instruction c*)

class `kivy.graphics.instructions.CanvasBase`

Bases: `kivy.graphics.instructions.InstructionGroup`

CanvasBase provides the context manager methods for the `Canvas`.

class `kivy.graphics.instructions.RenderContext(*args, **kwargs)`

Bases: `kivy.graphics.instructions.Canvas`

- The vertex shader
- The fragment shader
- The default texture
- The state stack (color, texture, matrix...)

shader

Return the shader attached to the render context.

use_parent_frag_modelview

If True, the parent fragment modelview matrix will be used.

New in version 1.10.1: `rc = RenderContext(use_parent_frag_modelview=True)`

use_parent_modelview

If True, the parent modelview matrix will be used.

New in version 1.7.0.

Before:

```
rc['modelview_mat'] = Window.render_context['modelview_mat']
```

Now:

```
rc = RenderContext(use_parent_modelview=True)
```

use_parent_projection

If True, the parent projection matrix will be used.

New in version 1.7.0.

Before:

```
rc['projection_mat'] = Window.render_context['projection_mat']
```

Now:

```
rc = RenderContext(use_parent_projection=True)
```

class `kivy.graphics.instructions.Callback(callback=None, **kwargs)`

Bases: `kivy.graphics.instructions.Instruction`

A Callback is an instruction that will be called when the drawing operation is performed. When adding instructions to a canvas, you can do this:

```
with self.canvas:  
    Color(1, 1, 1)  
    Rectangle(pos=self.pos, size=self.size)  
    Callback(self.my_callback)
```

The definition of the callback must be:

```
def my_callback(self, instr):  
    print('I have been called!')
```

Warning: Note that if you perform many and/or costly calls to callbacks, you might potentially slow down the rendering performance significantly.

The updating of your canvas does not occur until something new happens. From your callback, you can ask for an update:

```
with self.canvas:
    self.cb = Callback(self.my_callback)
# then later in the code
self.cb.ask_update()
```

If you use the Callback class to call rendering methods of another toolkit, you will have issues with the OpenGL context. The OpenGL state may have been manipulated by the other toolkit, and as soon as program flow returns to Kivy, it will just break. You can have glitches, crashes, black holes might occur, etc. To avoid that, you can activate the *reset_context* option. It will reset the OpenGL context state to make Kivy's rendering correct after the call to your callback.

Warning: The *reset_context* is not a full OpenGL reset. If you have issues regarding that, please contact us.

ask_update()

Inform the parent canvas that we'd like it to update on the next frame. This is useful when you need to trigger a redraw due to some value having changed for example.

New in version 1.0.4.

callback

Property for getting/setting func.

reset_context

Set this to True if you want to reset the OpenGL context for Kivy after the callback has been called.

25.4 CGL: standard C interface for OpenGL

Kivy uses OpenGL and therefore requires a backend that provides it. The backend used is controlled through the `USE_OPENGL MOCK` and `USE_SDL2` compile-time variables and through the `KIVY_GL_BACKEND` runtime environmental variable.

Currently, OpenGL is used through direct linking (gl/glew), sdl2, or by mocking it. Setting `USE_OPENGL MOCK` disables gl/glew. Similarly, setting `USE_SDL2` to 0 will disable sdl2. Mocking is always available.

At runtime the following backends are available and can be set using `KIVY_GL_BACKEND`:

- `gl` – Available on unix (the default backend). Unavailable when `USE_OPENGL MOCK=0`. Requires gl be installed.
- `glew` – Available on Windows (the default backend). Unavailable when `USE_OPENGL MOCK=0`. Requires glew be installed.
- `sdl2` – Available on Windows/unix (the default when gl/glew is disabled). Unavailable when `USE_SDL2=0`. Requires `kivy_deps.sdl2` be installed.
- `angle_sdl2` – Available on Windows with Python 3.5+. Unavailable when `USE_SDL2=0`. Requires `kivy_deps.sdl2` and `kivy_deps.angle` be installed.

- **mock** – Always available. Doesn't actually do anything.

Additionally, the following environmental runtime variables control the graphics system:

- **KIVY_GL_DEBUG** – Logs all gl calls when 1.
- **KIVY_GRAPHICS** – Forces OpenGL ES2 when it is **gles**. OpenGL ES2 is always used on the android, ios, rpi, and mali OSs.

```
kivy.graphics.cgl.cgl_get_backend_name(allowed=[], ignored=[])
```

```
kivy.graphics.cgl.cgl_get_initialized_backend_name()
```

```
kivy.graphics.cgl.cgl_init(allowed=[], ignored=[])
```

25.5 Context instructions

The context instructions represent non graphics elements such as:

- Matrix manipulations (PushMatrix, PopMatrix, Rotate, Translate, Scale, MatrixInstruction)
- Color manipulations (Color)
- Texture bindings (BindTexture)

Changed in version 1.0.8: The LineWidth instruction has been removed. It wasn't working before and we actually have no working implementation. We need to do more experimentation to get it right. Check the bug [#207](#) for more information.

class kivy.graphics.context_instructions.**Color**(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

drawn after it.

This represents a color between 0 and 1, but is applied as a *multiplier* to the texture of any vertex instructions following it in a canvas. If no texture is set, the vertex instruction takes the precise color of the Color instruction.

For instance, if a Rectangle has a texture with uniform color (0.5, 0.5, 0.5, 1.0) and the preceding Color has rgba=(1, 0.5, 2, 1), the actual visible color will be (0.5, 0.25, 1.0, 1.0) since the Color instruction is applied as a multiplier to every rgba component. In this case, a Color component outside the 0-1 range gives a visible result as the intensity of the blue component is doubled.

To declare a Color in Python, you can do:

```
from kivy.graphics import Color

# create red v
c = Color(1, 0, 0)
# create blue color
c = Color(0, 1, 0)
# create blue color with 50% alpha
c = Color(0, 1, 0, .5)

# using hsv mode
c = Color(0, 1, 1, mode='hsv')
# using hsv mode + alpha
c = Color(0, 1, 1, .2, mode='hsv')
```

You can also set color components that are available as properties by passing them as keyword arguments:


```
c = Color(b=0.5) # sets the blue component only
```

In kv lang you can set the color properties directly:

```
<Rule>:
    canvas:
        # red color
        Color:
            rgb: 1, 0, 0
        # blue color
        Color:
            rgb: 0, 1, 0
        # blue color with 50% alpha
        Color:
            rgba: 0, 1, 0, .5

        # using hsv mode
        Color:
            hsv: 0, 1, 1
        # using hsv mode + alpha
        Color:
            hsv: 0, 1, 1
            a: .5
```

a

Alpha component, between 0 and 1.

b

Blue component, between 0 and 1.

g

Green component, between 0 and 1.

h

Hue component, between 0 and 1.

hsv

HSV color, list of 3 values in 0-1 range, alpha will be 1.

r

Red component, between 0 and 1.

rgb

RGB color, list of 3 values in 0-1 range. The alpha will be 1.

rgba

RGBA color, list of 4 values in 0-1 range.

s

Saturation component, between 0 and 1.

v

Value component, between 0 and 1.

class kivy.graphics.context_instructions.**BindTexture**(**kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

The BindTexture Instruction will bind a texture and enable GL_TEXTURE_2D for subsequent drawing.

Parameters

texture: Texture Specifies the texture to bind to the given index.

source

Set/get the source (filename) to load for the texture.

class kivy.graphics.context_instructions.**PushMatrix**(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class kivy.graphics.context_instructions.**PopMatrix**(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

class kivy.graphics.context_instructions.**Rotate**(*args, **kwargs)

Bases: kivy.graphics.context_instructions.Transform

on the modelview matrix. You can set the properties of the instructions afterwards with e.g.

```
rot.angle = 90
rot.axis = (0, 0, 1)
```

angle

Property for getting/setting the angle of the rotation.

axis

Property for getting/setting the axis of the rotation.

The format of the axis is (x, y, z).

origin

Origin of the rotation.

New in version 1.7.0.

The format of the origin can be either (x, y) or (x, y, z).

set (float angle, float ax, float ay, float az)

Set the angle and axis of rotation.

```
>>> rotationobject.set(90, 0, 0, 1)
```

Deprecated since version 1.7.0: The set() method doesn't use the new *origin* property.

class kivy.graphics.context_instructions.**Scale**(*args, **kwargs)

Bases: kivy.graphics.context_instructions.Transform

Create using one or three arguments:

```
Scale(s)          # scale all three axes the same
Scale(x, y, z)    # scale the axes independently
```

Deprecated since version 1.6.0: Deprecated single scale property in favor of x, y, z, xyz axis independent scaled factors.

origin

Origin of the scale.

New in version 1.9.0.

The format of the origin can be either (x, y) or (x, y, z).

scale

Property for getting/setting the scale.

Deprecated since version 1.6.0: Deprecated in favor of per axis scale properties x,y,z, xyz, etc.

x

Property for getting/setting the scale on the X axis.

Changed in version 1.6.0.

xyz

3 tuple scale vector in 3D in x, y, and z axis.

Changed in version 1.6.0.

y

Property for getting/setting the scale on the Y axis.

Changed in version 1.6.0.

z

Property for getting/setting the scale on Z axis.

Changed in version 1.6.0.

class kivy.graphics.context_instructions.**Translate**(*args, **kwargs)

Bases: kivy.graphics.context_instructions.Transform

Construct by either:

Translate(x, y)	# translate in just the two axes
Translate(x, y, z)	# translate in all three axes

x

Property for getting/setting the translation on the X axis.

xy

2 tuple with translation vector in 2D for x and y axis.

xyz

3 tuple translation vector in 3D in x, y, and z axis.

y

Property for getting/setting the translation on the Y axis.

z

Property for getting/setting the translation on the Z axis.

class kivy.graphics.context_instructions.**MatrixInstruction**(*args, **kwargs)

Bases: *kivy.graphics.instructions.ContextInstruction*

matrix

Matrix property. Matrix from the transformation module. Setting the matrix using this property when a change is made is important because it will notify the context about the update.

stack

Name of the matrix stack to use. Can be 'modelview_mat', 'projection_mat' or 'frag_modelview_mat'.

New in version 1.6.0.

kivy.graphics.context_instructions.**gl_init_resources**()

25.6 Context management

New in version 1.2.0.

This class manages a registry of all created graphics instructions. It has the ability to flush and delete them.

You can read more about Kivy graphics contexts in the [Graphics](#) module documentation. These are based on [OpenGL graphics contexts](#).

class `kivy.graphics.context.Context`

Bases: `builtins.object`

observer callbacks. See [add_reload_observer\(\)](#) and [remove_reload_observer\(\)](#) for more information.

add_reload_observer(*callback, before=False*)

(internal) Add a callback to be called after the whole graphics context has been reloaded. This is where you can reupload your custom data into the GPU.

Parameters

callback: func(context) -> return None The first parameter will be the context itself

before: boolean, defaults to False If True, the callback will be executed before all the reloading processes. Use it if you want to clear your cache for example.

Changed in version 1.4.0: *before* parameter added.

flag_update_canvas()

flush() → void

gl_dealloc(*largs)

reload()

remove_reload_observer(*callback, before=False*)

(internal) Remove a callback from the observer list previously added by [add_reload_observer\(\)](#).

trigger_gl_dealloc()

25.7 Framebuffer

The Fbo is like an offscreen window. You can activate the fbo for rendering into a texture and use your fbo as a texture for other drawing.

The Fbo acts as a [kivy.graphics.instructions.Canvas](#).

Here is an example of using an fbo for some colored rectangles:

```
from kivy.graphics import Fbo, Color, Rectangle

class FboTest(Widget):
    def __init__(self, **kwargs):
        super(FboTest, self).__init__(**kwargs)

        # first step is to create the fbo and use the fbo texture on other
        # rectangle
```

(continues on next page)

```

with self.canvas:
    # create the fbo
    self.fbo = Fbo(size=(256, 256))

    # show our fbo on the widget in different size
    Color(1, 1, 1)
    Rectangle(size=(32, 32), texture=self.fbo.texture)
    Rectangle(pos=(32, 0), size=(64, 64), texture=self.fbo.texture)
    Rectangle(pos=(96, 0), size=(128, 128), texture=self.fbo.texture)

# in the second step, you can draw whatever you want on the fbo
with self.fbo:
    Color(1, 0, 0, .8)
    Rectangle(size=(256, 64))
    Color(0, 1, 0, .8)
    Rectangle(size=(64, 256))

```

If you change anything in the `self.fbo` object, it will be automatically updated. The canvas where the fbo is put will be automatically updated as well.

25.7.1 Reloading the FBO content

New in version 1.2.0.

If the OpenGL context is lost, then the FBO is lost too. You need to reupload data on it yourself. Use the `Fbo.add_reload_observer()` to add a reloading function that will be automatically called when needed:

```

def __init__(self, **kwargs):
    super(...).__init__(**kwargs)
    self.fbo = Fbo(size=(512, 512))
    self.fbo.add_reload_observer(self.populate_fbo)

    # and load the data now.
    self.populate_fbo(self.fbo)

def populate_fbo(self, fbo):
    with fbo:
        # .. put your Color / Rectangle / ... here

```

This way, you could use the same method for initialization and for reloading. But it's up to you.

class `kivy.graphics.fbo.Fbo(*args, **kwargs)`
 Bases: `kivy.graphics.instructions.RenderContext`
 “with” statement.

Parameters

clear_color: tuple, defaults to (0, 0, 0, 0) Define the default color for clearing the framebuffer

size: tuple, defaults to (1024, 1024) Default size of the framebuffer

push_viewport: bool, defaults to True If True, the OpenGL viewport will be set to the framebuffer size, and will be automatically restored when the framebuffer released.

with_depthbuffer: **bool**, defaults to **False** If True, the framebuffer will be allocated with a Z buffer.

with_stencilbuffer: **bool**, defaults to **False** New in version 1.9.0.

If True, the framebuffer will be allocated with a stencil buffer.

texture: **Texture**, defaults to **None** If None, a default texture will be created.

Note: Using both of `with_stencilbuffer` and `with_depthbuffer` is not supported in kivy 1.9.0

add_reload_observer(*callback*)

Add a callback to be called after the whole graphics context has been reloaded. This is where you can reupload your custom data in GPU.

New in version 1.2.0.

Parameters

callback: **func(context) -> return None** The first parameter will be the context itself

bind()

Bind the FBO to the current opengl context. *Bind* mean that you enable the Framebuffer, and all the drawing operations will act inside the Framebuffer, until **release()** is called.

The bind/release operations are automatically called when you add graphics objects into it. If you want to manipulate a Framebuffer yourself, you can use it like this:

```
self.fbo = FBO()
self.fbo.bind()
# do any drawing command
self.fbo.release()

# then, your fbo texture is available at
print(self.fbo.texture)
```

clear_buffer()

Clear the framebuffer with the **clear_color**.

You need to bind the framebuffer yourself before calling this method:

```
fbo.bind()
fbo.clear_buffer()
fbo.release()
```

clear_color

Clear color in (red, green, blue, alpha) format.

get_pixel_color(*int wx, int wy*)

Get the color of the pixel with specified window coordinates *wx*, *wy*. It returns result in RGBA format.

New in version 1.8.0.

pixels

Get the pixels texture, in RGBA format only, unsigned byte. The origin of the image is at bottom left.

New in version 1.7.0.

release()

Release the Framebuffer (unbind).

remove_reload_observer(*callback*)

Remove a callback from the observer list, previously added by *add_reload_observer()*.

New in version 1.2.0.

size

Size of the framebuffer, in (width, height) format.

If you change the size, the framebuffer content will be lost.

texture

Return the framebuffer texture

25.8 GL instructions

New in version 1.3.0.

25.8.1 Clearing an FBO

To clear an FBO, you can use *ClearColor* and *ClearBuffers* instructions like this example:

```
self.fbo = Fbo(size=self.size)
with self.fbo:
    ClearColor(0, 0, 0, 0)
    ClearBuffers()
```

class kivy.graphics.gl_instructions.**ClearColor**(*r, g, b, a, **kwargs*)

Bases: *kivy.graphics.instructions.Instruction*

New in version 1.3.0.

Sets the clear color used to clear buffers with the glClear function or *ClearBuffers* graphics instructions.

a

Alpha component, between 0 and 1.

b

Blue component, between 0 and 1.

g

Green component, between 0 and 1.

r

Red component, between 0 and 1.

rgb

RGB color, a list of 3 values in 0-1 range where alpha will be 1.

rgba

RGBA color used for the clear color, a list of 4 values in the 0-1 range.

class kivy.graphics.gl_instructions.**ClearBuffers**(*args, **kwargs)

Bases: *kivy.graphics.instructions.Instruction*

New in version 1.3.0.

Clear the buffers specified by the instructions buffer mask property. By default, only the colocal buffer is cleared.

clear_color

If True, the color buffer will be cleared.

clear_depth

If True, the depth buffer will be cleared.

clear_stencil

If True, the stencil buffer will be cleared.

25.9 Graphics compiler

Before rendering an *InstructionGroup*, we compile the group in order to reduce the number of instructions executed at rendering time.

25.9.1 Reducing the context instructions

Imagine that you have a scheme like this:

```
Color(1, 1, 1)
Rectangle(source='button.png', pos=(0, 0), size=(20, 20))
Color(1, 1, 1)
Rectangle(source='button.png', pos=(10, 10), size=(20, 20))
Color(1, 1, 1)
Rectangle(source='button.png', pos=(10, 20), size=(20, 20))
```

The real instructions seen by the graphics canvas would be:

```
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
```

Only the first *Color* and *BindTexture* are useful and really change the context. We can reduce them to:

```
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
Rectangle: push vertices (x1, y1...) to vbo & draw
Rectangle: push vertices (x1, y1...) to vbo & draw
```

This is what the compiler does in the first place, by flagging all the unused instruction with `GL_IGNORE` flag. As soon as a Color content changes, the whole *InstructionGroup* will be recompiled and a previously unused Color might be used for the next compilation.

Note to any Kivy contributor / internal developer:

- All context instructions are checked to see if they change anything in the cache.
- We must ensure that a context instruction is needed for our current Canvas.
- We must ensure that we don't depend of any other canvas.

- We must reset our cache if one of our children is another instruction group because we don't know whether it might do weird things or not.

25.10 OpenGL

This module is a Python wrapper for OpenGL commands.

Warning: Not every OpenGL command has been wrapped and because we are using the C binding for higher performance, and you should rather stick to the Kivy Graphics API. By using OpenGL commands directly, you might change the OpenGL context and introduce inconsistency between the Kivy state and the OpenGL state.

`kivy.graphics.opengl.glActiveTexture(GLenum texture)`

See: [glActiveTexture\(\)](#) on Kronos website

`kivy.graphics.opengl.glAttachShader(GLuint program, GLuint shader)`

See: [glAttachShader\(\)](#) on Kronos website

`kivy.graphics.opengl.glBindAttribLocation(GLuint program, GLuint index, bytes name)`

See: [glBindAttribLocation\(\)](#) on Kronos website

`kivy.graphics.opengl.glBindBuffer(GLenum target, GLuint buffer)`

See: [glBindBuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glBindFramebuffer(GLenum target, GLuint framebuffer)`

See: [glBindFramebuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glBindRenderbuffer(GLenum target, GLuint renderbuffer)`

See: [glBindRenderbuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glBindTexture(GLenum target, GLuint texture)`

See: [glBindTexture\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)`

See: [glBlendColor\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendEquation(GLenum mode)`

See: [glBlendEquation\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendEquationSeparate(GLenum modeRGB, GLenum modeAlpha)`

See: [glBlendEquationSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendFunc(GLenum sfactor, GLenum dfactor)`

See: [glBlendFunc\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendFuncSeparate(GLenum srcRGB, GLenum dstRGB, GLenum srcAlpha, GLenum dstAlpha)`

See: [glBlendFuncSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glBufferData(GLenum target, GLsizeiptr size, bytes data, GLenum usage)`

See: [glBufferData\(\)](#) on Kronos website

`kivy.graphics.opengl.glBufferSubData(GLenum target, GLintptr offset, GLsizeiptr size, bytes data)`

See: [glBufferSubData\(\)](#) on Kronos website

`kivy.graphics.opengl.glCheckFramebufferStatus`(*GLenum target*)
 See: [glCheckFramebufferStatus\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glClear`(*GLbitfield mask*)
 See: [glClear\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glClearColor`(*GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha*)
 See: [glClearColor\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glClearStencil`(*GLint s*)
 See: [glClearStencil\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glColorMask`(*GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha*)
 See: [glColorMask\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCompileShader`(*GLuint shader*)
 See: [glCompileShader\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCompressedTexImage2D`(*GLenum target, GLint level, GLenum internalformat, GLsizei width, GLsizei height, GLint border, GLsizei imageSize, bytes data*)
 See: [glCompressedTexImage2D\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCompressedTexSubImage2D`(*GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLsizei imageSize, bytes data*)
 See: [glCompressedTexSubImage2D\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCopyTexImage2D`(*GLenum target, GLint level, GLenum internalformat, GLint x, GLint y, GLsizei width, GLsizei height, GLint border*)
 See: [glCopyTexImage2D\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCopyTexSubImage2D`(*GLenum target, GLint level, GLint xoffset, GLint yoffset, GLint x, GLint y, GLsizei width, GLsizei height*)
 See: [glCopyTexSubImage2D\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCreateProgram`()
 See: [glCreateProgram\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCreateShader`(*GLenum type*)
 See: [glCreateShader\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glCullFace`(*GLenum mode*)
 See: [glCullFace\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glDeleteBuffers`(*GLsizei n, bytes buffers*)
 See: [glDeleteBuffers\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glDeleteFramebuffers`(*GLsizei n, bytes framebuffers*)
 See: [glDeleteFramebuffers\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glDeleteProgram`(*GLuint program*)
 See: [glDeleteProgram\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glDeleteRenderbuffers`(*GLsizei n, bytes renderbuffers*)
 See: [glDeleteRenderbuffers\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glDeleteShader`(*GLuint shader*)
 See: [glDeleteShader\(\)](#) on Kronos website

`kivy.graphics.opengl.glDeleteTextures`(*GLsizei n, bytes textures*)
 See: [glDeleteTextures\(\)](#) on Kronos website

`kivy.graphics.opengl.glDepthFunc`(*GLenum func*)
 See: [glDepthFunc\(\)](#) on Kronos website

`kivy.graphics.opengl.glDepthMask`(*GLboolean flag*)
 See: [glDepthMask\(\)](#) on Kronos website

`kivy.graphics.opengl.glDetachShader`(*GLuint program, GLuint shader*)
 See: [glDetachShader\(\)](#) on Kronos website

`kivy.graphics.opengl.glDisable`(*GLenum cap*)
 See: [glDisable\(\)](#) on Kronos website

`kivy.graphics.opengl.glDisableVertexAttribArray`(*GLuint index*)
 See: [glDisableVertexAttribArray\(\)](#) on Kronos website

`kivy.graphics.opengl.glDrawArrays`(*GLenum mode, GLint first, GLsizei count*)
 See: [glDrawArrays\(\)](#) on Kronos website

`kivy.graphics.opengl.glDrawElements`(*GLenum mode, GLsizei count, GLenum type, indices*)
 See: [glDrawElements\(\)](#) on Kronos website

`kivy.graphics.opengl.glEnable`(*GLenum cap*)
 See: [glEnable\(\)](#) on Kronos website

`kivy.graphics.opengl.glEnableVertexAttribArray`(*GLuint index*)
 See: [glEnableVertexAttribArray\(\)](#) on Kronos website

`kivy.graphics.opengl.glFinish`()
 See: [glFinish\(\)](#) on Kronos website

`kivy.graphics.opengl.glFlush`()
 See: [glFlush\(\)](#) on Kronos website

`kivy.graphics.opengl.glFramebufferRenderbuffer`(*GLenum target, GLenum attachment, GLenum renderbuffertarget, GLuint renderbuffer*)
 See: [glFramebufferRenderbuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glFramebufferTexture2D`(*GLenum target, GLenum attachment, GLenum textarget, GLuint texture, GLint level*)
 See: [glFramebufferTexture2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glFrontFace`(*GLenum mode*)
 See: [glFrontFace\(\)](#) on Kronos website

`kivy.graphics.opengl.glGenBuffers`(*GLsizei n*)
 See: [glGenBuffers\(\)](#) on Kronos website

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenFramebuffers`(*GLsizei n*)
 See: [glGenFramebuffers\(\)](#) on Kronos website

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenRenderbuffers`(*GLsizei n*)
 See: [glGenRenderbuffers\(\)](#) on Kronos website

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenTextures (GLsizei n)`

See: [glGenTextures\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenerateMipmap (GLenum target)`

See: [glGenerateMipmap\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetActiveAttrib (GLuint program, GLuint index)`

See: [glGetActiveAttrib\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetActiveUniform (GLuint program, GLuint index)`

See: [glGetActiveUniform\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttachedShaders (GLuint program, GLsizei maxcount)`

See: [glGetAttachedShaders\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttribLocation (GLuint program, bytes name)`

See: [glGetAttribLocation\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBooleanv (GLenum pname)`

See: [glGetBooleanv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBufferParameteriv (GLenum target, GLenum pname)`

See: [glGetBufferParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetError ()`

See: [glGetError\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetFloatv (GLenum pname)`

See: [glGetFloatv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetFramebufferAttachmentParameteriv (GLenum target,
GLenum attach-
ment, GLenum
pname)`

See: [glGetFramebufferAttachmentParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetIntegerv (GLenum pname)`

See: [glGetIntegerv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetProgramInfoLog (GLuint program, GLsizei bufsize)`

See: [glGetProgramInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetProgramiv (GLuint program, GLenum pname)`

See: [glGetProgramiv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetRenderbufferParameteriv`(*GLenum target, GLenum pname*)

See: [glGetRenderbufferParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetShaderInfoLog`(*GLuint shader, GLsizei bufsize*)

See: [glGetShaderInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderPrecisionFormat`(*GLenum shadertype, GLenum precisiontype*)

See: [glGetShaderPrecisionFormat\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glGetShaderSource`(*GLuint shader*)

See: [glGetShaderSource\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderiv`(*GLuint shader, GLenum pname*)

See: [glGetShaderiv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetString`(*GLenum name*)

See: [glGetString\(\) on Kronos website](#)

Unlike the C specification, the value will be returned as a string.

`kivy.graphics.opengl.glGetTexParameterfv`(*GLenum target, GLenum pname*)

See: [glGetTexParameterfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetTexParameteriv`(*GLenum target, GLenum pname*)

See: [glGetTexParameteriv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformLocation`(*GLuint program, bytes name*)

See: [glGetUniformLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformfv`(*GLuint program, GLint location*)

See: [glGetUniformfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformiv`(*GLuint program, GLint location*)

See: [glGetUniformiv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetVertexAttribPointerv`(*GLuint index, GLenum pname*)

See: [glGetVertexAttribPointerv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glGetVertexAttribfv`(*GLuint index, GLenum pname*)

See: [glGetVertexAttribfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetVertexAttribiv`(*GLuint index, GLenum pname*)

See: [glGetVertexAttribiv\(\) on Kronos website](#)

`kivy.graphics.opengl.glHint`(*GLenum target, GLenum mode*)

See: [glHint\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsBuffer`(*GLuint buffer*)

See: [glIsBuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsEnabled`(*GLenum cap*)

See: [glIsEnabled\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsFramebuffer`(*GLuint framebuffer*)

See: [glIsFramebuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsProgram`(*GLuint program*)

See: [glIsProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsRenderbuffer`(*GLuint renderbuffer*)

See: [glIsRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsShader`(*GLuint shader*)

See: [glIsShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsTexture`(*GLuint texture*)

See: [glIsTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glLineWidth`(*GLfloat width*)

See: [glLineWidth\(\) on Kronos website](#)

`kivy.graphics.opengl.glLinkProgram`(*GLuint program*)

See: [glLinkProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glPixelStorei`(*GLenum pname, GLint param*)

See: [glPixelStorei\(\) on Kronos website](#)

`kivy.graphics.opengl.glPolygonOffset`(*GLfloat factor, GLfloat units*)

See: [glPolygonOffset\(\) on Kronos website](#)

`kivy.graphics.opengl.glReadPixels`(*GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type*)

See: [glReadPixels\(\) on Kronos website](#)

We support only GL_RGB/GL_RGBA as a format and GL_UNSIGNED_BYTE as a type.

`kivy.graphics.opengl.glReleaseShaderCompiler`()

See: [glReleaseShaderCompiler\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glRenderbufferStorage`(*GLenum target, GLenum internalformat, GLsizei width, GLsizei height*)

See: [glRenderbufferStorage\(\) on Kronos website](#)

`kivy.graphics.opengl.glSampleCoverage`(*GLclampf value, GLboolean invert*)

See: [glSampleCoverage\(\) on Kronos website](#)

`kivy.graphics.opengl.glScissor`(*GLint x, GLint y, GLsizei width, GLsizei height*)

See: [glScissor\(\) on Kronos website](#)

`kivy.graphics.opengl.glShaderBinary`()

See: [glShaderBinary\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glShaderSource`(*GLuint shader, bytes source*)

See: [glShaderSource\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilFunc`(*GLenum func, GLint ref, GLuint mask*)

See: [glStencilFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilFuncSeparate`(*GLenum face, GLenum func, GLint ref, GLuint mask*)

See: [glStencilFuncSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilMask`(*GLuint mask*)

See: [glStencilMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilMaskSeparate`(*GLenum face, GLuint mask*)

See: [glStencilMaskSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilOp`(*GLenum fail, GLenum zfail, GLenum zpass*)

See: [glStencilOp\(\) on Kronos website](#)

`kivy.graphics.opengl.glStencilOpSeparate`(*GLenum face, GLenum fail, GLenum zfail, GLenum zpass*)

See: [glStencilOpSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glTexImage2D`(*GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, bytes pixels*)

See: [glTexImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glTexParameterf`(*GLenum target, GLenum pname, GLfloat param*)

See: [glTexParameterf\(\) on Kronos website](#)

`kivy.graphics.opengl.glTexParameterfv`(*GLenum target, GLenum pname*)

See: [glTexParameterfv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glTexParameteri`(*GLenum target, GLenum pname, GLint param*)

See: [glTexParameteri\(\) on Kronos website](#)

`kivy.graphics.opengl.glTexParameteriv`(*GLenum target, GLenum pname*)

See: [glTexParameteriv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glTexSubImage2D`(*GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, bytes pixels*)

See: [glTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1f`(*GLint location, GLfloat x*)

See: [glUniform1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1fv`(*GLint location, GLsizei count*)

See: [glUniform1fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform1i`(*GLint location, GLint x*)

See: [glUniform1i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1iv`(*GLint location, GLsizei count*)

See: [glUniform1iv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform2f`(*GLint location, GLfloat x, GLfloat y*)

See: [glUniform2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2fv`(*GLint location, GLsizei count*)

See: [glUniform2fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform2i`(*GLint location, GLint x, GLint y*)

See: [glUniform2i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2iv`(*GLint location, GLsizei count*)

See: [glUniform2iv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform3f`(*GLint location, GLfloat x, GLfloat y, GLfloat z*)

See: [glUniform3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3fv`(*GLint location, GLsizei count*)

See: [glUniform3fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform3i`(*GLint location, GLint x, GLint y, GLint z*)

See: [glUniform3i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3iv`(*GLint location, GLsizei count*)

See: [glUniform3iv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform4f`(*GLint location, GLfloat x, GLfloat y, GLfloat z, GLfloat w*)

See: [glUniform4f\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform4fv`(*GLint location, GLsizei count*)

See: [glUniform4fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniform4i`(*GLint location, GLint x, GLint y, GLint z, GLint w*)

See: [glUniform4i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform4iv`(*GLint location, GLsizei count*)

See: [glUniform4iv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix2fv`(*GLint location, GLsizei count*)

See: [glUniformMatrix2fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix3fv`(*GLint location, GLsizei count*)

See: [glUniformMatrix3fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix4fv`(*GLint location, GLsizei count, GLboolean transpose, bytes value*)

See: [glUniformMatrix4fv\(\) on Kronos website](#)

`kivy.graphics.opengl.glUseProgram`(*GLuint program*)

See: [glUseProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glValidateProgram`(*GLuint program*)

See: [glValidateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1f`(*GLuint indx, GLfloat x*)

See: [glVertexAttrib1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1fv`(*GLuint indx, list values*)

See: [glVertexAttrib1fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib2f`(*GLuint indx, GLfloat x, GLfloat y*)

See: [glVertexAttrib2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib2fv`(*GLuint indx, list values*)

See: [glVertexAttrib2fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib3f`(*GLuint indx, GLfloat x, GLfloat y, GLfloat z*)

See: [glVertexAttrib3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib3fv`(*GLuint indx, list values*)

See: [glVertexAttrib3fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib4f(GLuint indx, GLfloat x, GLfloat y, GLfloat z, GLfloat w)`

See: [glVertexAttrib4f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib4fv(GLuint indx, list values)`

See: [glVertexAttrib4fv\(\) on Kronos website](#)

Warning: Not implemented yet.

`kivy.graphics.opengl.glVertexAttribPointer(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, data)`

See: [glVertexAttribPointer\(\) on Kronos website](#)

`kivy.graphics.opengl.glViewport(GLint x, GLint y, GLsizei width, GLsizei height)`

See: [glViewport\(\) on Kronos website](#)

`kivy.graphics.opengl.gl_init_symbols(allowed=[], ignored=[])`

25.11 OpenGL utilities

New in version 1.0.7.

`kivy.graphics.opengl_utils.gl_get_extensions()` → list

Return a list of OpenGL extensions available. All the names in the list have the `GL_` stripped at the start (if it exists) and are in lowercase.

```
>>> print(gl_get_extensions())
['arb_blend_func_extended', 'arb_color_buffer_float', 'arb_compatibility',
 'arb_copy_buffer'... ]
```

`kivy.graphics.opengl_utils.gl_has_extension(name)` → int

Check if an OpenGL extension is available. If the name starts with `GL_`, it will be stripped for the test and converted to lowercase.

```
>>> gl_has_extension('NV_get_tex_image')
False
>>> gl_has_extension('OES_texture_npot')
True
```

`kivy.graphics.opengl_utils.gl_has_capability(int cap)` → int

Return the status of a OpenGL Capability. This is a wrapper that auto-discovers all the capabilities that Kivy might need. The current capabilities tested are:

- GLCAP_BGRA: Test the support of BGRA texture format
- GLCAP_NPOT: Test the support of Non Power of Two texture
- GLCAP_S3TC: Test the support of S3TC texture (DXT1, DXT3, DXT5)
- GLCAP_DXT1: Test the support of DXT texture (subset of S3TC)
- GLCAP_ETC1: Test the support of ETC1 texture

`kivy.graphics.opengl_utils.gl_register_get_size(int constid, int size)`

Register an association between an OpenGL Const used in `glGet*` to a number of elements.

By example, the GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX is a special pname that will return the integer 1 (nvidia only).

```
>>> GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX = 0x9047
>>> gl_register_get_size(GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX, 1)
>>> glGetIntegerv(GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX) [0]
524288
```

kivy.graphics.opengl_utils.gl_has_texture_format(*fmt*) → int
Return whether a texture format is supported by your system, natively or by conversion. For example, if your card doesn't support 'bgra', we are able to convert to 'rgba' but only in software mode.

kivy.graphics.opengl_utils.gl_has_texture_conversion(*fmt*) → int
Return 1 if the texture can be converted to a native format.

kivy.graphics.opengl_utils.gl_has_texture_native_format(*fmt*) → int
Return 1 if the texture format is handled natively.

```
>>> gl_has_texture_format('azdmok')
0
>>> gl_has_texture_format('rgba')
1
>>> gl_has_texture_format('s3tc_dxt1')
[INFO ] [GL          ] S3TC texture support is available
[INFO ] [GL          ] DXT1 texture support is available
1
```

kivy.graphics.opengl_utils.gl_get_texture_formats() → tuple
Return a list of texture formats recognized by kivy. The texture list is informative but might not been supported by your hardware. If you want a list of supported textures, you must filter that list as follows:

```
supported_fmts = [gl_has_texture_format(x) for x in gl_get_texture_formats()]
```

kivy.graphics.opengl_utils.gl_get_version() → tuple
Return the (major, minor) OpenGL version, parsed from the GL_VERSION.

New in version 1.2.0.

kivy.graphics.opengl_utils.gl_get_version_minor() → int
Return the minor component of the OpenGL version.

New in version 1.2.0.

kivy.graphics.opengl_utils.gl_get_version_major() → int
Return the major component of the OpenGL version.

New in version 1.2.0.

25.12 Scissor Instructions

New in version 1.9.1.

Scissor instructions clip your drawing area into a rectangular region.

- **ScissorPush**: Begins clipping, sets the bounds of the clip space
- **ScissorPop**: Ends clipping

The area provided to clip is in screenspace pixels and must be provided as integer values not floats.

The following code will draw a circle on top of our widget while clipping the circle so it does not expand beyond the widget borders.

```
with self.canvas.after:
    #If our widget is inside another widget that modified the coordinates
    #spacing (such as ScrollView) we will want to convert to Window coords
    x,y = self.to_window(*self.pos)
    width, height = self.size
    #We must convert from the possible float values provided by kivy
    #widgets to an integer screenspace, in python3 round returns an int so
    #the int cast will be unnecessary.
    ScissorPush(x=int(round(x)), y=int(round(y)),
                width=int(round(width)), height=int(round(height)))
    Color(rgba=(1., 0., 0., .5))
    Ellipse(size=(width*2., height*2.),
            pos=self.center)
    ScissorPop()
```

class kivy.graphics.scissor_instructions.**Rect**(*int x, int y, int width, int height*)
Bases: `builtins.object`

Rect class used internally by ScissorStack and ScissorPush to determine correct clipping area.

intersect(*Rect other*)

class kivy.graphics.scissor_instructions.**ScissorPop**
Bases: `kivy.graphics.instructions.Instruction`

Pop the scissor stack. Call after ScissorPush, once you have completed the drawing you wish to be clipped.

class kivy.graphics.scissor_instructions.**ScissorPush**(***kwargs*)
Bases: `kivy.graphics.instructions.Instruction`

to control the area and position of the scissoring region. Defaults to 0, 0, 100, 100

Scissor works by clipping all drawing outside of a rectangle starting at int x, int y position and having sides of int width by int height in Window space coordinates

class kivy.graphics.scissor_instructions.**ScissorStack**
Bases: `builtins.object`

Class used internally to keep track of the current state of glScissors regions. Do not instantiate, prefer to inspect the module's `scissor_stack`.

pop()

push(*element*)

25.13 Shader

The *Shader* class handles the compilation of the vertex and fragment shader as well as the creation of the program in OpenGL.

Todo: Include more complete documentation about the shader.

25.13.1 Header inclusion

New in version 1.0.7.

When you are creating a Shader, Kivy will always include default parameters. If you don't want to rewrite this each time you want to customize / write a new shader, you can add the "\$HEADER\$" token and it will be replaced by the corresponding shader header.

Here is the header for the fragment Shader:

```
#ifdef GL_ES
    precision highp float;
#endif

/* Outputs from the vertex shader */
varying vec4 frag_color;
varying vec2 tex_coord0;

/* uniform texture samplers */
uniform sampler2D texture0;

uniform mat4 frag_modelview_mat;
```

And the header for vertex Shader:

```
#ifdef GL_ES
    precision highp float;
#endif

/* Outputs to the fragment shader */
varying vec4 frag_color;
varying vec2 tex_coord0;

/* vertex attributes */
attribute vec2    vPosition;
attribute vec2    vTexCoords0;

/* uniform variables */
uniform mat4      modelview_mat;
uniform mat4      projection_mat;
uniform vec4      color;
uniform float     opacity;
```

25.13.2 Single file glsl shader programs

New in version 1.6.0.

To simplify shader management, the vertex and fragment shaders can be loaded automatically from a single glsl source file (plain text). The file should contain sections identified by a line starting with '—vertex' and '—fragment' respectively (case insensitive), e.g.

```
// anything before a meaningful section such as this comment are ignored

---VERTEX SHADER--- // vertex shader starts here
void main(){
    ...
}

---FRAGMENT SHADER--- // fragment shader starts here
```

(continues on next page)

```
void main(){
    ...
}
```

The source property of the Shader should be set to the filename of a glsl shader file (of the above format), e.g. *phong.glsl*

class kivy.graphics.shader.**Shader**(*unicode* vs=None, *unicode* fs=None, *unicode* source=None)

Bases: builtins.object

Parameters

vs: string, defaults to None Source code for vertex shader

fs: string, defaults to None Source code for fragment shader

fs

Fragment shader source code.

If you set a new fragment shader code source, it will be automatically compiled and will replace the current fragment shader.

source

glsl source code.

source should be the filename of a glsl shader that contains both the vertex and fragment shader sourcecode, each designated by a section header consisting of one line starting with either “-VERTEX” or “-FRAGMENT” (case insensitive).

New in version 1.6.0.

success

Indicate whether the shader loaded successfully and is ready for usage or not.

vs

Vertex shader source code.

If you set a new vertex shader code source, it will be automatically compiled and will replace the current vertex shader.

25.14 Stencil instructions

New in version 1.0.4.

Changed in version 1.3.0: The stencil operation has been updated to resolve some issues that appeared when nested. You **must** now have a StencilUnUse and repeat the same operation as you did after StencilPush.

Stencil instructions permit you to draw and use the current drawing as a mask. They don’t give as much control as pure OpenGL, but you can still do fancy things!

The stencil buffer can be controlled using these 3 instructions:

- **StencilPush**: push a new stencil layer. Any drawing that happens after this will be used as a mask.
- **StencilUse**: now draw the next instructions and use the stencil for masking them.
- **StencilUnUse**: stop using the stencil i.e. remove the mask and draw normally.
- **StencilPop**: pop the current stencil layer.

You should always respect this scheme:

StencilPush

PHASE 1: put any drawing instructions to use as a mask here.

StencilUse

*# PHASE 2: all the drawing here will be automatically clipped by the
mask created in PHASE 1.*

StencilUnUse

PHASE 3: put the same drawing instruction here as you did in PHASE 1

StencilPop

PHASE 4: the stencil is now removed from the stack and unloaded.

25.14.1 Limitations

- Drawing in PHASE 1 and PHASE 3 must not collide or you will get unexpected results
- The stencil is activated as soon as you perform a StencilPush
- The stencil is deactivated as soon as you've correctly popped all the stencil layers
- You must not play with stencils yourself between a StencilPush / StencilPop
- You can push another stencil after a StencilUse / before the StencilPop
- You can push up to 128 layers of stencils (8 for kivy < 1.3.0)

25.14.2 Example of stencil usage

Here is an example, in kv style:

StencilPush

create a rectangular mask with a pos of (100, 100) and a (100, 100) size.

Rectangle:

pos: 100, 100
size: 100, 100

StencilUse

*# we want to show a big green rectangle, however, the previous stencil
mask will crop us :)*

Color:

rgb: 0, 1, 0

Rectangle:

size: 900, 900

StencilUnUse

you must redraw the stencil mask to remove it

Rectangle:

pos: 100, 100
size: 100, 100

StencilPop

class kivy.graphics.stencil_instructions.StencilPush

Bases: *kivy.graphics.instructions.Instruction*

Push the stencil stack. See the module documentation for more information.

class kivy.graphics.stencil_instructions.StencilPop

Bases: *kivy.graphics.instructions.Instruction*

Pop the stencil stack. See the module documentation for more information.

class kivy.graphics.stencil_instructions.StencilUse(**kwargs)

Bases: *kivy.graphics.instructions.Instruction*

more information.

func_op

Determine the stencil operation to use for glStencilFunc(). Can be one of 'never', 'less', 'equal', 'lequal', 'greater', 'notequal', 'gequal' or 'always'.

By default, the operator is set to 'equal'.

New in version 1.5.0.

class kivy.graphics.stencil_instructions.StencilUnUse

Bases: *kivy.graphics.instructions.Instruction*

Use current stencil buffer to unset the mask.

25.15 SVG

New in version 1.9.0.

Warning: This is highly experimental and subject to change. Don't use it in production.

Load an SVG as a graphics instruction:

```
from kivy.graphics.svg import Svg
with widget.canvas:
    svg = Svg("image.svg")
```

There is no widget that can display Svg directly, you have to make your own for now. Check the *examples/svg* for more informations.

class kivy.graphics.svg.Svg(source, anchor_x=0, anchor_y=0, bezier_points=64, circle_points=64, color=None)

Bases: *kivy.graphics.instructions.RenderContext*

anchor_x

Horizontal anchor position for scaling and rotations. Defaults to 0. The symbolic values 'left', 'center' and 'right' are also accepted.

anchor_y

Vertical anchor position for scaling and rotations. Defaults to 0. The symbolic values 'bottom', 'center' and 'top' are also accepted.

color

The default color

Used for SvgElements that specify "currentColor"

Changed in version 1.10.3: The color is gettable and settable

New in version 1.9.1.

current_color

current_color: object

gradients

gradients: object

height

height: 'double'

source

Filename / source to load.

The parsing and rendering is done as soon as you set the source.

Changed in version 2.0.0: The property name is now *source* instead of *filename*

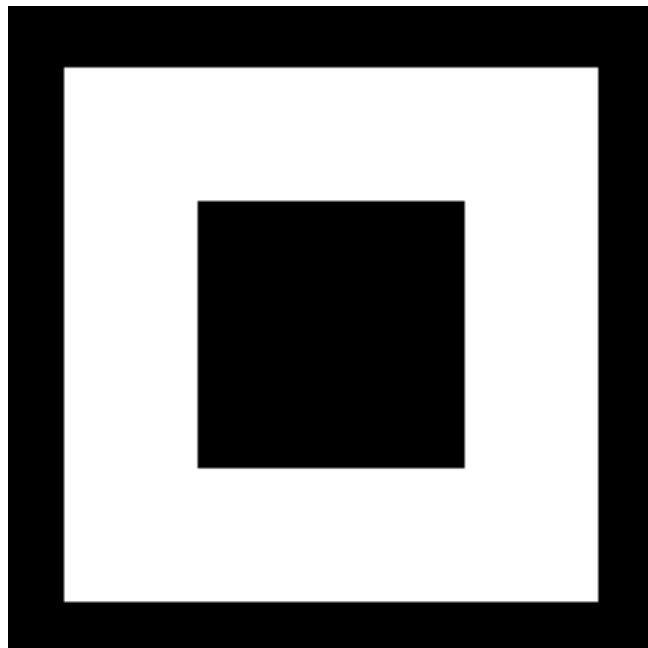
Changed in version 1.10.3: You can get the used filename

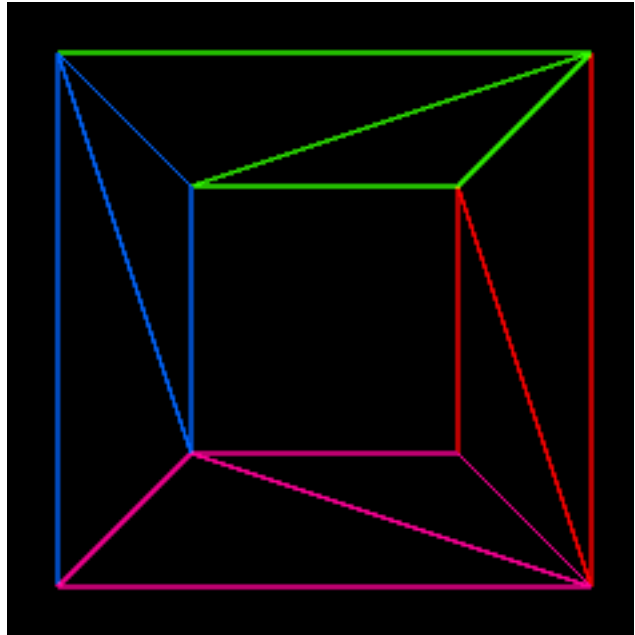
width

width: 'double'

25.16 Tesselator

New in version 1.9.0.





Warning: This is experimental and subject to change as long as this warning notice is present. Only TYPE_POLYGONS is currently supported.

Tesselator is a library for tessellating polygons, based on [libtess2](#). It renders concave filled polygons by first tessellating them into convex polygons. It also supports holes.

25.16.1 Usage

First, you need to create a *Tesselator* object and add contours. The first one is the external contour of your shape and all of the following ones should be holes:

```
from kivy.graphics.tesselator import Tesselator

tess = Tesselator()
tess.add_contour([0, 0, 200, 0, 200, 200, 0, 200])
tess.add_contour([50, 50, 150, 50, 150, 150, 50, 150])
```

Second, call the *Tesselator.tessellate()* method to compute the points. It is possible that the tessellator won't work. In that case, it can return False:

```
if not tess.tessellate():
    print "Tesselator didn't work :("
    return
```

After the tessellation, you have multiple ways to iterate over the result. The best approach is using *Tesselator.meshes* to get a format directly usable for a *Mesh*:

```
for vertices, indices in tess.meshes:
    self.canvas.add(Mesh(
        vertices=vertices,
        indices=indices,
        mode="triangle_fan"
    ))
```

Or, you can get the “raw” result, with just polygons and x/y coordinates with *Tessellator.vertices()*:

```
for vertices in tess.vertices:
    print "got polygon", vertices
```

class `kivy.graphics.tessellator.Tessellator`

Bases: `builtins.object`

Tessellator class. See module for more informations about the usage.

add_contour(*points*)

Add a contour to the tessellator. It can be:

- a list of `[x, y, x2, y2, ...]` coordinates
- a float array: `array("f", [x, y, x2, y2, ...])`
- any buffer with floats in it.

element_count

Returns the number of convex polygon.

meshes

Iterate through the result of the *tessellate()* to give a result that can be easily pushed into Kivy’s Mesh object.

It’s a list of: `[[vertices, indices], [vertices, indices], ...]`. The vertices in the format `[x, y, u, v, x2, y2, u2, v2]`.

Careful, u/v coordinates are the same as x/y. You are responsible to change them for texture mapping if you need to.

You can create Mesh objects like that:

```
tess = Tessellator()
# add contours here
tess.tessellate()
for vertices, indices in self.meshes:
    self.canvas.add(Mesh(
        vertices=vertices,
        indices=indices,
        mode="triangle_fan"))
```

tessellate(*int winding_rule=WINDING_ODD, int element_type=TYPE_POLYGONS, int polysize=65535*) → *int*

Compute all the contours added with *add_contour()*, and generate polygons.

Parameters

winding_rule: enum The winding rule classifies a region as inside if its winding number belongs to the chosen category. Can be one of `WINDING_ODD`, `WINDING_NONZERO`, `WINDING_POSITIVE`, `WINDING_NEGATIVE`, `WINDING_ABS_GEQ_TWO`. Defaults to `WINDING_ODD`.

element_type: enum The result type, you can generate the polygons with `TYPE_POLYGONS`, or the contours with `TYPE_BOUNDARY_CONTOURS`. Defaults to `TYPE_POLYGONS`.

Returns 1 if the tessellation happened, 0 otherwise.

Return type *int*

vertex_count

Returns the number of vertex generated.

This is the raw result, however, because the Tessellator format the result for you with *meshes* or *vertices* per polygon, you'll have more vertices in the result

vertices

Iterate through the result of the *tessellate()* in order to give only a list of $[x, y, x2, y2, \dots]$ polygons.

25.17 Texture

Changed in version 1.6.0: Added support for paletted texture on OES: 'palette4_rgb8', 'palette4_rgba8', 'palette4_r5_g6_b5', 'palette4_rgba4', 'palette4_rgb5_a1', 'palette8_rgb8', 'palette8_rgba8', 'palette8_r5_g6_b5', 'palette8_rgba4' and 'palette8_rgb5_a1'.

Texture is a class that handles OpenGL textures. Depending on the hardware, some OpenGL capabilities might not be available (BGRA support, NPOT support, etc.)

You cannot instantiate this class yourself. You must use the function *Texture.create()* to create a new texture:

```
texture = Texture.create(size=(640, 480))
```

When you create a texture, you should be aware of the default color and buffer format:

- the color/pixel format (*Texture.colorfmt*) that can be one of 'rgb', 'rgba', 'luminance', 'luminance_alpha', 'bgr' or 'bgra'. The default value is 'rgb'
- the buffer format determines how a color component is stored into memory. This can be one of 'ubyte', 'ushort', 'uint', 'byte', 'short', 'int' or 'float'. The default value and the most commonly used is 'ubyte'.

So, if you want to create an RGBA texture:

```
texture = Texture.create(size=(640, 480), colorfmt='rgba')
```

You can use your texture in almost all vertex instructions with the *kivy.graphics.VertexInstruction.texture* parameter. If you want to use your texture in kv lang, you can save it in an *ObjectProperty* inside your widget.

Warning: Using Texture before OpenGL has been initialized will lead to a crash. If you need to create textures before the application has started, import Window first: *from kivy.core.window import Window*

25.17.1 Blitting custom data

You can create your own data and blit it to the texture using *Texture.blit_buffer()*.

For example, to blit immutable bytes data:

```
# create a 64x64 texture, defaults to rgba / ubyte
texture = Texture.create(size=(64, 64))

# create 64x64 rgb tab, and fill with values from 0 to 255
# we'll have a gradient from black to white
size = 64 * 64 * 3
buf = [int(x * 255 / size) for x in range(size)]
```

(continues on next page)

```

# then, convert the array to a ubyte string
buf = b''.join(map(chr, buf))

# then blit the buffer
texture.blit_buffer(buf, colorfmt='rgb', bufferfmt='ubyte')

# that's all ! you can use it in your graphics now :)
# if self is a widget, you can do this
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=(64, 64))

```

Since 1.9.0, you can blit data stored in a instance that implements the python buffer interface, or a memoryview thereof, such as numpy arrays, python `array.array`, a `bytearray`, or a cython array. This is beneficial if you expect to blit similar data, with perhaps a few changes in the data.

When using a bytes representation of the data, for every change you have to regenerate the bytes instance, from perhaps a list, which is very inefficient. When using a buffer object, you can simply edit parts of the original data. Similarly, unless starting with a bytes object, converting to bytes requires a full copy, however, when using a buffer instance, no memory is copied, except to upload it to the GPU.

Continuing with the example above:

```

from array import array

size = 64 * 64 * 3
buf = [int(x * 255 / size) for x in range(size)]
# initialize the array with the buffer values
arr = array('B', buf)
# now blit the array
texture.blit_buffer(arr, colorfmt='rgb', bufferfmt='ubyte')

# now change some elements in the original array
arr[24] = arr[50] = 99
# blit again the buffer
texture.blit_buffer(arr, colorfmt='rgb', bufferfmt='ubyte')

```

25.17.2 BGR/BGRA support

The first time you try to create a BGR or BGRA texture, we check whether your hardware supports BGR / BGRA textures by checking the extension 'GL_EXT_bgra'.

If the extension is not found, the conversion to RGB / RGBA will be done in software.

25.17.3 NPOT texture

Changed in version 1.0.7: If your hardware supports NPOT, no POT is created.

As the OpenGL documentation says, a texture must be power-of-two sized. That means your width and height can be one of 64, 32, 256... but not 3, 68, 42. NPOT means non-power-of-two. OpenGL ES 2 supports NPOT textures natively but with some drawbacks. Another type of NPOT texture is called a rectangle texture. POT, NPOT and textures all have their own pro/cons.

Features	POT	NPOT	Rectangle
OpenGL Target	GL_TEXTURE_2D	GL_TEXTURE_2D	GL_TEXTURE_RECTANGLE_(NV ARB EXT)
Texture coords	0-1 range	0-1 range	width-height range
Mipmapping	Supported	Partially	No
Wrap mode	Supported	Supported	No

If you create a NPOT texture, we first check whether your hardware supports it by checking the extensions `GL_ARB_texture_non_power_of_two` or `OES_texture_npot`. If none of these are available, we create the nearest POT texture that can contain your NPOT texture. The `Texture.create()` will return a `TextureRegion` instead.

25.17.4 Texture atlas

A texture atlas is a single texture that contains many images. If you want to separate the original texture into many single ones, you don't need to. You can get a region of the original texture. That will return the original texture with custom texture coordinates:

```
# for example, load a 128x128 image that contain 4 64x64 images
from kivy.core.image import Image
texture = Image('mycombinedimage.png').texture

bottomleft = texture.get_region(0, 0, 64, 64)
bottomright = texture.get_region(0, 64, 64, 64)
topleft = texture.get_region(64, 0, 128, 64)
topright = texture.get_region(64, 64, 128, 128)
```

25.17.5 Mipmapping

New in version 1.0.7.

Mipmapping is an OpenGL technique for enhancing the rendering of large textures to small surfaces. Without mipmapping, you might see pixelation when you render to small surfaces. The idea is to precalculate the subtexture and apply some image filter as a linear filter. Then, when you render a small surface, instead of using the biggest texture, it will use a lower filtered texture. The result can look better this way.

To make that happen, you need to specify `mipmap=True` when you create a texture. Some widgets already give you the ability to create mipmapped textures, such as the `Label` and `Image`.

From the OpenGL Wiki : "So a 64x16 2D texture can have 5 mip-maps: 32x8, 16x4, 8x2, 4x1, 2x1, and 1x1". Check <http://www.opengl.org/wiki/Texture> for more information.

Note: As the table in previous section said, if your texture is NPOT, we create the nearest POT texture and generate a mipmap from it. This might change in the future.

25.17.6 Reloading the Texture

New in version 1.2.0.

If the OpenGL context is lost, the Texture must be reloaded. Textures that have a source are automatically reloaded but generated textures must be reloaded by the user.

Use the `Texture.add_reload_observer()` to add a reloading function that will be automatically called when needed:

```
def __init__(self, **kwargs):
    super(...).__init__(**kwargs)
    self.texture = Texture.create(size=(512, 512), colorfmt='RGB',
                                  bufferfmt='ubyte')
    self.texture.add_reload_observer(self.populate_texture)

    # and load the data now.
    self.cbuffer = '\x00\xff' * 512 * 512
    self.populate_texture(self.texture)

def populate_texture(self, texture):
    texture.blit_buffer(self.cbuffer)
```

This way, you can use the same method for initialization and reloading.

Note: For all text rendering with our core text renderer, the texture is generated but we already bind a method to redo the text rendering and reupload the text to the texture. You don't have to do anything.

```
class kivy.graphics.texture.Texture(width, height, target, texid=0, colorfmt=u'rgb',
                                     bufferfmt=u'ubyte', mipmap=False, source=None,
                                     callback=None, icolorfmt=u'rgb')
```

Bases: `builtins.object`

textures or complex textures based on `ImageData`.

add_reload_observer(*callback*)

Add a callback to be called after the whole graphics context has been reloaded. This is where you can reupload your custom data into the GPU.

New in version 1.2.0.

Parameters

callback: `func(context) -> return None` The first parameter will be the context itself.

ask_update(*callback*)

Indicate that the content of the texture should be updated and the callback function needs to be called when the texture will be used.

bind()

Bind the texture to the current opengl state.

blit_buffer(*pbuffer, size=None, colorfmt=None, pos=None, bufferfmt=None, mipmap_level=0, mipmap_generation=True, int rowlength=0*)

Blit a buffer into the texture.

Note: Unless the canvas will be updated due to other changes, `ask_update()` should be called in order to update the texture.

Parameters

pbuffer: bytes, or a class that implements the buffer interface (including memoryview).

A buffer containing the image data. It can be either a bytes object or a instance of a class that implements the python buffer interface, e.g. `array.array`, `bytearray`, `numpy arrays` etc. If it's not a bytes object, the underlying buffer

must be contiguous, have only one dimension and must not be readonly, even though the data is not modified, due to a cython limitation. See module description for usage details.

size: tuple, defaults to texture size Size of the image (width, height)

colorfmt: str, defaults to 'rgb' Image format, can be one of 'rgb', 'rgba', 'bgr', 'bgra', 'luminance' or 'luminance_alpha'.

pos: tuple, defaults to (0, 0) Position to blit in the texture.

bufferfmt: str, defaults to 'ubyte' Type of the data buffer, can be one of 'ubyte', 'ushort', 'uint', 'byte', 'short', 'int' or 'float'.

mipmap_level: int, defaults to 0 Indicate which mipmap level we are going to update.

mipmap_generation: bool, defaults to True Indicate if we need to regenerate the mipmap from level 0.

Changed in version 1.0.7: added *mipmap_level* and *mipmap_generation*

Changed in version 1.9.0: *pbuffer* can now be any class instance that implements the python buffer interface and / or memoryviews thereof.

blit_data(*im*, *pos=None*)

Replace a whole texture with image data.

bufferfmt

Return the buffer format used in this texture (readonly).

New in version 1.2.0.

colorfmt

Return the color format used in this texture (readonly).

New in version 1.0.7.

static create(*size=None*, *colorfmt=None*, *bufferfmt=None*, *mipmap=False*, *callback=None*, *icolorfmt=None*)

texture_create(*size=None*, *colorfmt=None*, *bufferfmt=None*, *mipmap=False*, *callback=None*, *icolorfmt=None*) Create a texture based on size.

Parameters

size: tuple, defaults to (128, 128) Size of the texture.

colorfmt: str, defaults to 'rgba' Color format of the texture. Can be 'rgba' or 'rgb', 'luminance' or 'luminance_alpha'. On desktop, additional values are available: 'red', 'rg'.

icolorfmt: str, defaults to the value of colorfmt Internal format storage of the texture. Can be 'rgba' or 'rgb', 'luminance' or 'luminance_alpha'. On desktop, additional values are available: 'r8', 'rg8', 'rgba8'.

bufferfmt: str, defaults to 'ubyte' Internal buffer format of the texture. Can be 'ubyte', 'ushort', 'uint', 'bute', 'short', 'int' or 'float'.

mipmap: bool, defaults to False If True, it will automatically generate the mipmap texture.

callback: callable(), defaults to False If a function is provided, it will be called when data is needed in the texture.

Changed in version 1.7.0: *callback* has been added

static create_from_data(*im, mipmap=False*)
 texture_create_from_data(im, mipmap=False) Create a texture from an ImageData class.

flip_horizontal()
 Flip tex_coords for horizontal display.
 New in version 1.9.0.

flip_vertical()
 Flip tex_coords for vertical display.

get_region(*x, y, width, height*)
 Return a part of the texture defined by the rectangular arguments (x, y, width, height). Returns a *TextureRegion* instance.

height
 Return the height of the texture (readonly).

id
 Return the OpenGL ID of the texture (readonly).

mag_filter
 Get/set the mag filter texture. Available values:

- linear
- nearest

Check the opengl documentation for more information about the behavior of these values :
<http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

min_filter
 Get/set the min filter texture. Available values:

- linear
- nearest
- linear_mipmap_linear
- linear_mipmap_nearest
- nearest_mipmap_nearest
- nearest_mipmap_linear

Check the opengl documentation for more information about the behavior of these values :
<http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

mipmap
 Return True if the texture has mipmap enabled (readonly).

pixels
 Get the pixels texture, in RGBA format only, unsigned byte. The origin of the image is at bottom left.
 New in version 1.7.0.

remove_reload_observer(*callback*)
 Remove a callback from the observer list, previously added by *add_reload_observer()*.
 New in version 1.2.0.

save(*filename, flipped=True, fmt=None*)
 Save the texture content to a file. Check *kivy.core.image.Image.save()* for more information.
 The flipped parameter flips the saved image vertically, and defaults to True.

New in version 1.7.0.

Changed in version 1.8.0: Parameter *flipped* added, defaults to True. All the OpenGL Texture are readed from bottom / left, it need to be flipped before saving. If you don't want to flip the image, set flipped to False.

Changed in version 1.11.0: Parameter *fnt* added, to pass the final format to the image provider. Used if filename is a BytesIO

size

Return the (width, height) of the texture (readonly).

target

Return the OpenGL target of the texture (readonly).

tex_coords

Return the list of tex_coords (opengl).

uvpos

Get/set the UV position inside the texture.

uvsize

Get/set the UV size inside the texture.

Warning: The size can be negative if the texture is flipped.

width

Return the width of the texture (readonly).

wrap

Get/set the wrap texture. Available values:

- repeat
- mirrored_repeat
- clamp_to_edge

Check the opengl documentation for more information about the behavior of these values : <http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

class `kivy.graphics.texture.TextureRegion`(*int x, int y, int width, int height, Texture origin*)

Bases: `kivy.graphics.texture.Texture`

texture handling.

ask_update(*callback*)

bind()

pixels

Get the pixels texture, in RGBA format only, unsigned byte. The origin of the image is at bottom left.

New in version 1.7.0.

25.18 Transformation

This module contains a Matrix class used for our Graphics calculations. We currently support:

- rotation, translation and scaling matrices

- multiplication matrix
- clip matrix (with or without perspective)
- transformation matrix for 3d touch

For more information on transformation matrices, please see the [OpenGL Matrices Tutorial](#).

Changed in version 1.6.0: Added *Matrix.perspective()*, *Matrix.look_at()* and *Matrix.transpose()*.

class kivy.graphics.transformation.**Matrix**
 Bases: `builtins.object`

```
>>> from kivy.graphics.transformation import Matrix
>>> m = Matrix()
>>> print(m)
[[ 1.000000  0.000000  0.000000  0.000000 ]
 [ 0.000000  1.000000  0.000000  0.000000 ]
 [ 0.000000  0.000000  1.000000  0.000000 ]
 [ 0.000000  0.000000  0.000000  1.000000 ]]
```

[0 1 2 3] [4 5 6 7] [8 9 10 11] [12 13 14 15]

get()

Retrieve the value of the current as a flat list.

New in version 1.9.1.

identity() → Matrix

Reset the matrix to the identity matrix (inplace).

inverse() → Matrix

Return the inverse of the matrix as a new Matrix.

look_at(*double eyex, double eyey, double eyez, double centerx, double centery, double centerz, double upx, double upy, double upz*)

Returns a new lookat Matrix (similar to *gluLookAt*).

Parameters

eyex: float Eyes X co-ordinate

eyey: float Eyes Y co-ordinate

eyez: float Eyes Z co-ordinate

centerx: float The X position of the reference point

centery: float The Y position of the reference point

centerz: float The Z position of the reference point

upx: float The X value up vector.

upy: float The Y value up vector.

upz: float The Z value up vector.

New in version 1.6.0.

multiply(*mb, Matrix ma*) → Matrix

Multiply the given matrix with self (from the left) i.e. we premultiply the given matrix by the current matrix and return the result (not inplace):

```
m.multiply(n) -> n * m
```

Parameters

ma: Matrix The matrix to multiply by

normal_matrix() → Matrix

Computes the normal matrix, which is the inverse transpose of the top left 3x3 modelview matrix used to transform normals into eye/camera space.

New in version 1.6.0.

perspective(*double fovy, double aspect, double zNear, double zFar*) → Matrix

Creates a perspective matrix (inplace).

Parameters

fovy: float “Field Of View” angle

aspect: float Aspect ratio

zNear: float Near clipping plane

zFar: float Far clippin plane

New in version 1.6.0.

project(*double objx, double objy, double objz, Matrix model, Matrix proj, double vx, double vy, double vw, double vh*)

Project a point from 3d space into a 2d viewport.

Parameters

objx: float Points X co-ordinate

objy: float Points Y co-ordinate

objz: float Points Z co-ordinate

model: Matrix The model matrix

proj: Matrix The projection matrix

vx: float Viewports X co-ordinate

vy: float Viewports y co-ordinate

vw: float Viewports width

vh: float Viewports height

New in version 1.7.0.

rotate(*double angle, double x, double y, double z*) → Matrix

Rotate the matrix through the angle around the axis (x, y, z) (inplace).

Parameters

angle: float The angle through which to rotate the matrix

x: float X position of the point

y: float Y position of the point

z: float Z position of the point

scale(*double x, double y, double z*) → Matrix

Scale the current matrix by the specified factors over each dimension (inplace).

Parameters

x: float The scale factor along the X axis

y: float The scale factor along the Y axis

z: float The scale factor along the Z axis

set (*flat=None, array=None*)

Insert custom values into the matrix in a flat list format or 4x4 array format like below:

```
m.set(array=[
    [1.0, 0.0, 0.0, 0.0],
    [0.0, 1.0, 0.0, 0.0],
    [0.0, 0.0, 1.0, 0.0],
    [0.0, 0.0, 0.0, 1.0]]
)
```

New in version 1.9.0.

tolist()

Retrieve the value of the current matrix in numpy format. for example m.tolist() will return:

```
[[1.000000, 0.000000, 0.000000, 0.000000],
 [0.000000, 1.000000, 0.000000, 0.000000],
 [0.000000, 0.000000, 1.000000, 0.000000],
 [0.000000, 0.000000, 0.000000, 1.000000]]
```

you can use this format to plug the result straight into numpy in this way
numpy.array(m.tolist())

New in version 1.9.0.

transform_point (*double x, double y, double z, t=None*) → tuple

Transforms the point by the matrix and returns the transformed point as a (x, y, z) tuple.
If the point is a vector v, the returned values is $v2 = matrix * v$.

If t is provided, it multiplies it with the last column of the matrix and returns the transformed (x, y, z, t).

translate (*double x, double y, double z*) → Matrix

Translate the matrix.

Parameters

x: float The translation factor along the X axis

y: float The translation factor along the Y axis

z: float The translation factor along the Z axis

transpose() → Matrix

Return the transposed matrix as a new Matrix.

New in version 1.6.0.

view_clip (*double left, double right, double bottom, double top, double near, double far, int perspective*) → Matrix

Create a clip matrix (inplace).

Parameters

left: float Co-ordinate

right: float Co-ordinate

bottom: float Co-ordinate

top: float Co-ordinate

near: float Co-ordinate

far: float Co-ordinate

perspective: int Co-ordinate

Changed in version 1.6.0: Enable support for perspective parameter.

25.19 Vertex Instructions

This module includes all the classes for drawing simple vertex objects.

25.19.1 Updating properties

The list attributes of the graphics instruction classes (e.g. *Triangle.points*, *Mesh.indices* etc.) are not Kivy properties but Python properties. As a consequence, the graphics will only be updated when the list object itself is changed and not when list values are modified.

For example in python:

```
class MyWidget(Button):

    triangle = ObjectProperty(None)
    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        with self.canvas:
            self.triangle = Triangle(points=[0,0, 100,100, 200,0])
```

and in kv:

```
<MyWidget>:
    text: 'Update'
    on_press:
        self.triangle.points[3] = 400
```

Although pressing the button will change the triangle coordinates, the graphics will not be updated because the list itself has not changed. Similarly, no updates will occur using any syntax that changes only elements of the list e.g. `self.triangle.points[0:2] = [10,10]` or `self.triangle.points.insert(10)` etc. To force an update after a change, the list variable itself must be changed, which in this case can be achieved with:

```
<MyWidget>:
    text: 'Update'
    on_press:
        self.triangle.points[3] = 400
        self.triangle.points = self.triangle.points
```

class `kivy.graphics.vertex_instructions.Triangle`(**kwargs)
Bases: *kivy.graphics.instructions.VertexInstruction*

Parameters

points: list List of points in the format (x1, y1, x2, y2, x3, y3).

points

Property for getting/settings points of the triangle.

class `kivy.graphics.vertex_instructions.Quad`(**kwargs)
Bases: *kivy.graphics.instructions.VertexInstruction*

Parameters

points: list List of point in the format (x1, y1, x2, y2, x3, y3, x4, y4).

points

Property for getting/settings points of the quad.

class `kivy.graphics.vertex_instructions.Rectangle(**kwargs)`

Bases: `kivy.graphics.instructions.VertexInstruction`

Parameters

pos: **list** Position of the rectangle, in the format (x, y).

size: **list** Size of the rectangle, in the format (width, height).

pos

Property for getting/settings the position of the rectangle.

size

Property for getting/settings the size of the rectangle.

class `kivy.graphics.vertex_instructions.RoundedRectangle(**kwargs)`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

New in version 1.9.1.

Parameters

segments: **int, defaults to 10** Define how many segments are needed for drawing the rounded corner. The drawing will be smoother if you have many segments.

radius: **list, defaults to [(10.0, 10.0), (10.0, 10.0), (10.0, 10.0), (10.0, 10.0)]** Specifies the radii used for the rounded corners clockwise: top-left, top-right, bottom-right, bottom-left. Elements of the list can be numbers or tuples of two numbers to specify different x,y dimensions. One value will define all corner radii to be of this value. Four values will define each corner radius separately. Higher numbers of values will be truncated to four. The first value will be used for all corners if there are fewer than four values.

radius

Corner radii of the rounded rectangle, defaults to [10,].

segments

Property for getting/setting the number of segments for each corner.

class `kivy.graphics.vertex_instructions.BorderImage(**kwargs)`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

concept of a CSS3 border-image.

Parameters

border: **list** Border information in the format (bottom, right, top, left). Each value is in pixels.

auto_scale: **string** New in version 1.9.1.

Changed in version 1.9.2: This used to be a bool and has been changed to be a string state.

Can be one of 'off', 'both', 'x_only', 'y_only', 'y_full_x_lower', 'x_full_y_lower', 'both_lower'.

Autoscale controls the behavior of the 9-slice.

By default the border values are preserved exactly, meaning that if the total size of the object is smaller than the border values you will have some 'rendering errors' where your texture appears inside out. This also makes it impossible to achieve a rounded button that scales larger than the size of its source texture.

The various options for `auto_scale` will let you achieve some mixes of the 2 types of rendering.

'off': is the default and behaves as `BorderImage` did when `auto_scale` was `False` before.

'both': Scales both x and y dimension borders according to the size of the `BorderImage`, this disables the `BorderImage` making it render the same as a regular `Image`.

'x_only': The Y dimension functions as the default, and the X scales to the size of the `BorderImage`'s width.

'y_only': The X dimension functions as the default, and the Y scales to the size of the `BorderImage`'s height.

'y_full_x_lower': Y scales as in 'y_only', Y scales if the size of the scaled version would be smaller than the provided border only.

'x_full_y_lower': X scales as in 'x_only', Y scales if the size of the scaled version would be smaller than the provided border only.

'both_lower': This is what `auto_scale` did when it was `True` in 1.9.1 Both X and Y dimensions will be scaled if the `BorderImage` is smaller than the source.

If the `BorderImage`'s size is less than the sum of its borders, horizontally or vertically, and this property is set to `True`, the borders will be rescaled to accommodate for the smaller size.

auto_scale

Property for setting if the corners are automatically scaled when the `BorderImage` is too small.

border

Property for getting/setting the border of the class.

display_border

Property for getting/setting the border display size.

class `kivy.graphics.vertex_instructions.Ellipse(*args, **kwargs)`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

Changed in version 1.0.7: Added `angle_start` and `angle_end`.

Parameters

segments: int, defaults to 180 Define how many segments are needed for drawing the ellipse. The drawing will be smoother if you have many segments.

angle_start: int, defaults to 0 Specifies the starting angle, in degrees, of the disk portion.

angle_end: int, defaults to 360 Specifies the ending angle, in degrees, of the disk portion.

angle_end

End angle of the ellipse in degrees, defaults to 360.

angle_start

Start angle of the ellipse in degrees, defaults to 0.

segments

Property for getting/setting the number of segments of the ellipse.

class `kivy.graphics.vertex_instructions.Line(**kwargs)`

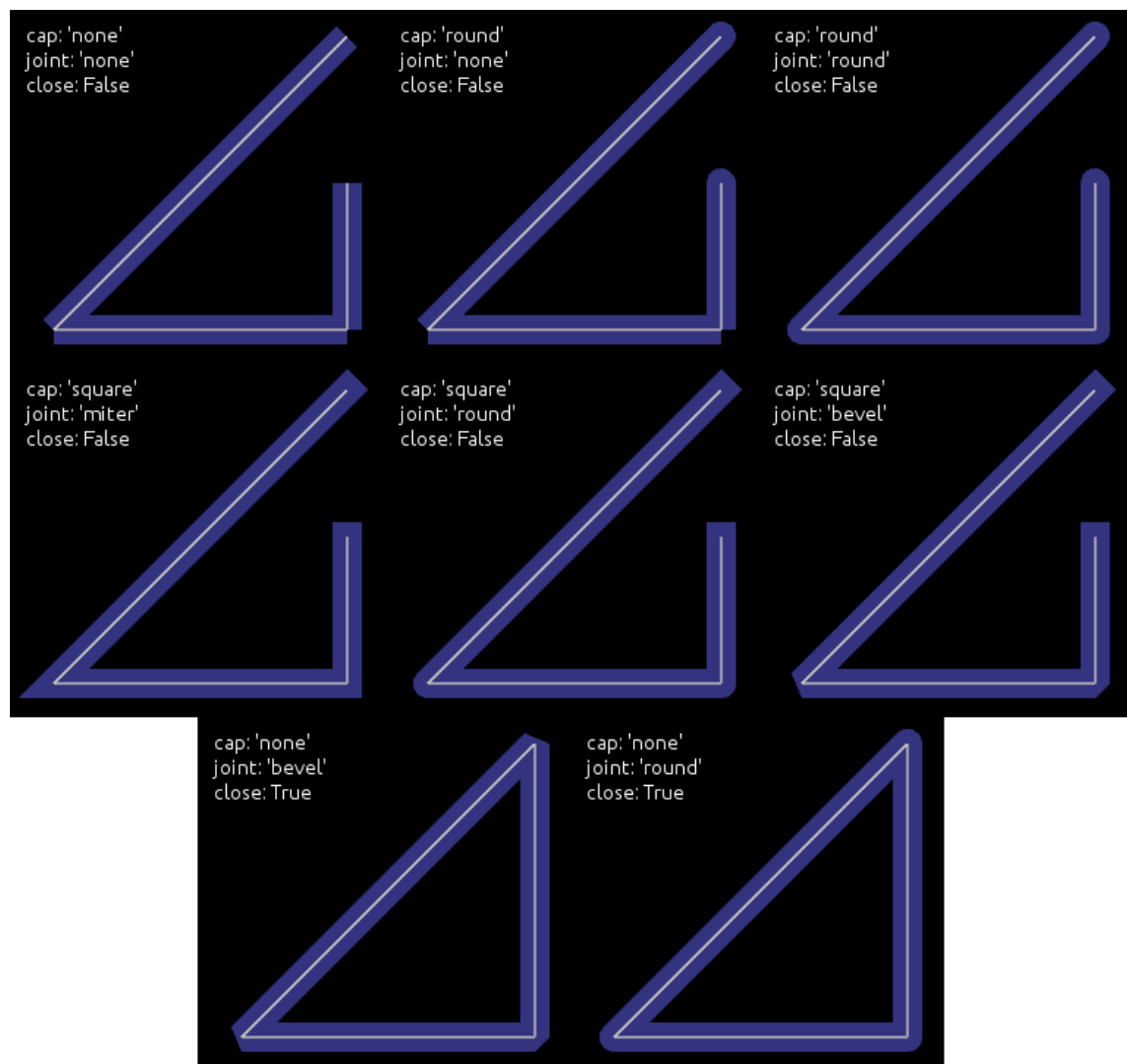
Bases: `kivy.graphics.instructions.VertexInstruction`

Drawing a line can be done easily:

```
with self.canvas:  
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

The line has 3 internal drawing modes that you should be aware of for optimal results:

1. If the *width* is 1.0, then the standard GL_LINE drawing from OpenGL will be used. *dash_length*, *dash_offset*, and *dashes* will work, while properties for cap and joint have no meaning here.
2. If the *width* is greater than 1.0, then a custom drawing method, based on triangulation, will be used. *dash_length*, *dash_offset*, and *dashes* do not work in this mode. Additionally, if the current color has an alpha less than 1.0, a stencil will be used internally to draw the line.



Parameters

points: list List of points in the format (x1, y1, x2, y2...)

dash_length: int Length of a segment (if dashed), defaults to 1.

dash_offset: int Offset between the end of a segment and the beginning of the next one, defaults to 0. Changing this makes it dashed.

dashes: list of ints List of [ON length, offset, ON length, offset, ...]. E.g. [2, 4, 1, 6, 8, 2] would create a line with the first dash length 2 then an offset of 4 then

a dash length of 1 then an offset of 6 and so on. Defaults to `[]`. Changing this makes it dashed and overrides `dash_length` and `dash_offset`.

width: float Width of the line, defaults to 1.0.

cap: str, defaults to 'round' See `cap` for more information.

joint: str, defaults to 'round' See `joint` for more information.

cap_precision: int, defaults to 10 See `cap_precision` for more information

joint_precision: int, defaults to 10 See `joint_precision` for more information
See `cap_precision` for more information.

joint_precision: int, defaults to 10 See `joint_precision` for more information.

close: bool, defaults to False If True, the line will be closed.

circle: list If set, the `points` will be set to build a circle. See `circle` for more information.

ellipse: list If set, the `points` will be set to build an ellipse. See `ellipse` for more information.

rectangle: list If set, the `points` will be set to build a rectangle. See `rectangle` for more information.

bezier: list If set, the `points` will be set to build a bezier line. See `bezier` for more information.

bezier_precision: int, defaults to 180 Precision of the Bezier drawing.

Changed in version 1.0.8: `dash_offset` and `dash_length` have been added.

Changed in version 1.4.1: `width`, `cap`, `joint`, `cap_precision`, `joint_precision`, `close`, `ellipse`, `rectangle` have been added.

Changed in version 1.4.1: `bezier`, `bezier_precision` have been added.

Changed in version 1.11.0: `dashes` have been added

bezier

Use this property to build a bezier line, without calculating the `points`. You can only set this property, not get it.

The argument must be a tuple of 2n elements, n being the number of points.

Usage:

```
Line(bezier=(x1, y1, x2, y2, x3, y3))
```

New in version 1.4.2.

Note: Bezier lines calculations are inexpensive for a low number of points, but complexity is quadratic, so lines with a lot of points can be very expensive to build, use with care!

bezier_precision

Number of iteration for drawing the bezier between 2 segments, defaults to 180. The `bezier_precision` must be at least 1.

New in version 1.4.2.

cap

Determine the cap of the line, defaults to 'round'. Can be one of 'none', 'square' or 'round'

New in version 1.4.1.

cap_precision

Number of iteration for drawing the “round” cap, defaults to 10. The cap_precision must be at least 1.

New in version 1.4.1.

circle

Use this property to build a circle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (center_x, center_y, radius, angle_start, angle_end, segments):

- center_x and center_y represent the center of the circle
- radius represent the radius of the circle
- (optional) angle_start and angle_end are in degree. The default value is 0 and 360.
- (optional) segments is the precision of the ellipse. The default value is calculated from the range between angle.

Note that it’s up to you to *close* the circle or not.

For example, for building a simple ellipse, in python:

```
# simple circle
Line(circle=(150, 150, 50))

# only from 90 to 180 degrees
Line(circle=(150, 150, 50, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(circle=(150, 150, 50, 90, 180, 20))
```

New in version 1.4.1.

close

If True, the line will be closed.

New in version 1.4.1.

dash_length

Property for getting/setting the length of the dashes in the curve

New in version 1.0.8.

dash_offset

Property for getting/setting the offset between the dashes in the curve

New in version 1.0.8.

dashes

Property for getting/setting dashes.

List of [ON length, offset, ON length, offset, ...]. E.g. [2,4,1,6,8,2] would create a line with the first dash length 2 then an offset of 4 then a dash length of 1 then an offset of 6 and so on.

New in version 1.11.0.

ellipse

Use this property to build an ellipse, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (x, y, width, height, angle_start, angle_end, segments):

- x and y represent the bottom left of the ellipse
- width and height represent the size of the ellipse
- (optional) angle_start and angle_end are in degree. The default value is 0 and 360.
- (optional) segments is the precision of the ellipse. The default value is calculated from the range between angle.

Note that it's up to you to *close* the ellipse or not.

For example, for building a simple ellipse, in python:

```
# simple ellipse
Line(ellipse=(0, 0, 150, 150))

# only from 90 to 180 degrees
Line(ellipse=(0, 0, 150, 150, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(ellipse=(0, 0, 150, 150, 90, 180, 20))
```

New in version 1.4.1.

joint

Determine the join of the line, defaults to 'round'. Can be one of 'none', 'round', 'bevel', 'miter'.

New in version 1.4.1.

joint_precision

Number of iteration for drawing the "round" joint, defaults to 10. The joint_precision must be at least 1.

New in version 1.4.1.

points

Property for getting/settings points of the line

Warning: This will always reconstruct the whole graphics from the new points list. It can be very CPU expensive.

rectangle

Use this property to build a rectangle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of (x, y, width, height):

- x and y represent the bottom-left position of the rectangle
- width and height represent the size

The line is automatically closed.

Usage:

```
Line(rectangle=(0, 0, 200, 200))
```

New in version 1.4.1.

rounded_rectangle

Use this property to build a rectangle, without calculating the *points*. You can only set this property, not get it.

The argument must be a tuple of one of the following forms:

- (x, y, width, height, corner_radius)
- (x, y, width, height, corner_radius, resolution)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4, resolution)
- x and y represent the bottom-left position of the rectangle
- width and height represent the size
- corner_radius is the number of pixels between two borders and the center of the circle arc joining them
- resolution is the number of line segment that will be used to draw the circle arc at each corner (defaults to 30)

The line is automatically closed.

Usage:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40, 100))
```

New in version 1.9.0.

width

Determine the width of the line, defaults to 1.0.

New in version 1.4.1.

class `kivy.graphics.vertex_instructions.Point(**kwargs)`

Bases: `kivy.graphics.instructions.VertexInstruction`

width/height of 2 times the *pointsize*.

Parameters

points: **list** List of points in the format (x1, y1, x2, y2...), where each pair of coordinates specifies the center of a new point.

pointsize: **float, defaults to 1.** The size of the point, measured from the center to the edge. A value of 1.0 therefore means the real size will be 2.0 x 2.0.

Warning: Starting from version 1.0.7, vertex instruction have a limit of 65535 vertices (indices of vertex to be accurate). 2 entries in the list (x, y) will be converted to 4 vertices. So the limit inside Point() class is $2^{15}-2$.

add_point(*float x, float y*)

Add a point to the current *points* list.

If you intend to add multiple points, prefer to use this method instead of reassigning a new *points* list. Assigning a new *points* list will recalculate and reupload the whole buffer into the GPU. If you use `add_point`, it will only upload the changes.

points

Property for getting/settings the center points in the points list. Each pair of coordinates specifies the center of a new point.

pointsize

Property for getting/setting point size. The size is measured from the center to the edge, so a value of 1.0 means the real size will be 2.0 x 2.0.

class kivy.graphics.vertex_instructions.**Mesh**(**kwargs)
Bases: *kivy.graphics.instructions.VertexInstruction*

In OpenGL ES 2.0 and in our graphics implementation, you cannot have more than 65535 indices.

A list of vertices is described as:

```
vertices = [x1, y1, u1, v1, x2, y2, u2, v2, ...]
            |         |         |         |
            +----- i1 -----+ +----- i2 -----+
```

If you want to draw a triangle, add 3 vertices. You can then make an indices list as follows:

```
indices = [0, 1, 2]
```

New in version 1.1.0.

Parameters

vertices: **iterable** List of vertices in the format (x1, y1, u1, v1, x2, y2, u2, v2...).

indices: **iterable** List of indices in the format (i1, i2, i3...).

mode: **str** Mode of the vbo. Check *mode* for more information. Defaults to 'points'.

fnt: **list** The format for vertices, by default, each vertex is described by 2D coordinates (x, y) and 2D texture coordinate (u, v). Each element of the list should be a tuple or list, of the form

(variable_name, size, type)

which will allow mapping vertex data to the glsl instructions.

[(b'v_pos', 2, 'float'), (b'v_tc', 2, 'float'),]

will allow using

attribute vec2 v_pos; attribute vec2 v_tc;

in glsl's vertex shader.

Changed in version 1.8.1: Before, *vertices* and *indices* would always be converted to a list, now, they are only converted to a list if they do not implement the buffer interface. So e.g. numpy arrays, python arrays etc. are used in place, without creating any additional copies. However, the buffers cannot be readonly (even though they are not changed, due to a cython limitation) and must be contiguous in memory.

Note: When passing a memoryview or a instance that implements the buffer interface, *vertices* should be a buffer of floats ('f' code in python array) and *indices* should be a buffer of unsigned short ('H' code in python array). Arrays in other formats will still have to be converted internally, negating any potential gain.

indices

Vertex indices used to specify the order when drawing the mesh.

mode

VBO Mode used for drawing vertices/indices. Can be one of 'points', 'line_strip', 'line_loop', 'lines', 'triangles', 'triangle_strip' or 'triangle_fan'.

vertices

List of x, y, u, v coordinates used to construct the Mesh. Right now, the Mesh instruction doesn't allow you to change the format of the vertices, which means it's only x, y + one texture coordinate.

exception `kivy.graphics.vertex_instructions.GraphicException`

Bases: `Exception`

Exception raised when a graphics error is fired.

class `kivy.graphics.vertex_instructions.Bezier(**kwargs)`

Bases: `kivy.graphics.instructions.VertexInstruction`

New in version 1.0.8.

Parameters

points: `list` List of points in the format (x1, y1, x2, y2...)

segments: `int`, **defaults to 180** Define how many segments are needed for drawing the curve. The drawing will be smoother if you have many segments.

loop: `bool`, **defaults to False** Set the bezier curve to join the last point to the first.

dash_length: `int` Length of a segment (if dashed), defaults to 1.

dash_offset: `int` Distance between the end of a segment and the start of the next one, defaults to 0. Changing this makes it dashed.

dash_length

Property for getting/setting the length of the dashes in the curve.

dash_offset

Property for getting/setting the offset between the dashes in the curve.

points

Property for getting/setting the points of the triangle.

Warning: This will always reconstruct the whole graphic from the new points list. It can be very CPU intensive.

segments

Property for getting/setting the number of segments of the curve.

class `kivy.graphics.vertex_instructions.SmoothLine(**kwargs)`

Bases: `kivy.graphics.vertex_instructions.Line`

results. It has few drawbacks:

- drawing a line with alpha will probably not have the intended result if the line crosses itself.
- `cap`, `joint` and `dash` properties are not supported.
- it uses a custom texture with a premultiplied alpha.
- lines under 1px in width are not supported: they will look the same.

Warning: This is an unfinished work, experimental, and subject to crashes.

New in version 1.9.0.

overdraw_width

Determine the overdraw width of the line, defaults to 1.2.

premultiplied_texture()

Input management

Our input system is wide and simple at the same time. We are currently able to natively support :

- Windows multitouch events (pencil and finger)
- OS X touchpads
- Linux multitouch events (kernel and mtdev)
- Linux wacom drivers (pencil and finger)
- TUIO

All the input management is configurable in the Kivy *config*. You can easily use many multitouch devices in one Kivy application.

When the events have been read from the devices, they are dispatched through a post processing module before being sent to your application. We also have several default modules for :

- Double tap detection
- Decreasing jittering
- Decreasing the inaccuracy of touch on “bad” DIY hardware
- Ignoring regions

class kivy.input.MotionEvent(*device, id, args*)

Bases: *kivy.input.motionevent.MotionEvent*

Abstract class that represents an input event (touch or non-touch).

Parameters

id: **str** unique ID of the MotionEvent

args: **list** list of parameters, passed to the *depack()* function

apply_transform_2d(*transform*)

Apply a transformation on x, y, z, px, py, pz, ox, oy, oz, dx, dy, dz

copy_to(*to*)

Copy some attribute to another touch object.

unpack(*args*)

Depack *args* into attributes of the class

device = None

Device used for creating this touch

distance(*other_touch*)

Return the distance between the current touch and another touch.

double_tap_time = None

If the touch is a *is_double_tap*, this is the time between the previous tap and the current touch.

dpos

Return delta between last position and current position, in the screen coordinate system (self.dx, self.dy)

dsx = None

Delta between self.sx and self.psx, in 0-1 range.

dsy = None

Delta between self.sy and self.psy, in 0-1 range.

dsz = None

Delta between self.sz and self.psz, in 0-1 range.

dx = None

Delta between self.x and self.px, in window range

dy = None

Delta between self.y and self.py, in window range

dz = None

Delta between self.z and self.pz, in window range

grab(*class_instance*, *exclusive=False*)

Grab this motion event. You can grab a touch if you want to receive subsequent *on_touch_move()* and *on_touch_up()* events, even if the touch is not dispatched by the parent:

```
def on_touch_down(self, touch):
    touch.grab(self)

def on_touch_move(self, touch):
    if touch.grab_current is self:
        # I received my grabbed touch
    else:
        # it's a normal touch

def on_touch_up(self, touch):
    if touch.grab_current is self:
        # I receive my grabbed touch, I must ungrab it!
        touch.ungrab(self)
    else:
        # it's a normal touch
        pass
```

grab_current = None

Used to determine which widget the touch is being dispatched to. Check the *grab()* function for more information.

id = None

Id of the touch, not uniq. This is generally the Id set by the input provider, like ID in TUIO. If

you have multiple TUIO source, the same id can be used. Prefer to use *uid* attribute instead.

is_double_tap = None

Indicate if the touch is a double tap or not

is_mouse_scrolling

Returns True if the touch is a mousewheel scrolling

New in version 1.6.0.

is_touch = None

True if the Motion Event is a Touch. Can be also verified is *pos* is *profile*.

is_triple_tap = None

Indicate if the touch is a triple tap or not

New in version 1.7.0.

move(*args*)

Move the touch to another position

opos

Return the initial position of the touch in the screen coordinate system (self.ox, self.oy)

osx = None

Origin X position, in 0-1 range.

osy = None

Origin Y position, in 0-1 range.

osz = None

Origin Z position, in 0-1 range.

ox = None

Origin X position, in window range

oy = None

Origin Y position, in window range

oz = None

Origin Z position, in window range

pop()

Pop attributes values from the stack

pos = None

Position (X, Y), in window range

ppos

Return the previous position of the touch in the screen coordinate system (self.px, self.py)

profile = None

Profiles currently used in the touch

psx = None

Previous X position, in 0-1 range.

psy = None

Previous Y position, in 0-1 range.

psz = None

Previous Z position, in 0-1 range.

push(*attrs=None*)

Push attribute values in *attrs* onto the stack

push_attrs_stack = None
 Attributes to push by default, when we use *push()* : x, y, z, dx, dy, dz, ox, oy, oz, px, py, pz.

px = None
 Previous X position, in window range

py = None
 Previous Y position, in window range

pz = None
 Previous Z position, in window range

scale_for_screen(*w, h, p=None, rotation=0, smode='None', kheight=0*)
 Scale position for the screen

shape = None
 Shape of the touch, subclass of *Shape*. By default, the property is set to None

spos
 Return the position in the 0-1 coordinate system (self.sx, self.sy)

sx = None
 X position, in 0-1 range

sy = None
 Y position, in 0-1 range

sz = None
 Z position, in 0-1 range

time_end = None
 Time of the end event (last touch usage)

time_start = None
 Initial time of the touch creation

time_update = None
 Time of the last update

triple_tap_time = None
 If the touch is a *is_triple_tap*, this is the time between the first tap and the current touch.
 New in version 1.7.0.

ud = None
 User data dictionary. Use this dictionary to save your own data on the touch.

uid = None
 Uniq ID of the touch. You can safely use this property, it will be never the same accross all existing touches.

ungrab(*class_instance*)
 Ungrab a previously grabbed touch

x = None
 X position, in window range

y = None
 Y position, in window range

z = None
 Z position, in window range

class `kivy.input.MotionEventProvider`(*device, args*)
 Bases: `builtins.object`
 Base class for a provider.

start()

Start the provider. This method is automatically called when the application is started and if the configuration uses the current provider.

stop()

Stop the provider.

update(*dispatch_fn*)

Update the provider and dispatch all the new touch events though the *dispatch_fn* argument.

class kivy.input.MotionEventFactory

Bases: `builtins.object`

MotionEvent factory is a class that registers all availables input factories. If you create a new input factory, you need to register it here:

```
MotionEventFactory.register('myproviderid', MyInputProvider)
```

static get(*name*)

Get a provider class from the provider id

static list()

Get a list of all available providers

static register(*name*, *classname*)

Register a input provider in the database

26.1 Input Postprocessing

26.1.1 Calibration

New in version 1.9.0.

Recalibrate input device to a specific range / offset.

Let's say you have 3 1080p displays, the 2 firsts are multitouch. By default, both will have mixed touch, the range will conflict with each others: the 0-1 range will goes to 0-5760 px (remember, $3 * 1920 = 5760$.)

To fix it, you need to manually reference them. For example:

```
[input]
left = mtdev,/dev/input/event17
middle = mtdev,/dev/input/event15
# the right screen is just a display.
```

Then, you can use the calibration postproc module:

```
[postproc:calibration]
left = xratio=0.3333
middle = xratio=0.3333,xoffset=0.3333
```

Now, the touches from the left screen will be within 0-0.3333 range, and the touches from the middle screen will be within 0.3333-0.6666 range.

You can also match calibration rules to devices based on their provider type. This is useful when `probesysfs` is used to match devices. For example:

```
[input]
mtdev_%(name)s = probesysfs,provider=mtdev
```

Then to apply calibration to any mtdev device, you can assign rules to the provider name enclosed by parentheses:

```
[postproc:calibration]
(mtdev) = xratio=0.3333,xoffset=0.3333
```

Calibrating devices like this means the device's path doesn't need to be configured ahead of time. Note that with this method, all mtdev inputs will have the same calibration applied to them. For this reason, matching by provider will typically be useful when expecting only one input device.

class kivy.input.postproc.calibration.**InputPostprocCalibration**

Bases: `builtins.object`

Recalibrate the inputs.

The configuration must go within a section named *postproc:calibration*. Within the section, you must have a line like:

```
devicename = param=value,param=value
```

If you wish to match by provider, you must have a line like:

```
(provider) = param=value,param=value
```

Parameters

xratio: float Value to multiply X

yratio: float Value to multiply Y

xoffset: float Value to add to X

yoffset: float Value to add to Y

auto: str If set, then the touch is transformed from screen-relative to window-relative. The value is used as an indication of screen size, e.g for fullHD:

auto=1920x1080

If present, this setting overrides all the others. This assumes the input device exactly covers the display area, if they are different, the computations will be wrong.

Changed in version 1.11.0: Added *auto* parameter

26.1.2 Dejitter

Prevent blob jittering.

A problem that is often faced (esp. in optical MT setups) is that of jitterish BLOBs caused by bad camera characteristics. With this module you can get rid of that jitter. You just define a threshold *jitter_distance* in your config, and all touch movements that move the touch by less than the jitter distance are considered 'bad' movements caused by jitter and will be discarded.

class kivy.input.postproc.dejitter.**InputPostprocDejitter**

Bases: `builtins.object`

Get rid of jitterish BLOBs. Example:

```
[postproc]
jitter_distance = 0.004
jitter_ignore_devices = mouse,mactouch
```

Configuration

jitter_distance: **float** A float in range 0-1.

jitter_ignore_devices: **string** A comma-separated list of device identifiers that should not be processed by dejitter (because they're very precise already).

26.1.3 Double Tap

Search touch for a double tap

class kivy.input.postproc.doubletap.**InputPostprocDoubleTap**

Bases: `builtins.object`

`InputPostprocDoubleTap` is a post-processor to check if a touch is a double tap or not. Double tap can be configured in the Kivy config file:

```
[postproc]
double_tap_time = 250
double_tap_distance = 20
```

Distance parameter is in the range 0-1000 and time is in milliseconds.

find_double_tap(*ref*)

Find a double tap touch within `self.touches`. The touch must be not a previous double tap and the distance must be within the specified threshold. Additionally, the touch profiles must be the same kind of touch.

26.1.4 Ignore list

Ignore touch on some areas of the screen

class kivy.input.postproc.ignorelist.**InputPostprocIgnoreList**

Bases: `builtins.object`

`InputPostprocIgnoreList` is a post-processor which removes touches in the Ignore list. The Ignore list can be configured in the Kivy config file:

```
[postproc]
# Format: [(xmin, ymin, xmax, ymax), ...]
ignore = [(0.1, 0.1, 0.15, 0.15)]
```

The Ignore list coordinates are in the range 0-1, not in screen pixels.

26.1.5 Retain Touch

Reuse touch to counter lost finger behavior

class kivy.input.postproc.retaintouch.**InputPostprocRetainTouch**

Bases: `builtins.object`

`InputPostprocRetainTouch` is a post-processor to delay the 'up' event of a touch, to reuse it under certain conditions. This module is designed to prevent lost finger touches on some hardware/setups.

Retain touch can be configured in the Kivy config file:

```
[postproc]
retain_time = 100
retain_distance = 50
```

The distance parameter is in the range 0-1000 and time is in milliseconds.

26.1.6 Triple Tap

New in version 1.7.0.

Search touch for a triple tap

class `kivy.input.postproc.tripletap.InputPostprocTripleTap`

Bases: `builtins.object`

`InputPostProcTripleTap` is a post-processor to check if a touch is a triple tap or not. Triple tap can be configured in the Kivy config file:

```
[postproc]
triple_tap_time = 250
triple_tap_distance = 20
```

The distance parameter is in the range 0-1000 and time is in milliseconds.

find_triple_tap(*ref*)

Find a triple tap touch within *self.touches*. The touch must be not be a previous triple tap and the distance must be within the bounds specified. Additionally, the touch profile must be the same kind of touch.

26.2 Providers

26.2.1 Android Joystick Input Provider

This module is based on the PyGame JoyStick Input Provider. For more information, please refer to <http://www.pygame.org/docs/ref/joystick.html>

class `kivy.input.providers.androidjoystick.AndroidMotionEventProvider`(*device*,
args)

Bases: `kivy.input.provider.MotionEventProvider`

start()

Start the provider. This method is automatically called when the application is started and if the configuration uses the current provider.

stop()

Stop the provider.

update(*dispatch_fn*)

Update the provider and dispatch all the new touch events though the *dispatch_fn* argument.

26.2.2 Auto Create Input Provider Config Entry for Available MT Hardware (linux only).

Thanks to Marc Tardif for the probing code, taken from scan-for-mt-device.

The device discovery is done by this provider. However, the reading of input can be performed by other providers like: `hidinput`, `mtdev` and `linuxwacom`. `mtdev` is used prior to other providers. For more information about `mtdev`, check [mtdev](#).

Here is an example of auto creation:


```
[input]
# using mtdev
device_%(name)s = probesysfs,provider=mtdev
# using hidinput
device_%(name)s = probesysfs,provider=hidinput
# using mtdev with a match on name
device_%(name)s = probesysfs,provider=mtdev,match=acer

# using hidinput with custom parameters to hidinput (all on one line)
%(name)s = probesysfs,
    provider=hidinput,param=min_pressure=1,param=max_pressure=99

# you can also match your wacom touchscreen
touch = probesysfs,match=E3 Finger,provider=linuxwacom,
    select_all=1,param=mode=touch
# and your wacom pen
pen = probesysfs,match=E3 Pen,provider=linuxwacom,
    select_all=1,param=mode=pen
```

By default, ProbeSysfs module will enumerate hardware from the `/sys/class/input` device, and configure hardware with `ABS_MT_POSITION_X` capability. But for example, the wacom screen doesn't support this capability. You can prevent this behavior by putting `select_all=1` in your config line. Add `use_mouse=1` to also include touchscreen hardware that offers core pointer functionality.

26.2.3 Common definitions for a Windows provider

This file provides common definitions for constants used by `WM_Touch` / `WM_Pen`.

26.2.4 Leap Motion - finger only

class `kivy.input.providers.leapfinger.LeapFingerEventProvider`(*device, args*)

Bases: `kivy.input.provider.MotionEventProvider`

start()

Start the provider. This method is automatically called when the application is started and if the configuration uses the current provider.

update(*dispatch_fn*)

Update the provider and dispatch all the new touch events though the *dispatch_fn* argument.

class `kivy.input.providers.leapfinger.LeapFingerEvent`(*device, id, args*)

Bases: `kivy.input.motionEvent.MotionEvent`

unpack(*args*)

Depack *args* into attributes of the class

26.2.5 Mouse provider implementation

On linux systems, the mouse provider can be annoying when used with another multitouch provider (hidinput or mtdev). The Mouse can conflict with them: a single touch can generate one event from the mouse provider and another from the multitouch provider.

To avoid this behavior, you can activate the “`disable_on_activity`” token in the mouse configuration. Then, if any touches are created by another provider, the mouse event will be discarded. Add this to your configuration:

```
[input]
mouse = mouse,disable_on_activity
```

Using multitouch interaction with the mouse

New in version 1.3.0.

By default, the middle and right mouse buttons, as well as a combination of ctrl + left mouse button are used for multitouch emulation. If you want to use them for other purposes, you can disable this behavior by activating the “disable_multitouch” token:

```
[input]
mouse = mouse,disable_multitouch
```

Changed in version 1.9.0.

You can now selectively control whether a click initiated as described above will emulate multi-touch. If the touch has been initiated in the above manner (e.g. right mouse button), a *multitouch_sim* value will be added to the touch’s profile, and a *multitouch_sim* property will be added to the touch. By default, *multitouch_sim* is True and multitouch will be emulated for that touch. If, however, *multitouch_on_demand* is added to the config:

```
[input]
mouse = mouse,multitouch_on_demand
```

then *multitouch_sim* defaults to *False*. In that case, if *multitouch_sim* is set to True before the mouse is released (e.g. in on_touch_down/move), the touch will simulate a multi-touch event. For example:

```
if 'multitouch_sim' in touch.profile:
    touch.multitouch_sim = True
```

Following is a list of the supported values for the *profile* property list.

Profile value	Description
button	Mouse button (one of <i>left</i> , <i>right</i> , <i>middle</i> , <i>scrollup</i> or <i>scrolldown</i>). Accessed via the ‘button’ property.
pos	2D position. Also reflected in the <i>x</i> , <i>y</i> and <i>pos</i> properties.
multi-touch_sim	Specifies whether multitouch is simulated or not. Accessed via the ‘multitouch_sim’ property.

class kivy.input.providers.mouse.**MouseMotionEventProvider**(*device, args*)

Bases: *kivy.input.provider.MotionEventProvider*

start()

Start the mouse provider

stop()

Stop the mouse provider

update(*dispatch_fn*)

Update the mouse provider (pop event from the queue)

26.2.6 Native support for HID input from the linux kernel

Support starts from 2.6.32-ubuntu, or 2.6.34.

To configure HIDInput, add this to your configuration:

```
[input]
# devicename = hidinput,/dev/input/eventXX
# example with Stantum MTP4.3" screen
stantum = hidinput,/dev/input/event2
```

Note: You must have read access to the input event.

You can use a custom range for the X, Y and pressure values. For some drivers, the range reported is invalid. To fix that, you can add these options to the argument line:

- `invert_x` : 1 to invert X axis
- `invert_y` : 1 to invert Y axis
- `min_position_x` : X minimum
- `max_position_x` : X maximum
- `min_position_y` : Y minimum
- `max_position_y` : Y maximum
- `min_pressure` : pressure minimum
- `max_pressure` : pressure maximum
- `rotation` : rotate the input coordinate (0, 90, 180, 270)

For example, on the Asus T101M, the touchscreen reports a range from 0-4095 for the X and Y values, but the real values are in a range from 0-32768. To correct this, you can add the following to the configuration:

```
[input]
t101m = hidinput,/dev/input/event7,max_position_x=32768,max_position_y=32768
```

New in version 1.9.1: *rotation* configuration token added.

class `kivy.input.providers.hidinput.HIDMotionEvent`(*device, id, args*)

Bases: `kivy.input.motionEvent.MotionEvent`

depack(*args*)

Depack *args* into attributes of the class

26.2.7 Native support for Multitouch devices on Linux, using libmtdev.

The Mtdev project is a part of the Ubuntu Maverick multitouch architecture. You can read more on <http://wiki.ubuntu.com/Multitouch>

To configure MTDev, it's preferable to use `probesysfs` providers. Check *probesysfs* for more information.

Otherwise, add this to your configuration:

```
[input]
# devicename = hidinput,/dev/input/eventXX
acert230h = mtdev,/dev/input/event2
```

Note: You must have read access to the input event.

You can use a custom range for the X, Y and pressure values. On some drivers, the range reported is invalid. To fix that, you can add these options to the argument line:

- `invert_x` : 1 to invert X axis
- `invert_y` : 1 to invert Y axis
- `min_position_x` : X minimum
- `max_position_x` : X maximum
- `min_position_y` : Y minimum
- `max_position_y` : Y maximum
- `min_pressure` : pressure minimum
- `max_pressure` : pressure maximum
- `min_touch_major` : width shape minimum
- `max_touch_major` : width shape maximum
- `min_touch_minor` : width shape minimum
- `max_touch_minor` : height shape maximum
- `rotation` : 0,90,180 or 270 to rotate

class `kivy.input.providers.mtdev.MTDMotionEvent`(*device, id, args*)

Bases: `kivy.input.motionEvent.MotionEvent`

depack(*args*)

Depack *args* into attributes of the class

26.2.8 Native support of MultitouchSupport framework for MacBook (MaxOSX platform)

class `kivy.input.providers.mactouch.MacMotionEventProvider`(**largs, **kwargs*)

Bases: `kivy.input.provider.MotionEventProvider`

start()

Start the provider. This method is automatically called when the application is started and if the configuration uses the current provider.

stop()

Stop the provider.

update(*dispatch_fn*)

Update the provider and dispatch all the new touch events though the *dispatch_fn* argument.

26.2.9 Native support of Wacom tablet from linuxwacom driver

To configure LinuxWacom, add this to your configuration:

```
[input]
pen = linuxwacom,/dev/input/event2,mode=pen
finger = linuxwacom,/dev/input/event3,mode=touch
```

Note: You must have read access to the input event.

You can use a custom range for the X, Y and pressure values. On some drivers, the range reported is invalid. To fix that, you can add these options to the argument line:

- `invert_x` : 1 to invert X axis
- `invert_y` : 1 to invert Y axis
- `min_position_x` : X minimum
- `max_position_x` : X maximum
- `min_position_y` : Y minimum
- `max_position_y` : Y maximum
- `min_pressure` : pressure minimum
- `max_pressure` : pressure maximum

class `kivy.input.providers.linuxwacom.LinuxWacomMotionEvent` (*device, id, args*)

Bases: `kivy.input.motionevent.MotionEvent`

depack (*args*)

Depack *args* into attributes of the class

26.2.10 Support for WM_PEN messages (Windows platform)

class `kivy.input.providers.wm_pen.WM_Pen` (*device, id, args*)

Bases: `kivy.input.motionevent.MotionEvent`

MotionEvent representing the WM_Pen event. Supports the pos profile.

depack (*args*)

Depack *args* into attributes of the class

26.2.11 Support for WM_TOUCH messages (Windows platform)

class `kivy.input.providers.wm_touch.WM_MotionEvent` (*device, id, args*)

Bases: `kivy.input.motionevent.MotionEvent`

MotionEvent representing the WM_MotionEvent event. Supports pos, shape and size profiles.

depack (*args*)

Depack *args* into attributes of the class

26.2.12 TUIO Input Provider

TUIO is the de facto standard network protocol for the transmission of touch and fiducial information between a server and a client. To learn more about TUIO (which is itself based on the OSC protocol), please refer to <http://tuio.org> – The specification should be of special interest.

Configure a TUIO provider in the `config.ini`

The TUIO provider can be configured in the configuration file in the `[input]` section:

```
[input]
# name = tuio,<ip>:<port>
multitouchtable = tuio,192.168.0.1:3333
```

Configure a TUIO provider in the App

You must add the provider before your application is run, like this:

```
from kivy.app import App
from kivy.config import Config

class TestApp(App):
    def build(self):
        Config.set('input', 'multitouchscreen1', 'tuio,0.0.0.0:3333')
        # You can also add a second TUIO listener
        # Config.set('input', 'source2', 'tuio,0.0.0.0:3334')
        # Then do the usual things
        # ...
        return
```

```
class kivy.input.providers.tuio.TuioMotionEventProvider(device, args)
    Bases: kivy.input.provider.MotionEventProvider
```

The TUIO provider listens to a socket and handles some of the incoming OSC messages:

- /tuio/2Dcur
- /tuio/2Dobj

You can easily extend the provider to handle new TUIO paths like so:

```
# Create a class to handle the new TUIO type/path
# Replace NEWPATH with the pathname you want to handle
class TuioNEWPATHMotionEvent(MotionEvent):
    def __init__(self, id, args):
        super(TuioNEWPATHMotionEvent, self).__init__(id, args)

    def unpack(self, args):
        # In this method, implement 'unpacking' for the received
        # arguments. you basically translate from TUIO args to Kivy
        # MotionEvent variables. If all you receive are x and y
        # values, you can do it like this:
        if len(args) == 2:
            self.sx, self.sy = args
            self.profile = ('pos', )
            self.sy = 1 - self.sy
            super(TuioNEWPATHMotionEvent, self).unpack(args)

# Register it with the TUIO MotionEvent provider.
# You obviously need to replace the PATH placeholders appropriately.
TuioMotionEventProvider.register('/tuio/PATH', TuioNEWPATHMotionEvent)
```

Note: The class name is of no technical importance. Your class will be associated with the path that you pass to the `register()` function. To keep things simple, you should name your class after the path that it handles, though.

static create(*oscpath, **kwargs*)

Create a touch event from a TUIO path

static register(*oscpath, classname*)

Register a new path to handle in TUIO provider

start()

Start the TUIO provider

```

stop()
    Stop the TUIO provider

static unregister(oscpath, classname)
    Unregister a path to stop handling it in the TUIO provider

update(dispatch_fn)
    Update the TUIO provider (pop events from the queue)

class kivy.input.providers.tuio.Tuio2dCurMotionEvent(device, id, args)
    Bases: kivy.input.providers.tuio.TuioMotionEvent
    A 2dCur TUIO touch.

    depack(args)
        Depack args into attributes of the class

class kivy.input.providers.tuio.Tuio2dObjMotionEvent(device, id, args)
    Bases: kivy.input.providers.tuio.TuioMotionEvent
    A 2dObj TUIO object.

    depack(args)
        Depack args into attributes of the class

```

26.3 Input recorder

New in version 1.1.0.

Warning: This part of Kivy is still experimental and this API is subject to change in a future version.

This is a class that can record and replay some input events. This can be used for test cases, screen savers etc.

Once activated, the recorder will listen for any input event and save its properties in a file with the delta time. Later, you can play the input file: it will generate fake touch events with the saved properties and dispatch it to the event loop.

By default, only the position is saved ('pos' profile and 'sx', 'sy', attributes). Change it only if you understand how input handling works.

26.3.1 Recording events

The best way is to use the "recorder" module. Check the [Modules](#) documentation to see how to activate a module.

Once activated, you can press F8 to start the recording. By default, events will be written to <current-path>/recorder.kvi. When you want to stop recording, press F8 again.

You can replay the file by pressing F7.

Check the [Recorder module](#) module for more information.

26.3.2 Manual play

You can manually open a recorder file, and play it by doing:

```
from kivy.input.recorder import Recorder

rec = Recorder(filename='myrecorder.kvi')
rec.play = True
```

If you want to loop over that file, you can do:

```
from kivy.input.recorder import Recorder

def recorder_loop(instance, value):
    if value is False:
        instance.play = True

rec = Recorder(filename='myrecorder.kvi')
rec.bind(play=recorder_loop)
rec.play = True
```

26.3.3 Recording more attributes

You can extend the attributes to save on one condition: attributes values must be simple values, not instances of complex classes.

Let's say you want to save the angle and pressure of the touch, if available:

```
from kivy.input.recorder import Recorder

rec = Recorder(filename='myrecorder.kvi',
    record_attrs=['is_touch', 'sx', 'sy', 'angle', 'pressure'],
    record_profile_mask=['pos', 'angle', 'pressure'])
rec.record = True
```

Or with modules variables:

```
$ python main.py -m recorder,attrs=is_touch:sx:sy:angle:pressure,
→profile_mask=pos:angle:pressure
```

26.3.4 Known limitations

- Unable to save attributes with instances of complex classes.
- Values that represent time will not be adjusted.
- Can replay only complete records. If a begin/update/end event is missing, this could lead to ghost touches.
- Stopping the replay before the end can lead to ghost touches.

class kivy.input.recorder.**Recorder**(**kwargs)
Bases: *kivy.event.EventDispatcher*

Recorder class. Please check module documentation for more information.

Events

on_stop: Fired when the playing stops.

Changed in version 1.10.0: Event *on_stop* added.

counter

Number of events recorded in the last session.

counter is a *NumericProperty* and defaults to 0, read-only.

filename

Filename to save the output of the recorder.

filename is a *StringProperty* and defaults to 'recorder.kvi'.

play

Boolean to start/stop the replay of the current file (if it exists).

play is a *BooleanProperty* and defaults to False.

record

Boolean to start/stop the recording of input events.

record is a *BooleanProperty* and defaults to False.

record_attrs

Attributes to record from the motion event.

record_attrs is a *ListProperty* and defaults to ['is_touch', 'sx', 'sy'].

record_profile_mask

Profile to save in the fake motion event when replayed.

record_profile_mask is a *ListProperty* and defaults to ['pos'].

window

Window instance to attach the recorder. If None, it will use the default instance.

window is a *ObjectProperty* and defaults to None.

26.4 Motion Event

The *MotionEvent* is the base class used for events provided by pointing devices (touch and non-touch). This class defines all the properties and methods needed to handle 2D and 3D movements but has many more capabilities.

Note: You never create the *MotionEvent* yourself: this is the role of the *providers*.

26.4.1 Motion Event and Touch

We differentiate between a Motion Event and Touch event. A Touch event is a *MotionEvent* with the *pos* profile. Only these events are dispatched throughout the widget tree.

1. The *MotionEvent* 's are gathered from input providers.
2. All the *MotionEvent* 's are dispatched from *on_motion()*.
3. If a *MotionEvent* has a *pos* profile, we dispatch it through *on_touch_down()*, *on_touch_move()* and *on_touch_up()*.

26.4.2 Listening to a Motion Event

If you want to receive all MotionEvents, Touch or not, you can bind the MotionEvent from the Window to your own callback:

```
def on_motion(self, etype, motionevent):
    # will receive all motion events.
    pass

Window.bind(on_motion=on_motion)
```

You can also listen to changes of the mouse position by watching *mouse_pos*.

26.4.3 Profiles

The *MotionEvent* stores device specific information in various properties listed in the *profile*. For example, you can receive a MotionEvent that has an angle, a fiducial ID, or even a shape. You can check the *profile* attribute to see what is currently supported by the MotionEvent provider.

This is a short list of the profile values supported by default. Please check the *MotionEvent.profile* property to see what profile values are available.

Profile value	Description
angle	2D angle. Accessed via the <i>a</i> property.
button	Mouse button ('left', 'right', 'middle', 'scrollup' or 'scrolldown'). Accessed via the <i>button</i> property.
markerid	Marker or Fiducial ID. Accessed via the <i>fid</i> property.
pos	2D position. Accessed via the <i>x</i> , <i>y</i> or <i>pos</i> properties.
pos3d	3D position. Accessed via the <i>x</i> , <i>y</i> or <i>z</i> properties.
pressure	Pressure of the contact. Accessed via the <i>pressure</i> property.
shape	Contact shape. Accessed via the <i>shape</i> property .

If you want to know whether the current *MotionEvent* has an angle:

```
def on_touch_move(self, touch):
    if 'angle' in touch.profile:
        print('The touch angle is', touch.a)
```

If you want to select only the fiducials:

```
def on_touch_move(self, touch):
    if 'markerid' not in touch.profile:
        return
```

class `kivy.input.motionevent.MotionEvent(device, id, args)`

Bases: *kivy.input.motionevent.MotionEvent*

Abstract class that represents an input event (touch or non-touch).

Parameters

id: `str` unique ID of the MotionEvent

args: `list` list of parameters, passed to the `depack()` function

apply_transform_2d(*transform*)

Apply a transformation on *x*, *y*, *z*, *px*, *py*, *pz*, *ox*, *oy*, *oz*, *dx*, *dy*, *dz*

copy_to(*to*)

Copy some attribute to another touch object.

depack(*args*)

Depack *args* into attributes of the class

device = None

Device used for creating this touch

distance(*other_touch*)

Return the distance between the current touch and another touch.

double_tap_time = None

If the touch is a *is_double_tap*, this is the time between the previous tap and the current touch.

dpos

Return delta between last position and current position, in the screen coordinate system (self.dx, self.dy)

dsx = None

Delta between self.sx and self.psx, in 0-1 range.

dsy = None

Delta between self.sy and self.psy, in 0-1 range.

dsz = None

Delta between self.sz and self.psz, in 0-1 range.

dx = None

Delta between self.x and self.px, in window range

dy = None

Delta between self.y and self.py, in window range

dz = None

Delta between self.z and self.pz, in window range

grab(*class_instance*, *exclusive=False*)

Grab this motion event. You can grab a touch if you want to receive subsequent *on_touch_move()* and *on_touch_up()* events, even if the touch is not dispatched by the parent:

```
def on_touch_down(self, touch):
    touch.grab(self)

def on_touch_move(self, touch):
    if touch.grab_current is self:
        # I received my grabbed touch
    else:
        # it's a normal touch

def on_touch_up(self, touch):
    if touch.grab_current is self:
        # I receive my grabbed touch, I must ungrab it!
        touch.ungrab(self)
    else:
        # it's a normal touch
        pass
```

grab_current = None

Used to determine which widget the touch is being dispatched to. Check the *grab()* function for more information.

id = None

Id of the touch, not uniq. This is generally the Id set by the input provider, like ID in TUIO. If you have multiple TUIO source, the same id can be used. Prefer to use *uid* attribute instead.

is_double_tap = None

Indicate if the touch is a double tap or not

is_mouse_scrolling

Returns True if the touch is a mousewheel scrolling

New in version 1.6.0.

is_touch = None

True if the Motion Event is a Touch. Can be also verified is *pos* is *profile*.

is_triple_tap = None

Indicate if the touch is a triple tap or not

New in version 1.7.0.

move(*args*)

Move the touch to another position

opos

Return the initial position of the touch in the screen coordinate system (self.ox, self.oy)

osx = None

Origin X position, in 0-1 range.

osy = None

Origin Y position, in 0-1 range.

osz = None

Origin Z position, in 0-1 range.

ox = None

Origin X position, in window range

oy = None

Origin Y position, in window range

oz = None

Origin Z position, in window range

pop()

Pop attributes values from the stack

pos = None

Position (X, Y), in window range

ppos

Return the previous position of the touch in the screen coordinate system (self.px, self.py)

profile = None

Profiles currently used in the touch

psx = None

Previous X position, in 0-1 range.

psy = None

Previous Y position, in 0-1 range.

psz = None

Previous Z position, in 0-1 range.

push(*attrs=None*)

Push attribute values in *attrs* onto the stack

push_attrs_stack = None

Attributes to push by default, when we use *push()* : x, y, z, dx, dy, dz, ox, oy, oz, px, py, pz.

px = None
Previous X position, in window range

py = None
Previous Y position, in window range

pz = None
Previous Z position, in window range

scale_for_screen(*w, h, p=None, rotation=0, smode='None', kheight=0*)
Scale position for the screen

shape = None
Shape of the touch, subclass of *Shape*. By default, the property is set to None

spos
Return the position in the 0-1 coordinate system (self.sx, self.sy)

sx = None
X position, in 0-1 range

sy = None
Y position, in 0-1 range

sz = None
Z position, in 0-1 range

time_end = None
Time of the end event (last touch usage)

time_start = None
Initial time of the touch creation

time_update = None
Time of the last update

triple_tap_time = None
If the touch is a *is_triple_tap*, this is the time between the first tap and the current touch.
New in version 1.7.0.

ud = None
User data dictionary. Use this dictionary to save your own data on the touch.

uid = None
Uniq ID of the touch. You can safely use this property, it will be never the same accross all existing touches.

ungrab(*class_instance*)
Ungrab a previously grabbed touch

x = None
X position, in window range

y = None
Y position, in window range

z = None
Z position, in window range

26.5 Motion Event Factory

Factory of *MotionEvent* providers.

class kivy.input.factory.**MotionEventFactory**

Bases: `builtins.object`

MotionEvent factory is a class that registers all available input factories. If you create a new input factory, you need to register it here:

```
MotionEventFactory.register('myproviderid', MyInputProvider)
```

static **get**(*name*)

Get a provider class from the provider id

static **list**()

Get a list of all available providers

static **register**(*name, classname*)

Register a input provider in the database

26.6 Motion Event Provider

Abstract class for the implementation of a *MotionEvent* provider. The implementation must support the *start()*, *stop()* and *update()* methods.

class kivy.input.provider.**MotionEventProvider**(*device, args*)

Bases: `builtins.object`

Base class for a provider.

start()

Start the provider. This method is automatically called when the application is started and if the configuration uses the current provider.

stop()

Stop the provider.

update(*dispatch_fn*)

Update the provider and dispatch all the new touch events through the *dispatch_fn* argument.

26.7 Motion Event Shape

Represent the shape of the *MotionEvent*

class kivy.input.shape.**Shape**

Bases: `builtins.object`

Abstract class for all implementations of a shape

class kivy.input.shape.**ShapeRect**

Bases: *kivy.input.shape.Shape*

Class for the representation of a rectangle.

height

Height of the rect

width

Width of the rect

Kivy Language

The Kivy language is a language dedicated to describing user interface and interactions. You could compare this language to Qt's QML (<http://qt.nokia.com>), but we included new concepts such as rule definitions (which are somewhat akin to what you may know from CSS), templating and so on.

Changed in version 1.7.0: The Builder doesn't execute canvas expressions in realtime anymore. It will pack all the expressions that need to be executed first and execute them after dispatching input, just before drawing the frame. If you want to force the execution of canvas drawing, just call *Builder.sync*.

An experimental profiling tool for the kv lang is also included. You can activate it by setting the environment variable `KIVY_PROFILE_LANG=1`. It will then generate an html file named *builder_stats.html*.

27.1 Overview

The language consists of several constructs that you can use:

Rules A rule is similar to a CSS rule. A rule applies to specific widgets (or classes thereof) in your widget tree and modifies them in a certain way. You can use rules to specify interactive behaviour or use them to add graphical representations of the widgets they apply to. You can target a specific class of widgets (similar to the CSS concept of a *class*) by using the `cls` attribute (e.g. `cls=MyTestWidget`).

A Root Widget You can use the language to create your entire user interface. A kv file must contain only one root widget at most.

Dynamic Classes (*introduced in version 1.7.0*) Dynamic classes let you create new widgets and rules on-the-fly, without any Python declaration.

Templates (deprecated) (*introduced in version 1.0.5, deprecated from version 1.7.0*) Templates were used to populate parts of an application, such as styling the content of a list (e.g. icon on the left, text on the right). They are now deprecated by dynamic classes.

27.2 Syntax of a kv File

A Kivy language file must have `.kv` as filename extension.

The content of the file should always start with the Kivy header, where *version* must be replaced with the Kivy language version you're using. For now, use 1.0:

```
#:kivy `1.0`  
  
# content here
```

The *content* can contain rule definitions, a root widget, dynamic class definitions and templates:

```
# Syntax of a rule definition. Note that several Rules can share the same  
# definition (as in CSS). Note the braces: they are part of the definition.  
<Rule1,Rule2>:  
    # .. definitions ..  
  
<Rule3>:  
    # .. definitions ..  
  
# Syntax for creating a root widget  
RootClassName:  
    # .. definitions ..  
  
# Syntax for creating a dynamic class  
<NewWidget@BaseClass>:  
    # .. definitions ..  
  
# Syntax for create a template  
[TemplateName@BaseClass1,BaseClass2]:  
    # .. definitions ..
```

Regardless of whether it's a rule, root widget, dynamic class or template you're defining, the definition should look like this:

```
# With the braces it's a rule. Without them, it's a root widget.  
<ClassName>:  
    prop1: value1  
    prop2: value2  
  
    canvas:  
        CanvasInstruction1:  
            canvasprop1: value1  
        CanvasInstruction2:  
            canvasprop2: value2  
  
    AnotherClass:  
        prop3: value1
```

Here *prop1* and *prop2* are the properties of *ClassName* and *prop3* is the property of *AnotherClass*. If the widget doesn't have a property with the given name, an *ObjectProperty* will be automatically created and added to the widget.

AnotherClass will be created and added as a child of the *ClassName* instance.

- The indentation is important and must be consistent. The spacing must be a multiple of the number of spaces used on the first indented line. Spaces are encouraged: mixing tabs and spaces is not recommended.
- The value of a property must be given on a single line (for now at least).

- Keep class names capitalized to avoid syntax errors.
- The *canvas* property is special: you can put graphics instructions in it to create a graphical representation of the current class.

Here is a simple example of a kv file that contains a root widget:

```
#:kivy 1.0

Button:
    text: 'Hello world'
```

Changed in version 1.7.0: The indentation is not limited to 4 spaces anymore. The spacing must be a multiple of the number of spaces used on the first indented line.

Both the *load_file()* and the *load_string()* methods return the root widget defined in your kv file/string. They will also add any class and template definitions to the *Factory* for later usage.

27.3 Value Expressions, on_property Expressions, ids, and Reserved Keywords

When you specify a property's value, the value is evaluated as a Python expression. This expression can be static or dynamic, which means that the value can use the values of other properties using reserved keywords.

self The keyword self references the “current widget instance”:

```
Button:
    text: 'My state is %s' % self.state
```

root This keyword is available only in rule definitions and represents the root widget of the rule (the first instance of the rule):

```
<MyWidget>:
    custom: 'Hello world'
    Button:
        text: root.custom
```

app This keyword always refers to your app instance. It's equivalent to a call to *kivy.app.App.get_running_app()* in Python.

```
Label:
    text: app.name
```

args This keyword is available in on_<action> callbacks. It refers to the arguments passed to the callback.

```
TextInput:
    on_focus: self.insert_text("Focus" if args[1] else "No focus")
```

27.3.1 ids

Class definitions may contain ids which can be used as a keywords::

```
<MyWidget>:
    Button:
        id: btn1
    Button:
        text: 'The state of the other button is %s' % btn1.state
```

Please note that the *id* will not be available in the widget instance: it is used exclusively for external references. *id* is a weakref to the widget, and not the widget itself. The widget itself can be accessed with *<id>.__self__* (*btn1.__self__* in this case).

When the kv file is processed, weakrefs to all the widgets tagged with ids are added to the root widget's *ids* dictionary. In other words, following on from the example above, the buttons state could also be accessed as follows:

```
widget = MyWidget()
state = widget.ids["btn1"].state

# Or, as an alternative syntax,
state = widget.ids.btn1.state
```

Note that the outermost widget applies the kv rules to all its inner widgets before any other rules are applied. This means if an inner widget contains ids, these ids may not be available during the inner widget's *__init__* function.

27.3.2 Valid expressions

There are two places that accept python statements in a kv file: after a property, which assigns to the property the result of the expression (such as the text of a button as shown above) and after a *on_property*, which executes the statement when the property is updated (such as *on_state*).

In the former case, the **expression** can only span a single line, cannot be extended to multiple lines using newline escaping, and must return a value. An example of a valid expression is **text: self.state** and (**'up' if self.state == 'normal' else 'down'**).

In the latter case, multiple single line statements are valid, including those that escape their newline, as long as they don't add an indentation level.

Examples of valid statements are:

```
on_press: if self.state == 'normal': print('normal')
on_state:
    if self.state == 'normal': print('normal')
    else: print('down')
    if self.state == 'normal': \
        print('multiline normal')
    for i in range(10): print(i)
    print([1,2,3,4,
           5,6,7])
```

An example of a invalid statement:

```
on_state:
    if self.state == 'normal':
        print('normal')
```

27.4 Relation Between Values and Properties

When you use the Kivy language, you might notice that we do some work behind the scenes to automatically make things work properly. You should know that *Properties* implement the *Observer Design Pattern*. That means that you can bind your own function to be called when the value of a property changes (i.e. you passively *observe* the property for potential changes).

The Kivy language detects properties in your *value* expression and will create create callbacks to automatically update the property via your expression when changes occur.

Here's a simple example that demonstrates this behaviour:

```
Button:
    text: str(self.state)
```

In this example, the parser detects that *self.state* is a dynamic value (a property). The *state* property of the button can change at any moment (when the user touches it). We now want this button to display its own state as text, even as the state changes. To do this, we use the state property of the Button and use it in the value expression for the button's *text* property, which controls what text is displayed on the button (We also convert the state to a string representation). Now, whenever the button state changes, the text property will be updated automatically.

Remember: The value is a python expression! That means that you can do something more interesting like:

```
Button:
    text: 'Plop world' if self.state == 'normal' else 'Release me!'
```

The Button text changes with the state of the button. By default, the button text will be 'Plop world', but when the button is being pressed, the text will change to 'Release me!'.

More precisely, the kivy language parser detects all substrings of the form *X.a.b* where *X* is *self* or *root* or *app* or a known id, and *a* and *b* are properties: it then adds the appropriate dependencies to cause the the constraint to be reevaluated whenever something changes. For example, this works exactly as expected:

```
<IndexedExample>:
    beta: self.a.b[self.c.d]
```

However, due to limitations in the parser which hopefully may be lifted in the future, the following doesn't work:

```
<BadExample>:
    beta: self.a.b[self.c.d].e.f
```

indeed the *.e.f* part is not recognized because it doesn't follow the expected pattern, and so, does not result in an appropriate dependency being setup. Instead, an intermediate property should be introduced to allow the following constraint:

```
<GoodExample>:
    alpha: self.a.b[self.c.d]
    beta: self.alpha.e.f
```

27.5 Graphical Instructions

The graphical instructions are a special part of the Kivy language. They are handled by the 'canvas' property definition:

```
Widget:
  canvas:
    Color:
      rgb: (1, 1, 1)
    Rectangle:
      size: self.size
      pos: self.pos
```

All the classes added inside the canvas property must be derived from the *Instruction* class. You cannot put any Widget class inside the canvas property (as that would not make sense because a widget is not a graphics instruction).

If you want to do theming, you'll have the same question as in CSS: which rules have been executed first? In our case, the rules are executed in processing order (i.e. top-down).

If you want to change how Buttons are rendered, you can create your own kv file and add something like this:

```
<Button>:
  canvas:
    Color:
      rgb: (1, 0, 0)
    Rectangle:
      pos: self.pos
      size: self.size
    Rectangle:
      pos: self.pos
      size: self.texture_size
      texture: self.texture
```

This will result in buttons having a red background with the label in the bottom left, in addition to all the preceding rules. You can clear all the previous instructions by using the *Clear* command:

```
<Button>:
  canvas:
    Clear
    Color:
      rgb: (1, 0, 0)
    Rectangle:
      pos: self.pos
      size: self.size
    Rectangle:
      pos: self.pos
      size: self.texture_size
      texture: self.texture
```

Then, only your rules that follow the *Clear* command will be taken into consideration.

27.6 Dynamic classes

Dynamic classes allow you to create new widgets on-the-fly, without any python declaration in the first place. The syntax of the dynamic classes is similar to the Rules, but you need to specify the base classes you want to subclass.

The syntax looks like:

```
# Simple inheritance
<NewWidget@Button>:
    # kv code here ...

# Multiple inheritance
<NewWidget@ButtonBehavior+Label>:
    # kv code here ...
```

The @ character is used to separate your class name from the classes you want to subclass. The Python equivalent would have been:

```
# Simple inheritance
class NewWidget(Button):
    pass

# Multiple inheritance
class NewWidget(ButtonBehavior, Label):
    pass
```

Any new properties, usually added in python code, should be declared first. If the property doesn't exist in the dynamic class, it will be automatically created as an *ObjectProperty* (pre 1.8.0) or as an appropriate typed property (from version 1.8.0).

Changed in version 1.8.0: If the property value is an expression that can be evaluated right away (no external binding), then the value will be used as default value of the property, and the type of the value will be used for the specialization of the Property class. In other terms: if you declare *hello*: "world", a new *StringProperty* will be instantiated, with the default value "world". Lists, tuples, dictionaries and strings are supported.

Let's illustrate the usage of these dynamic classes with an implementation of a basic Image button. We could derive our classes from the Button and just add a property for the image filename:

```
<ImageButton@Button>:
    source: None

    Image:
        source: root.source
        pos: root.pos
        size: root.size

# let's use the new classes in another rule:
<MainUI>:
    BoxLayout:
        ImageButton:
            source: 'hello.png'
            on_press: root.do_something()
        ImageButton:
            source: 'world.png'
            on_press: root.do_something_else()
```

In Python, you can create an instance of the dynamic class as follows:

```
from kivy.factory import Factory
button_inst = Factory.ImageButton()
```

Note: Using dynamic classes, a child class can be declared before its parent. This however, leads to the unintuitive situation where the parent properties/methods override those of the child. Be careful if

you choose to do this.

27.7 Templates

Changed in version 1.7.0: Template usage is now deprecated. Please use Dynamic classes instead.

27.7.1 Syntax of templates

Using a template in Kivy requires 2 things :

1. a context to pass for the context (will be ctx inside template).
2. a kv definition of the template.

Syntax of a template:

```
# With only one base class
[ClassName@BaseClass]:
    # .. definitions ..

# With more than one base class
[ClassName@BaseClass1,BaseClass2]:
    # .. definitions ..
```

For example, for a list, you'll need to create a entry with a image on the left, and a label on the right. You can create a template for making that definition easier to use. So, we'll create a template that uses 2 entries in the context: an image filename and a title:

```
[IconItem@BoxLayout]:
    Image:
        source: ctx.image
    Label:
        text: ctx.title
```

Then in Python, you can instantiate the template using:

```
from kivy.lang import Builder

# create a template with hello world + an image
# the context values should be passed as kwargs to the Builder.template
# function
icon1 = Builder.template('IconItem', title='Hello world',
                        image='myimage.png')

# create a second template with other information
ctx = {'title': 'Another hello world',
       'image': 'myimage2.png'}
icon2 = Builder.template('IconItem', **ctx)
# and use icon1 and icon2 as other widget.
```

27.7.2 Template example

Most of time, when you are creating a screen in the kv lang, you use a lot of redefinitions. In our example, we'll create a Toolbar, based on a BoxLayout, and put in a few *Image* widgets that will react to the *on_touch_down* event.

```

<MyToolbar>:
  BoxLayout:
    Image:
      source: 'data/text.png'
      size: self.texture_size
      size_hint: None, None
      on_touch_down: self.collide_point(*args[1].pos) and root.create_text()

    Image:
      source: 'data/image.png'
      size: self.texture_size
      size_hint: None, None
      on_touch_down: self.collide_point(*args[1].pos) and root.create_image()

    Image:
      source: 'data/video.png'
      size: self.texture_size
      size_hint: None, None
      on_touch_down: self.collide_point(*args[1].pos) and root.create_video()

```

We can see that the size and size_hint attribute are exactly the same. More than that, the callback in on_touch_down and the image are changing. These can be the variable part of the template that we can put into a context. Let's try to create a template for the Image:

```

[ToolbarButton@Image]:

# This is the same as before
size: self.texture_size
size_hint: None, None

# Now, we are using the ctx for the variable part of the template
source: 'data/%s.png' % ctx.image
on_touch_down: self.collide_point(*args[1].pos) and ctx.callback()

```

The template can be used directly in the MyToolbar rule:

```

<MyToolbar>:
  BoxLayout:
    ToolbarButton:
      image: 'text'
      callback: root.create_text
    ToolbarButton:
      image: 'image'
      callback: root.create_image
    ToolbarButton:
      image: 'video'
      callback: root.create_video

```

That's all :)

27.7.3 Template limitations

When you are creating a context:

1. you cannot use references other than "root":

```

<MyRule>:
  Widget:

```

(continues on next page)

(continued from previous page)

```
id: mywidget
value: 'bleh'
Template:
    ctxkey: mywidget.value # << fail, this references the id
    # mywidget
```

2. not all of the dynamic parts will be understood:

```
<MyRule>:
    Template:
        ctxkey: 'value 1' if root.prop1 else 'value2' # << even if
        # root.prop1 is a property, if it changes value, ctxkey
        # will not be updated
```

Template definitions also replace any similarly named definitions in their entirety and thus do not support inheritance.

27.8 Redefining a widget's style

Sometimes we would like to inherit from a widget in order to use its Python properties without also using its .kv defined style. For example, we would like to inherit from a `Label`, but we would also like to define our own canvas instructions instead of automatically using the canvas instructions inherited from the `Label`. We can achieve this by prepending a dash (-) before the class name in the .kv style definition.

In `myapp.py`:

```
class MyWidget(Label):
    pass
```

and in `my.kv`:

```
<-MyWidget>:
    canvas:
        Color:
            rgb: 1, 1, 1
        Rectangle:
            size: (32, 32)
```

`MyWidget` will now have a `Color` and `Rectangle` instruction in its canvas without any of the instructions inherited from the `Label`.

27.9 Redefining a widget's property style

Similar to *redefining style*, sometimes we would like to inherit from a widget, keep all its KV defined styles, except for the style applied to a specific property. For example, we would like to inherit from a `Button`, but we would also like to set our own `state_image`, rather than relying on the `background_normal` and `background_down` values. We can achieve this by prepending a dash (-) before the `state_image` property name in the .kv style definition.

In `myapp.py`:

```
class MyWidget(Button):
    new_background = StringProperty('my_background.png')
```


and in my.kv:

```
<MyWidget>:
    -state_image: self.new_background
```

MyWidget will now have a *state_image* background set only by *new_background*, and not by any previous styles that may have set *state_image*.

Note: Although the previous rules are cleared, they are still applied during widget construction and are only removed when the new rule with the dash is reached. This means that initially, previous rules could be used to set the property.

27.10 Order of kwargs and KV rule application

Properties can be initialized in KV as well as in python. For example, in KV:

```
<MyRule@Widget>:
    text: 'Hello'
    ramp: 45.
    order: self.x + 10
```

Then *MyRule()* would initialize all three kivy properties to the given KV values. Separately in python, if the properties already exist as kivy properties one can do for example *MyRule(line='Bye', side=55)*.

However, what will be the final values of the properties when *MyRule(text='Bye', order=55)* is executed? The quick rule is that python initialization is stronger than KV initialization only for constant rules.

Specifically, the *kwargs* provided to the python initializer are always applied first. So in the above example, *text* is set to 'Bye' and *order* is set to 55. Then, all the KV rules are applied, except those constant rules that overwrite a python initializer provided value.

That is, the KV rules that do not creates bindings such as *text: 'Hello'* and *ramp: 45.*, if a value for that property has been provided in python, then that rule will not be applied.

So in the *MyRule(text='Bye', order=55)* example, *text* will be 'Bye', *ramp* will be 45., and *order*, which creates a binding, will first be set to 55, but then when KV rules are applied will end up being whatever *self.x + 10* is.

Changed in version 1.9.1: Before, KV rules always overwrote the python values, now, python values are not overwritten by constant rules.

27.11 Lang Directives

You can use directives to add declarative commands, such as imports or constant definitions, to the lang files. Directives are added as comments in the following format:

```
#:<directivename> <options>
```

27.11.1 import <package>

New in version 1.0.5.

Syntax:

```
#:import <alias> <package>
```

You can import a package by writing:

```
#:import os os

<Rule>:
    Button:
        text: os.getcwd()
```

Or more complex:

```
#:import ut kivy.utils

<Rule>:
    canvas:
        Color:
            rgba: ut.get_random_color()
```

New in version 1.0.7.

You can directly import classes from a module:

```
#: import Animation kivy.animation.Animation

<Rule>:
    on_prop: Animation(x=.5).start(self)
```

27.11.2 set <key> <expr>

New in version 1.0.6.

Syntax:

```
#:set <key> <expr>
```

Set a key that will be available anywhere in the kv. For example:

```
#:set my_color (.4, .3, .4)
#:set my_color_hl (.5, .4, .5)

<Rule>:
    state: 'normal'
    canvas:
        Color:
            rgb: my_color if self.state == 'normal' else my_color_hl
```

27.11.3 include <file>

New in version 1.9.0.

Syntax:

```
#:include [force] <file>
```

Includes an external kivy file. This allows you to split complex widgets into their own files. If the include is forced, the file will first be unloaded and then reloaded again. For example:

```
# Test.kv
#:include mycomponent.kv
#:include force mybutton.kv
```

```
<Rule>:
    state: 'normal'
    MyButton:
    MyComponent:
```

```
# mycomponent.kv
#:include mybutton.kv
```

```
<MyComponent>:
    MyButton:
```

```
# mybutton.kv
```

```
<MyButton>:
    canvas:
        Color:
            rgb: (1.0, 0.0, 0.0)
        Rectangle:
            pos: self.pos
            size: (self.size[0]/4, self.size[1]/4)
```

class kivy.lang.Observable

Bases: *kivy.event.ObjectWithUid*

Observable is a stub class defining the methods required for binding. *EventDispatcher* is (the) one example of a class that implements the binding interface. See *EventDispatcher* for details.

New in version 1.9.0.

bind(**kwargs)

fbind(name, func, *largs, **kwargs)
See *EventDispatcher.fbind()*.

Note: To keep backward compatibility with derived classes which may have inherited from *Observable* before, the *fbind()* method was added. The default implementation of *fbind()* is to create a partial function that it passes to bind while saving the uid and largs/kwags. However, *funbind()* (and *unbind_uid()*) are fairly inefficient since we have to first lookup this partial function using the largs/kwags or uid and then call *unbind()* on the returned function. It is recommended to overwrite these methods in derived classes to bind directly for better performance.

Similarly to *EventDispatcher.fbind()*, this method returns 0 on failure and a positive unique uid on success. This uid can be used with *unbind_uid()*.

funbind(name, func, *largs, **kwargs)
See *fbind()* and *EventDispatcher.funbind()*.

unbind(**kwargs)

unbind_uid(name, uid)
See *fbind()* and *EventDispatcher.unbind_uid()*.

class kivy.lang.BuilderBase

Bases: `builtins.object`

The Builder is responsible for creating a *Parser* for parsing a kv file, merging the results into its internal rules, templates, etc.

By default, **Builder** is a global Kivy instance used in widgets that you can use to load other kv files in addition to the default ones.

apply(*widget*, *ignored_consts=set()*, *rule_children=None*, *dispatch_kv_post=False*)

Search all the rules that match the widget and apply them.

Parameters

widget: *Widget* The widget whose class rules should be applied to this widget.

ignored_consts: *set* A set or list type whose elements are property names for which constant KV rules (i.e. those that don't create bindings) of that widget will not be applied. This allows e.g. skipping constant rules that overwrite a value initialized in python.

rule_children: *list* If not **None**, it should be a list that will be populated with all the widgets created by the kv rules being applied.

Changed in version 1.11.0.

dispatch_kv_post: *bool* Normally the class *Widget* dispatches the *on_kv_post* event to widgets created during kv rule application. But if the rules are manually applied by calling *apply()*, that may not happen, so if this is *True*, we will dispatch the *on_kv_post* event where needed after applying the rules to *widget* (we won't dispatch it for *widget* itself).

Defaults to **False**.

Changed in version 1.11.0.

apply_rules(*widget*, *rule_name*, *ignored_consts=set()*, *rule_children=None*, *dispatch_kv_post=False*)

Search all the rules that match the name *rule_name* and apply them to *widget*.

New in version 1.10.0.

Parameters

widget: *Widget* The widget to whom the matching rules should be applied to.

ignored_consts: *set* A set or list type whose elements are property names for which constant KV rules (i.e. those that don't create bindings) of that widget will not be applied. This allows e.g. skipping constant rules that overwrite a value initialized in python.

rule_children: *list* If not **None**, it should be a list that will be populated with all the widgets created by the kv rules being applied.

Changed in version 1.11.0.

dispatch_kv_post: *bool* Normally the class *Widget* dispatches the *on_kv_post* event to widgets created during kv rule application. But if the rules are manually applied by calling *apply()*, that may not happen, so if this is *True*, we will dispatch the *on_kv_post* event where needed after applying the rules to *widget* (we won't dispatch it for *widget* itself).

Defaults to **False**.

Changed in version 1.11.0.

classmethod create_from(*builder*)

Creates a instance of the class, and initializes to the state of **builder**.

Parameters **builder** – The builder to initialize from.

Returns A new instance of this class.

load_file(*filename*, ***kwargs*)

Insert a file into the language builder and return the root widget (if defined) of the kv file.

Parameters

rulesonly: bool, defaults to False If True, the Builder will raise an exception if you have a root widget inside the definition.

load_string(*string*, ***kwargs*)

Insert a string into the Language Builder and return the root widget (if defined) of the kv string.

Parameters

rulesonly: bool, defaults to False If True, the Builder will raise an exception if you have a root widget inside the definition.

filename: str, defaults to None If specified, the filename used to index the kv rules.

The filename parameter can be used to unload kv strings in the same way as you unload kv files. This can be achieved using pseudo file names e.g.:

```
Build.load_string("""
    <MyRule>:
        Label:
            text="Hello"
""", filename="myrule.kv")
```

can be unloaded via:

```
Build.unload_file("myrule.kv")
```

match(*widget*)

Return a list of ParserRule objects matching the widget.

match_rule_name(*rule_name*)

Return a list of ParserRule objects matching the widget.

sync()

Execute all the waiting operations, such as the execution of all the expressions related to the canvas.

New in version 1.7.0.

template(**args*, ***ctx*)

Create a specialized template using a specific context.

New in version 1.0.5.

With templates, you can construct custom widgets from a kv lang definition by giving them a context. Check [Template usage](#).

unbind_property(*widget*, *name*)

Unbind the handlers created by all the rules of the widget that set the name.

This effectively clears all the rules of widget that take the form:

```
name: rule
```

For example:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_property(w, 'height')
>>> w.width = 222
>>> w.size
[222, 500]
>>> w.y = 500
>>> w.pos
[550, 500]
```

New in version 1.9.1.

unbind_widget(*uid*)

Unbind all the handlers created by the KV rules of the widget. The `kivy.uix.widget.Widget.uid` is passed here instead of the widget itself, because Builder is using it in the widget destructor.

This effectively clears all the KV rules associated with this widget. For example:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_widget(w.uid)
>>> w.width = 222
>>> w.y = 500
>>> w.size
[222, 500]
>>> w.pos
[50, 500]
```

New in version 1.7.2.

unload_file(*filename*)

Unload all rules associated with a previously imported file.

New in version 1.0.8.

Warning: This will not remove rules or templates already applied/used on current widgets. It will only effect the next widgets creation or template invocation.

exception `kivy.lang.BuilderException(context, line, message, cause=None)`

Bases: `kivy.lang.parser.ParserException`

Exception raised when the Builder failed to apply a rule on a widget.

class `kivy.lang.Parser(**kwargs)`

Bases: `builtins.object`

Create a Parser object to parse a Kivy language file or Kivy content.

parse(*content*)

Parse the contents of a Parser file and return a list of root objects.

parse_level(*level, lines, spaces=0*)

Parse the current level (level * spaces) indentation.

strip_comments(*lines*)

Remove all comments from all lines in-place. Comments need to be on a single line and not at the end of a line. i.e. a comment line's first non-whitespace character must be a #.

exception `kivy.lang.ParserException(context, line, message, cause=None)`

Bases: `Exception`

Exception raised when something wrong happened in a kv file.

27.12 Builder

Class used for the registering and application of rules for specific widgets.

class `kivy.lang.builder.Observable`

Bases: `kivy.event.ObjectWithUid`

Observable is a stub class defining the methods required for binding. `EventDispatcher` is (the) one example of a class that implements the binding interface. See `EventDispatcher` for details.

New in version 1.9.0.

bind(***kwargs*)

fbind(*name, func, *largs, **kwargs*)

See `EventDispatcher.fbind()`.

Note: To keep backward compatibility with derived classes which may have inherited from *Observable* before, the `fbind()` method was added. The default implementation of `fbind()` is to create a partial function that it passes to `bind` while saving the uid and largs/kwarg. However, `funbind()` (and `unbind_uid()`) are fairly inefficient since we have to first lookup this partial function using the largs/kwarg or uid and then call `unbind()` on the returned function. It is recommended to overwrite these methods in derived classes to bind directly for better performance.

Similarly to `EventDispatcher.fbind()`, this method returns 0 on failure and a positive unique uid on success. This uid can be used with `unbind_uid()`.

funbind(*name, func, *largs, **kwargs*)

See `fbind()` and `EventDispatcher.funbind()`.

unbind(**kwargs)

unbind_uid(name, uid)

See **fbind()** and `EventDispatcher.unbind_uid()`.

`kivy.lang.builder.Builder` = <`kivy.lang.builder.BuilderBase` object>

Main instance of a **BuilderBase**.

class `kivy.lang.builder.BuilderBase`

Bases: `builtins.object`

The Builder is responsible for creating a **Parser** for parsing a kv file, merging the results into its internal rules, templates, etc.

By default, **Builder** is a global Kivy instance used in widgets that you can use to load other kv files in addition to the default ones.

apply(widget, ignored_consts=set(), rule_children=None, dispatch_kv_post=False)

Search all the rules that match the widget and apply them.

Parameters

widget: **Widget** The widget whose class rules should be applied to this widget.

ignored_consts: **set** A set or list type whose elements are property names for which constant KV rules (i.e. those that don't create bindings) of that widget will not be applied. This allows e.g. skipping constant rules that overwrite a value initialized in python.

rule_children: **list** If not **None**, it should be a list that will be populated with all the widgets created by the kv rules being applied.

Changed in version 1.11.0.

dispatch_kv_post: **bool** Normally the class *Widget* dispatches the *on_kv_post* event to widgets created during kv rule application. But if the rules are manually applied by calling **apply()**, that may not happen, so if this is **True**, we will dispatch the *on_kv_post* event where needed after applying the rules to *widget* (we won't dispatch it for *widget* itself).

Defaults to False.

Changed in version 1.11.0.

apply_rules(widget, rule_name, ignored_consts=set(), rule_children=None, dispatch_kv_post=False)

Search all the rules that match the name *rule_name* and apply them to *widget*.

New in version 1.10.0.

Parameters

widget: **Widget** The widget to whom the matching rules should be applied to.

ignored_consts: **set** A set or list type whose elements are property names for which constant KV rules (i.e. those that don't create bindings) of that widget will not be applied. This allows e.g. skipping constant rules that overwrite a value initialized in python.

rule_children: **list** If not **None**, it should be a list that will be populated with all the widgets created by the kv rules being applied.

Changed in version 1.11.0.

dispatch_kv_post: **bool** Normally the class *Widget* dispatches the *on_kv_post* event to widgets created during kv rule application. But if the rules are manually applied by calling **apply()**, that may not happen, so if this is **True**, we

will dispatch the *on_kv_post* event where needed after applying the rules to *widget* (we won't dispatch it for *widget* itself).

Defaults to False.

Changed in version 1.11.0.

classmethod `create_from(builder)`

Creates a instance of the class, and initializes to the state of *builder*.

Parameters *builder* – The builder to initialize from.

Returns A new instance of this class.

load_file(*filename*, ***kwargs*)

Insert a file into the language builder and return the root widget (if defined) of the kv file.

Parameters

rulesonly: bool, defaults to False If True, the Builder will raise an exception if you have a root widget inside the definition.

load_string(*string*, ***kwargs*)

Insert a string into the Language Builder and return the root widget (if defined) of the kv string.

Parameters

rulesonly: bool, defaults to False If True, the Builder will raise an exception if you have a root widget inside the definition.

filename: str, defaults to None If specified, the filename used to index the kv rules.

The filename parameter can be used to unload kv strings in the same way as you unload kv files. This can be achieved using pseudo file names e.g.:

```
Build.load_string("""
    <MyRule>:
        Label:
            text="Hello"
""", filename="myrule.kv")
```

can be unloaded via:

```
Build.unload_file("myrule.kv")
```

match(*widget*)

Return a list of ParserRule objects matching the widget.

match_rule_name(*rule_name*)

Return a list of ParserRule objects matching the widget.

sync()

Execute all the waiting operations, such as the execution of all the expressions related to the canvas.

New in version 1.7.0.

template(**args*, ***ctx*)

Create a specialized template using a specific context.

New in version 1.0.5.

With templates, you can construct custom widgets from a kv lang definition by giving them a context. Check [Template usage](#).

unbind_property(*widget, name*)

Unbind the handlers created by all the rules of the widget that set the name.

This effectively clears all the rules of widget that take the form:

```
name: rule
```

For example:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_property(w, 'height')
>>> w.width = 222
>>> w.size
[222, 500]
>>> w.y = 500
>>> w.pos
[550, 500]
```

New in version 1.9.1.

unbind_widget(*uid*)

Unbind all the handlers created by the KV rules of the widget. The `kivy.uix.widget.Widget.uid` is passed here instead of the widget itself, because Builder is using it in the widget destructor.

This effectively clears all the KV rules associated with this widget. For example:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_widget(w.uid)
>>> w.width = 222
>>> w.y = 500
>>> w.size
[222, 500]
>>> w.pos
[50, 500]
```

New in version 1.7.2.

unload_file(*filename*)

Unload all rules associated with a previously imported file.

New in version 1.0.8.

Warning: This will not remove rules or templates already applied/used on current widgets. It will only effect the next widgets creation or template invocation.

exception `kivy.lang.builder.BuilderException`(*context, line, message, cause=None*)

Bases: `kivy.lang.parser.ParserException`

Exception raised when the Builder failed to apply a rule on a widget.

27.13 Parser

Class used for the parsing of .kv files into rules.

class `kivy.lang.parser.Parser`(***kwargs*)

Bases: `builtins.object`

Create a Parser object to parse a Kivy language file or Kivy content.

parse(*content*)

Parse the contents of a Parser file and return a list of root objects.

parse_level(*level, lines, spaces=0*)

Parse the current level (level * spaces) indentation.

strip_comments(*lines*)

Remove all comments from all lines in-place. Comments need to be on a single line and not at the end of a line. i.e. a comment line's first non-whitespace character must be a #.

exception `kivy.lang.parser.ParserException`(*context, line, message, cause=None*)

Bases: `Exception`

Exception raised when something wrong happened in a kv file.

External libraries

Kivy comes with other python/C libraries:

- *ddsfile* - used for parsing and saving DDS files.
- *osc* - a modified/optimized version of PyOSC for using the Open Sound Control protocol.
- *mtdev* - provides support for the Kernel multi-touch transformation library.

Warning: Even though Kivy comes with these external libraries, we do not provide any support for them and they might change in the future. Don't rely on them in your code.

28.1 GstPlayer

New in version 1.8.0.

GstPlayer is a media player implemented specifically for Kivy with Gstreamer 1.0. It doesn't use Gi at all and is focused on what we want: the ability to read video and stream the image in a callback, or read an audio file. Don't use it directly but use our Core providers instead.

This player is automatically compiled if you have *pkg-config --libs --cflags gstreamer-1.0* working.

Warning: This is an external library and Kivy does not provide any support for it. It might change in the future and we advise you don't rely on it in your code.

28.2 DDS File library

This library can be used to parse and save DDS (*DirectDraw Surface* <https://en.wikipedia.org/wiki/DirectDraw_Surface>) files.

The initial version was written by:

Alexey Borzenkov (snaury@gmail.com)

All the initial work credits go to him! Thank you :)

This version uses structs instead of ctypes.

28.2.1 DDS Format

```
[DDS ][SurfaceDesc][Data]

[SurfaceDesc]:: (everything is uint32)
    Size
    Flags
    Height
    Width
    PitchOrLinearSize
    Depth
    MipmapCount
    Reserved1 * 11
    [PixelFormat]::
        Size
        Flags
        FourCC
        RGBBitCount
        RBitMask
        GBitMask
        BBitMask
        ABitMask
    [Caps]::
        Caps1
        Caps2
        Reserved1 * 2
    Reserverd2
```

Warning: This is an external library and Kivy does not provide any support for it. It might change in the future and we advise you don't rely on it in your code.

exception kivy.lib.ddsfile.DDSException

Bases: Exception

class kivy.lib.ddsfile.QueryDict

Bases: builtins.dict

28.3 Python mtdev

The mtdev module provides Python bindings to the [Kernel multi-touch transformation library](#), also known as mtdev (MIT license).

The mtdev library transforms all variants of kernel MT events to the slotted type B protocol. The events put into mtdev may be from any MT device, specifically type A without contact tracking, type A with contact tracking, or type B with contact tracking. See the kernel documentation for further details.

Warning: This is an external library and Kivy does not provide any support for it. It might change in the future and we advise you don't rely on it in your code.

```
class kivy.lib.mtdev.input_absinfo
```

```
    Bases: _ctypes.Structure
```

```
class kivy.lib.mtdev.input_event
```

```
    Bases: _ctypes.Structure
```

```
class kivy.lib.mtdev.mtdev
```

```
    Bases: _ctypes.Structure
```

```
class kivy.lib.mtdev.mtdev_caps
```

```
    Bases: _ctypes.Structure
```

```
class kivy.lib.mtdev.timeval
```

```
    Bases: _ctypes.Structure
```

Modules

Modules are classes that can be loaded when a Kivy application is starting. The loading of modules is managed by the config file. Currently, we include:

- *touchring*: Draw a circle around each touch.
- *monitor*: Add a red topbar that indicates the FPS and a small graph indicating input activity.
- *keybinding*: Bind some keys to actions, such as a screenshot.
- *recorder*: Record and playback a sequence of events.
- *screen*: Emulate the characteristics (dpi/density/ resolution) of different screens.
- *inspector*: Examines your widget hierarchy and widget properties.
- *webdebugger*: Realtime examination of your app internals via a web browser.
- *joycursor*: Navigate in your app with a joystick.
- *showborder*: Show widget's border.

Modules are automatically loaded from the Kivy path and User path:

- *PATH_TO_KIVY/kivy/modules*
- *HOME/.kivy/mods*

29.1 Activating a module

There are various ways in which you can activate a kivy module.

29.1.1 Activate a module in the config

To activate a module this way, you can edit your configuration file (in your *HOME/.kivy/config.ini*):

```
[modules]
# uncomment to activate
touchring =
# monitor =
# keybinding =
```

Only the name of the module followed by “=” is sufficient to activate the module.

29.1.2 Activate a module in Python

Before starting your application, preferably at the start of your import, you can do something like this:

```
import kivy
kivy.require('1.0.8')

# Activate the touchring module
from kivy.config import Config
Config.set('modules', 'touchring', '')
```

29.1.3 Activate a module via the commandline

When starting your application from the commandline, you can add a `-m <modulename>` to the arguments. For example:

```
python main.py -m webdebugger
```

Note: Some modules, such as the screen, may require additional parameters. They will, however, print these parameters to the console when launched without them.

29.2 Create your own module

Create a file in your `HOME/.kivy/mods`, and create 2 functions:

```
def start(win, ctx):
    pass

def stop(win, ctx):
    pass
```

Start/stop are functions that will be called for every window opened in Kivy. When you are starting a module, you can use these to store and manage the module state. Use the `ctx` variable as a dictionary. This context is unique for each instance/start() call of the module, and will be passed to stop() too.

29.3 Console

New in version 1.9.1.

Reboot of the old inspector, designed to be modular and keep concerns separated. It also has an addons architecture that allow you to add a button, panel, or more in the Console itself.

Warning: This module works, but might fail in some cases. Please contribute!

29.3.1 Usage

For normal module usage, please see the *modules* documentation:

```
python main.py -m console
```

29.3.2 Mouse navigation

When the “Select” button is activated, you can:

- tap once on a widget to select it without leaving inspect mode
- double tap on a widget to select and leave inspect mode (then you can manipulate the widget again)

29.3.3 Keyboard navigation

- “Ctrl + e”: toggle console
- “Escape”: cancel widget lookup, then hide inspector view
- “Up”: select the parent widget
- “Down”: select the first child of the currently selected widget
- “Left”: select the previous sibling
- “Right”: select the next sibling

29.3.4 Additional information

Some properties can be edited live. However, due to the delayed usage of some properties, it might crash if you don’t handle the required cases.

29.3.5 Addons

Addons must be added to *Console.addons* before the first Clock tick of the application, or before *create_console* is called. You currently cannot add addons on the fly. Addons are quite cheap until the Console is activated. Panels are even cheaper as nothing is done until the user selects them.

We provide multiple addons activated by default:

- *ConsoleAddonFps*: display the FPS at the top-right
- *ConsoleAddonSelect*: activate the selection mode
- *ConsoleAddonBreadcrumb*: display the hierarchy of the current widget at the bottom
- *ConsoleAddonWidgetTree*: panel to display the widget tree of the application
- *ConsoleAddonWidgetPanel*: panel to display the properties of the selected widget

If you need to add custom widgets in the Console, please use either *ConsoleButton*, *ConsoleToggleButton* or *ConsoleLabel*.

An addon must inherit from the *ConsoleAddon* class.

For example, here is a simple addon for displaying the FPS at the top/right of the Console:

```
from kivy.modules.console import Console, ConsoleAddon

class ConsoleAddonFps(ConsoleAddon):
    def init(self):
        self.lbl = ConsoleLabel(text="0 Fps")
        self.console.add_toolbar_widget(self.lbl, right=True)

    def activate(self):
        self.event = Clock.schedule_interval(self.update_fps, 1 / 2.)

    def deactivated(self):
        self.event.cancel()

    def update_fps(self, *args):
        fps = Clock.get_fps()
        self.lbl.text = "{} Fps".format(int(fps))

Console.register_addon(ConsoleAddonFps)
```

You can create addons that add panels. Panel activation/deactivation is not tied to the addon activation/deactivation, but in some cases, you can use the same callback for deactivating the addon and the panel. Here is a simple “About” panel addon:

```
from kivy.modules.console import Console, ConsoleAddon, ConsoleLabel

class ConsoleAddonAbout(ConsoleAddon):
    def init(self):
        self.console.add_panel("About", self.panel_activate,
                               self.panel_deactivate)

    def panel_activate(self):
        self.console.bind(widget=self.update_content)
        self.update_content()

    def panel_deactivate(self):
        self.console.unbind(widget=self.update_content)

    def deactivate(self):
        self.panel_deactivate()

    def update_content(self, *args):
        widget = self.console.widget
        if not widget:
            return
        text = "Selected widget is: {!r}".format(widget)
        lbl = ConsoleLabel(text=text)
        self.console.set_content(lbl)

Console.register_addon(ConsoleAddonAbout)
```

`kivy.modules.console.start(win, ctx)`

Create an Console instance attached to the *ctx* and bound to the Window’s *on_keyboard()* event for capturing the keyboard shortcut.

Parameters

win: A **Window** The application Window to bind to.

ctx: A **Widget** or subclass The Widget to be inspected.

`kivy.modules.console.stop(win, ctx)`

Stop and unload any active Inspectors for the given *ctx*.

class `kivy.modules.console.Console(**kwargs)`

Bases: `kivy.uix.relativelayout.RelativeLayout`

Console interface

This widget is created by `create_console()`, when the module is loaded. During that time, you can add addons on the console to extend the functionalities, or add your own application stats / debugging module.

activated

True if the Console is activated (showed)

add_panel(*name, cb_activate, cb_deactivate, cb_refresh=None*)

Add a new panel in the Console.

- *cb_activate* is a callable that will be called when the panel is activated by the user.
- *cb_deactivate* is a callable that will be called when the panel is deactivated or when the console will hide.
- *cb_refresh* is an optional callable that is called if the user click again on the button for display the panel

When activated, it's up to the panel to display a content in the Console by using `set_content()`.

add_toolbar_widget(*widget, right=False*)

Add a widget in the top left toolbar of the Console. Use *right=True* if you wanna add the widget at the right instead.

addons = [`<class 'kivy.modules.console.ConsoleAddonSelect'>`, `<class 'kivy.modules.c`

Array of addons that will be created at Console creation

highlight_at(*x, y*)

Select a widget from a x/y window coordinate. This is mostly used internally when Select mode is activated

inspect_enabled

Indicate if the inspector inspection is enabled. If yes, the next touch down will select a the widget under the touch

mode

Display mode of the Console, either docked at the bottom, or as a floating window.

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.

pick(*widget*, *x*, *y*)

Pick a widget at *x*/*y*, given a root *widget*

remove_toolbar_widget(*widget*)

Remove a widget from the toolbar

set_content(*content*)

Replace the Console content with a new one.

widget

Current widget being selected

class `kivy.modules.console.ConsoleAddon`(*console*)

Bases: `builtins.object`

Base class for implementing addons

activate()

Method called when the addon is activated by the console (when the console is displayed)

console = `None`

Console instance

deactivate()

Method called when the addon is deactivated by the console (when the console is hidden)

init()

Method called when the addon is instantiated by the Console

class `kivy.modules.console.ConsoleButton`(***kwargs*)

Bases: `kivy.uix.button.Button`

Button specialized for the Console

class `kivy.modules.console.ConsoleToggleButton`(***kwargs*)

Bases: `kivy.uix.togglebutton.ToggleButton`

ToggleButton specialized for the Console

class `kivy.modules.console.ConsoleLabel`(***kwargs*)

Bases: `kivy.uix.label.Label`

LabelButton specialized for the Console

29.4 Inspector

New in version 1.0.9.

Warning: This module is highly experimental, use it with care.

The Inspector is a tool for finding a widget in the widget tree by clicking or tapping on it. Some keyboard shortcuts are activated:

- “Ctrl + e”: activate / deactivate the inspector view
- “Escape”: cancel widget lookup first, then hide the inspector view

Available inspector interactions:

- tap once on a widget to select it without leaving inspect mode
- double tap on a widget to select and leave inspect mode (then you can manipulate the widget again)

Some properties can be edited live. However, due to the delayed usage of some properties, it might crash if you don't handle all the cases.

29.4.1 Usage

For normal module usage, please see the [modules](#) documentation.

The Inspector, however, can also be imported and used just like a normal python module. This has the added advantage of being able to activate and deactivate the module programmatically:

```
from kivy.core.window import Window
from kivy.app import App
from kivy.uix.button import Button
from kivy.modules import inspector

class Demo(App):
    def build(self):
        button = Button(text="Test")
        inspector.create_inspector(Window, button)
        return button

Demo().run()
```

To remove the Inspector, you can do the following:

```
inspector.stop(Window, button)
```

`kivy.modules.inspector.stop(win, ctx)`

Stop and unload any active Inspectors for the given *ctx*.

`kivy.modules.inspector.create_inspector(win, ctx, *l)`

Create an Inspector instance attached to the *ctx* and bound to the Window's [on_keyboard\(\)](#) event for capturing the keyboard shortcut.

Parameters

win: A [Window](#) The application Window to bind to.

ctx: A [Widget](#) or subclass The Widget to be inspected.

29.5 JoyCursor

New in version 1.10.0.

The JoyCursor is a tool for navigating with a joystick as if using a mouse or touch. Most of the actions that are possible for a mouse user are available in this module.

For example:

- left click
- right click
- double click (two clicks)

- moving the cursor
- holding the button (+ moving at the same time)
- selecting
- scrolling

There are some properties that can be edited live, such as intensity of the JoyCursor movement and toggling mouse button holding.

29.5.1 Usage

For normal module usage, please see the *modules* documentation and these bindings:

Event	Joystick
cursor move	Axis 3, Axis 4
cursor intensity	Button 0, Button 1
left click	Button 2
right click	Button 3
scroll up	Button 4
scroll down	Button 5
hold button	Button 6
joycursor on/off	Button 7

The JoyCursor, like Inspector, can also be imported and used as a normal python module. This has the added advantage of being able to activate and deactivate the module programmatically:

```
from kivy.lang import Builder
from kivy.base import runTouchApp
runTouchApp(Builder.load_string("""
#:import jc kivy.modules.joycursor
BoxLayout:
    Button:
        text: 'Press & activate with Ctrl+E or Button 7'
        on_release: jc.create_joycursor(root.parent, root)
    Button:
        text: 'Disable'
        on_release: jc.stop(root.parent, root)
"""))
```

`kivy.modules.joycursor.stop(win, ctx)`

Stop and unload any active JoyCursors for the given *ctx*.

`kivy.modules.joycursor.create_joycursor(win, ctx, *args)`

Create a JoyCursor instance attached to the *ctx* and bound to the Window's *on_keyboard()* event for capturing the keyboard shortcuts.

Parameters

win: A **Window** The application Window to bind to.

ctx: A **Widget** or subclass The Widget for JoyCursor to attach to.

29.6 Keybinding

This module forces the mapping of some keys to functions:

- F11: Rotate the Window through 0, 90, 180 and 270 degrees
- Shift + F11: Switches between portrait and landscape on desktops
- F12: Take a screenshot

Note: this doesn't work if the application requests the keyboard beforehand.

29.6.1 Usage

For normal module usage, please see the *modules* documentation.

The Keybinding module, however, can also be imported and used just like a normal python module. This has the added advantage of being able to activate and deactivate the module programmatically:

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.modules import keybinding
from kivy.core.window import Window

class Demo(App):
    def build(self):
        button = Button(text="Hello")
        keybinding.start(Window, button)
        return button

Demo().run()
```

To remove the Keybinding, you can do the following:

```
Keybinding.stop(Window, button)
```

29.7 Monitor module

The Monitor module is a toolbar that shows the activity of your current application :

- FPS
- Graph of input events

29.7.1 Usage

For normal module usage, please see the *modules* documentation.

29.8 Recorder module

New in version 1.1.0.

Create an instance of *Recorder*, attach to the class, and bind some keys to record / play sequences:

- F6: play the last record in a loop
- F7: read the latest recording
- F8: record input events

29.8.1 Configuration

Parameters

attrs: str, defaults to **record_attrs** value. Attributes to record from the motion event

profile_mask: str, defaults to **record_profile_mask** value. Mask for motion event profile. Used to filter which profile will appear in the fake motion event when replayed.

filename: str, defaults to 'recorder.kvi' Name of the file to record / play with

29.8.2 Usage

For normal module usage, please see the **modules** documentation.

29.9 Screen

This module changes some environment and configuration variables to match the density / dpi / screensize of a specific device.

To see a list of the available screenid's, just run:

```
python main.py -m screen
```

To simulate a medium-density screen such as the Motorola Droid 2:

```
python main.py -m screen:droid2
```

To simulate a high-density screen such as HTC One X, in portrait:

```
python main.py -m screen:onex,portrait
```

To simulate the iPad 2 screen:

```
python main.py -m screen:ipad
```

If the generated window is too large, you can specify a scale:

```
python main.py -m screen:note2,portrait,scale=.75
```

Note that to display your contents correctly on a scaled window you must consistently use units 'dp' and 'sp' throughout your app. See **metrics** for more details.

29.10 Touchring

Shows rings around every touch on the surface / screen. You can use this module to check that you don't have any calibration issues with touches.

29.10.1 Configuration

Parameters

image: str, defaults to '<kivy>/data/images/ring.png' Filename of the image to use.

scale: float, defaults to 1. Scale of the image.

alpha: float, defaults to 1. Opacity of the image.

29.10.2 Example

In your configuration (`~/.kivy/config.ini`), you can add something like this:

```
[modules]
touchring = image=mypointer.png,scale=.3,alpha=.7
```

29.11 Web Debugger

New in version 1.2.0.

Warning: This module is highly experimental, use it with care.

This module will start a webserver and run in the background. You can see how your application evolves during runtime, examine the internal cache etc.

Run with:

```
python main.py -m webdebugger
```

Then open your webbrowser on <http://localhost:5000/>

Network support

Kivy currently supports basic, asynchronous network requests. Please refer to [kivy.network.urlrequest.UrlRequest](#).

30.1 UrlRequest

New in version 1.0.8.

You can use the [UrlRequest](#) to make asynchronous requests on the web and get the result when the request is completed. The spirit is the same as the XMLHttpRequest object in Javascript.

The content is also decoded if the Content-Type is application/json and the result automatically passed through `json.loads`.

The syntax to create a request:

```
from kivy.network.urlrequest import UrlRequest
req = UrlRequest(url, on_success, on_redirect, on_failure, on_error,
                 on_progress, req_body, req_headers, chunk_size,
                 timeout, method, decode, debug, file_path, ca_file,
                 verify)
```

Only the first argument is mandatory: the rest are optional. By default, a “GET” request will be sent. If the [UrlRequest.req_body](#) is not None, a “POST” request will be sent. It’s up to you to adjust [UrlRequest.req_headers](#) to suit your requirements and the response to the request will be accessible as the parameter called “result” on the callback function of the on_success event.

Example of fetching JSON:

```
def got_json(req, result):
    for key, value in result['headers'].items():
        print('{}: {}'.format(key, value))

req = UrlRequest('https://httpbin.org/headers', got_json)
```

Example of Posting data (adapted from urllib example):

```
import urllib

def bug_posted(req, result):
    print('Our bug is posted!')
    print(result)

params = urllib.urlencode({'@number': 12524, '@type': 'issue',
                           '@action': 'show'})
headers = {'Content-type': 'application/x-www-form-urlencoded',
           'Accept': 'text/plain'}
req = URLRequest('bugs.python.org', on_success=bug_posted, req_body=params,
                 req_headers=headers)
```

If you want a synchronous request, you can call the `wait()` method.

```
class kivy.network.urlrequest.UrlRequest(url, on_success=None, on_redirect=None,
                                         on_failure=None, on_error=None,
                                         on_progress=None, req_body=None,
                                         req_headers=None, chunk_size=8192,
                                         timeout=None, method=None, decode=True,
                                         debug=False, file_path=None, ca_file=None,
                                         verify=True, proxy_host=None,
                                         proxy_port=None, proxy_headers=None,
                                         user_agent=None, on_cancel=None)
```

Bases: `threading.Thread`

A `UrlRequest`. See module documentation for usage.

Changed in version 1.5.1: Add *debug* parameter

Changed in version 1.0.10: Add *method* parameter

Changed in version 1.8.0: Parameter *decode* added. Parameter *file_path* added. Parameter *on_redirect* added. Parameter *on_failure* added.

Changed in version 1.9.1: Parameter *ca_file* added. Parameter *verify* added.

Changed in version 1.10.0: Parameters *proxy_host*, *proxy_port* and *proxy_headers* added.

Changed in version 1.11.0: Parameters *on_cancel* added.

Parameters

url: `str` Complete url string to call.

on_success: `callback(request, result)` Callback function to call when the result has been fetched.

on_redirect: `callback(request, result)` Callback function to call if the server returns a Redirect.

on_failure: `callback(request, result)` Callback function to call if the server returns a Client or Server Error.

on_error: `callback(request, error)` Callback function to call if an error occurs.

on_progress: `callback(request, current_size, total_size)` Callback function that will be called to report progression of the download. *total_size* might be -1 if no Content-Length has been reported in the http response. This callback will be called after each *chunk_size* is read.

on_cancel: `callback(request)` Callback function to call if user requested to cancel the download operation via the `.cancel()` method.

req_body: str, defaults to None Data to sent in the request. If it's not None, a POST will be done instead of a GET.

req_headers: dict, defaults to None Custom headers to add to the request.

chunk_size: int, defaults to 8192 Size of each chunk to read, used only when *on_progress* callback has been set. If you decrease it too much, a lot of *on_progress* callbacks will be fired and will slow down your download. If you want to have the maximum download speed, increase the *chunk_size* or don't use *on_progress*.

timeout: int, defaults to None If set, blocking operations will timeout after this many seconds.

method: str, defaults to 'GET' (or 'POST' if *body* is specified) The HTTP method to use.

decode: bool, defaults to True If False, skip decoding of the response.

debug: bool, defaults to False If True, it will use the `Logger.debug` to print information about url access/progression/errors.

file_path: str, defaults to None If set, the result of the `URLRequest` will be written to this path instead of in memory.

ca_file: str, defaults to None Indicates a SSL CA certificate file path to validate HTTPS certificates against

verify: bool, defaults to True If False, disables SSL CA certificate verification

proxy_host: str, defaults to None If set, the proxy host to use for this connection.

proxy_port: int, defaults to None If set, and *proxy_host* is also set, the port to use for connecting to the proxy server.

proxy_headers: dict, defaults to None If set, and *proxy_host* is also set, the headers to send to the proxy server in the `CONNECT` request.

cancel()

Cancel the current request. It will be aborted, and the result will not be dispatched. Once cancelled, the callback *on_cancel* will be called.

New in version 1.11.0.

chunk_size

Return the size of a chunk, used only in "progress" mode (when *on_progress* callback is set.)

decode_result(result, resp)

Decode the result fetched from url according to his Content-Type. Currently supports only application/json.

error

Return the error of the request. This value is not determined until the request is completed.

get_connection_for_scheme(scheme)

Return the Connection class for a particular scheme. This is an internal function that can be expanded to support custom schemes.

Actual supported schemes: http, https.

is_finished

Return True if the request has finished, whether it's a success or a failure.

req_body = None

Request body passed in `__init__`

req_headers = None

Request headers passed in `__init__`

resp_headers

If the request has been completed, return a dictionary containing the headers of the response. Otherwise, it will return `None`.

resp_status

Return the status code of the response if the request is complete, otherwise return `None`.

result

Return the result of the request. This value is not determined until the request is finished.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

url = None

Url of the request

wait(delay=0.5)

Wait for the request to finish (until *resp_status* is not `None`)

Note: This method is intended to be used in the main thread, and the callback will be dispatched from the same thread from which you're calling.

New in version 1.1.0.

Storage

New in version 1.7.0.

Warning: This module is still experimental, and the API is subject to change in a future version.

31.1 Usage

The idea behind the Storage module is to be able to load/store any number of key-value pairs via an indexed key. The default model is abstract so you cannot use it directly. We provide some implementations such as:

- *kivy.storage.dictstore.DictStore*: use a python dict as a store
- *kivy.storage.jsonstore.JsonStore*: use a JSON file as a store
- *kivy.storage.redisstore.RedisStore*: use a Redis database with *redis-py*

31.2 Examples

For example, let's use a *JsonStore*:

```
from kivy.storage.jsonstore import JsonStore

store = JsonStore('hello.json')

# put some values
store.put('tito', name='Mathieu', org='kivy')
store.put('tshirtman', name='Gabriel', age=27)

# using the same index key erases all previously added key-value pairs
```

(continues on next page)

(continued from previous page)

```
store.put('tito', name='Mathieu', age=30)

# get a value using a index key and key
print('tito is', store.get('tito')['age'])

# or guess the key/entry for a part of the key
for item in store.find(name='Gabriel'):
    print('tshirtmans index key is', item[0])
    print('his key value pairs are', str(item[1]))
```

Because the data is persistent, you can check later to see if the key exists:

```
from kivy.storage.jsonstore import JsonStore

store = JsonStore('hello.json')
if store.exists('tito'):
    print('tite exists:', store.get('tito'))
    store.delete('tito')
```

31.3 Synchronous / Asynchronous API

All the standard methods (*get()*, *put()*, *exists()*, *delete()*, *find()*) have an asynchronous version.

For example, the *get* method has a *callback* parameter. If set, the *callback* will be used to return the result to the user when available: the request will be asynchronous. If the *callback* is *None*, then the request will be synchronous and the result will be returned directly.

Without callback (Synchronous API):

```
entry = mystore.get('tito')
print('tito =', entry)
```

With callback (Asynchronous API):

```
def my_callback(store, key, result):
    print('the key', key, 'has a value of', result)
mystore.get('plop', callback=my_callback)
```

The callback signature (for almost all methods) is:

```
def callback(store, key, result):
    """
    store: the `Store` instance currently used.
    key: the key sought for.
    result: the result of the lookup for the key.
    """
```

31.4 Synchronous container type

The storage API emulates the container type for the synchronous API:

```

store = JsonStore('hello.json')

# original: store.get('tito')
store['tito']

# original: store.put('tito', name='Mathieu')
store['tito'] = {'name': 'Mathieu'}

# original: store.delete('tito')
del store['tito']

# original: store.count()
len(store)

# original: store.exists('tito')
'tito' in store

# original: for key in store.keys()
for key in store:
    pass

```

class kivy.storage.**AbstractStore**(**kwargs)

Bases: **kivy.event.EventDispatcher**

Abstract class used to implement a Store

async_clear(callback)

Asynchronous version of **clear()**.

async_count(callback)

Asynchronously return the number of entries in the storage.

async_delete(callback, key)

Asynchronous version of **delete()**.

Callback arguments

store: **AbstractStore** instance Store instance

key: **string** Name of the key to search for

result: **bool** Indicate True if the storage has been updated, or False if nothing has been done (no changes). None if any error.

async_exists(callback, key)

Asynchronous version of **exists()**.

Callback arguments

store: **AbstractStore** instance Store instance

key: **string** Name of the key to search for

result: **boo** Result of the query, None if any error

async_find(callback, **filters)

Asynchronous version of **find()**.

The callback will be called for each entry in the result.

Callback arguments

store: **AbstractStore** instance Store instance

key: **string** Name of the key to search for, or None if we reach the end of the results

result: bool Indicate True if the storage has been updated, or False if nothing has been done (no changes). None if any error.

async_get(*callback, key*)

Asynchronous version of *get()*.

Callback arguments

store: *AbstractStore* instance Store instance

key: *string* Name of the key to search for

result: *dict* Result of the query, None if any error

async_keys(*callback*)

Asynchronously return all the keys in the storage.

async_put(*callback, key, **values*)

Asynchronous version of *put()*.

Callback arguments

store: *AbstractStore* instance Store instance

key: *string* Name of the key to search for

result: *bool* Indicate True if the storage has been updated, or False if nothing has been done (no changes). None if any error.

clear()

Wipe the whole storage.

count()

Return the number of entries in the storage.

delete(*key*)

Delete a key from the storage. If the key is not found, a *KeyError* exception will be thrown.

exists(*key*)

Check if a key exists in the store.

find(***filters*)

Return all the entries matching the filters. The entries are returned through a generator as a list of (key, entry) pairs where *entry* is a dict of key-value pairs

```
for key, entry in store.find(name='Mathieu'):
    print('key:', key, ', entry:', entry)
```

Because it's a generator, you cannot directly use it as a list. You can do:

```
# get all the (key, entry) disponibles
entries = list(store.find(name='Mathieu'))
# get only the entry from (key, entry)
entries = list((x[1] for x in store.find(name='Mathieu')))
```

get(*key*)

Get the key-value pairs stored at *key*. If the key is not found, a *KeyError* exception will be thrown.

keys()

Return a list of all the keys in the storage.

put(*key, **values*)

Put new key-value pairs (given in *values*) into the storage. Any existing key-value pairs will be removed.

31.5 Dictionary store

Use a Python dictionary as a store.

class `kivy.storage.dictstore.DictStore`(*filename*, *data=None*, ***kwargs*)

Bases: `kivy.storage.AbstractStore`

Store implementation using a pickled *dict*. See the `kivy.storage` module documentation for more information.

31.6 JSON store

A `Storage` module used to save/load key-value pairs from a json file.

class `kivy.storage.jsonstore.JsonStore`(*filename*, *indent=None*, *sort_keys=False*, ***kwargs*)

Bases: `kivy.storage.AbstractStore`

Store implementation using a json file for storing the key-value pairs. See the `kivy.storage` module documentation for more information.

31.7 Redis Store

Store implementation using Redis. You must have redis-py installed.

Usage example:

```
from kivy.storage.redisstore import RedisStore

params = dict(host='localhost', port=6379, db=14)
store = RedisStore(params)
```

All the key-value pairs will be stored with a prefix 'store' by default. You can instantiate the storage with another prefix like this:

```
from kivy.storage.redisstore import RedisStore

params = dict(host='localhost', port=6379, db=14)
store = RedisStore(params, prefix='mystore2')
```

The params dictionary will be passed to the `redis.StrictRedis` class.

See `redis-py`.

class `kivy.storage.redisstore.RedisStore`(*redis_params*, ***kwargs*)

Bases: `kivy.storage.AbstractStore`

Store implementation using a Redis database. See the `kivy.storage` module documentation for more informations.

Tools

The tools module provides various utility scripts, modules and examples.

32.1 Scripts

Some useful scripts include:

- `viewer.py`: for viewing kv files with automatic updating
- `benchmark.py`: provides detailed OpenGL hardware information as well as some benchmarks measuring kivy specific performance
- `reports.py`: provides a comprehensive report covering your systems providers, libraries, configuration, environment, input devices and options
- `texturecompress.py`: a command line utility for compressing images into PVRTC or ETC1 formats
- `generate-icons.py`: generates set of icons suitable for the various store and package formats
- `gles_compat/subset_gles.py`: examines compatibility between GLEXT and GLES2 headers for finding compatible subsets

32.2 Modules

Tool modules provide various resources for:

- *packaging*
- text editor highlighting

32.3 Other

Other miscellaneous resources include

- `pep8checker`: pep8 checking scripts and git hook
- `theming`: demonstrates an alternative theme for kivy
- `travis`: travis continuous integration

This help document is a work-in-progress and currently under construction.

32.4 Packaging

This module contains [PyInstaller](#) hooks in order to assist in the process of building binary packages. PyInstaller allows you to produce stand-alone, self-contained executables of your Kivy app for Windows, Linux and Mac.

For more information, please see the [PyInstaller website](#)

Widgets

Widgets are elements of a graphical user interface that form part of the [User Experience](#). The *kivy.uix* module contains classes for creating and managing Widgets. Please refer to the *Widget class* documentation for further information.

Kivy widgets can be categorized as follows:

- **UX widgets:** Classical user interface widgets, ready to be assembled to create more complex widgets.
Label, Button, CheckBox, Image, Slider, Progress Bar, Text Input, Toggle button, Switch, Video
 - **Layouts:** A layout widget does no rendering but just acts as a trigger that arranges its children in a specific way. Read more on *Layouts here*.
Anchor Layout, Box Layout, Float Layout, Grid Layout, PageLayout, Relative Layout, Scatter Layout, Stack Layout
 - **Complex UX widgets:** Non-atomic widgets that are the result of combining multiple classic widgets. We call them complex because their assembly and usage are not as generic as the classical widgets.
Bubble, Drop-Down List, FileChooser, Popup, Spinner, RecycleView, TabbedPanel, Video player, VKeyboard,
 - **Behaviors widgets:** These widgets do no rendering but act on the graphics instructions or interaction (touch) behavior of their children.
Scatter, Stencil View
 - **Screen manager:** Manages screens and transitions when switching from one to another.
Screen Manager
-

33.1 Behaviors

New in version 1.8.0.

33.1.1 Behavior mixin classes

This module implements behaviors that can be **mixed in** with existing base widgets. The idea behind these classes is to encapsulate properties and events associated with certain types of widgets.

Isolating these properties and events in a mixin class allows you to define your own implementation for standard kivy widgets that can act as drop-in replacements. This means you can re-style and re-define widgets as desired without breaking compatibility: as long as they implement the behaviors correctly, they can simply replace the standard widgets.

33.1.2 Adding behaviors

Say you want to add **Button** capabilities to an **Image**, you could do:

```
class IconButton(ButtonBehavior, Image):  
    pass
```

This would give you an **Image** with the events and properties inherited from **ButtonBehavior**. For example, the *on_press* and *on_release* events would be fired when appropriate:

```
class IconButton(ButtonBehavior, Image):  
    def on_press(self):  
        print("on_press")
```

Or in kv:

```
IconButton:  
    on_press: print('on_press')
```

Naturally, you could also bind to any property changes the behavior class offers:

```
def state_changed(*args):  
    print('state changed')
```



```
button = IconButton()  
button.bind(state=state_changed)
```

Note: The behavior class must always be *_before_* the widget class. If you don't specify the inheritance in this order, the behavior will not work because the behavior methods are overwritten by the class method listed first.

Similarly, if you combine a behavior class with a class which requires the use of the methods also defined by the behavior class, the resulting class may not function properly. For example, when combining the **ButtonBehavior** with a **Slider**, both of which use the *on_touch_up()* method, the resulting class may not work properly.

Changed in version 1.9.1: The individual behavior classes, previously in one big *behaviors.py* file, has been split into a single file for each class under the **behaviors** module. All the behaviors are still imported in the **behaviors** module so they are accessible as before (e.g. both *from kivy.uix.behaviors import ButtonBehavior* and *from kivy.uix.behaviors.button import ButtonBehavior* work).

```
class kivy.uix.behaviors.ButtonBehavior(**kwargs)  
    Bases: builtins.object
```

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

Events

on_press Fired when the button is pressed.

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

always_release

This determines whether or not the widget fires an *on_release* event if the touch_up is outside the widget.

New in version 1.9.0.

Changed in version 1.10.0: The default value is now False.

always_release is a *BooleanProperty* and defaults to *False*.

last_touch

Contains the last relevant touch received by the Button. This can be used in *on_press* or *on_release* in order to know which touch dispatched the event.

New in version 1.8.0.

last_touch is a *ObjectProperty* and defaults to *None*.

min_state_time

The minimum period of time which the widget must remain in the 'down' state.

New in version 1.9.1.

min_state_time is a float and defaults to 0.035. This value is taken from *Config*.

state

The state of the button, must be one of 'normal' or 'down'. The state is 'down' only when the button is currently touched/clicked, otherwise its 'normal'.

state is an *OptionProperty* and defaults to 'normal'.

trigger_action(*duration=0.1*)

Trigger whatever action(s) have been bound to the button by calling both the *on_press* and *on_release* callbacks.

This simulates a quick button press without using any touch events.

Duration is the length of the press in seconds. Pass 0 if you want the action to happen instantly.

New in version 1.8.0.

class `kivy.uix.behaviors.ToggleButtonBehavior`(***kwargs*)

Bases: *kivy.uix.behaviors.button.ButtonBehavior*

This *mixin* class provides *togglebutton* behavior. Please see the *togglebutton behaviors module* documentation for more information.

New in version 1.8.0.

allow_no_selection

This specifies whether the widgets in a group allow no selection i.e. everything to be deselected.

New in version 1.9.0.

allow_no_selection is a *BooleanProperty* and defaults to *True*

static `get_widgets`(*groupname*)

Return a list of the widgets contained in a specific group. If the group doesn't exist, an empty list will be returned.

Note: Always release the result of this method! Holding a reference to any of these widgets can prevent them from being garbage collected. If in doubt, do:

```
l = ToggleButtonBehavior.get_widgets('mygroup')
# do your job
del l
```

Warning: It's possible that some widgets that you have previously deleted are still in the list. The garbage collector might need to release other objects before flushing them.

group

Group of the button. If *None*, no group will be used (the button will be independent). If specified, *group* must be a hashable object, like a string. Only one button in a group can be in a 'down' state.

group is a *ObjectProperty* and defaults to *None*.

class `kivy.uix.behaviors.DragBehavior(**kwargs)`

Bases: `builtins.object`

The `DragBehavior` *mixin* provides Drag behavior. When combined with a widget, dragging in the rectangle defined by *drag_rectangle* will drag the widget. Please see the *drag behaviors module* documentation for more information.

New in version 1.8.0.

drag_distance

Distance to move before dragging the *DragBehavior*, in pixels. As soon as the distance has been traveled, the *DragBehavior* will start to drag, and no touch event will be dispatched to the children. It is advisable that you base this value on the dpi of your target device's screen.

drag_distance is a *NumericProperty* and defaults to the *scroll_distance* as defined in the user *Config* (20 pixels by default).

drag_rect_height

Height of the axis aligned bounding rectangle where dragging is allowed.

drag_rect_height is a *NumericProperty* and defaults to 100.

drag_rect_width

Width of the axis aligned bounding rectangle where dragging is allowed.

drag_rect_width is a *NumericProperty* and defaults to 100.

drag_rect_x

X position of the axis aligned bounding rectangle where dragging is allowed (in window coordinates).

drag_rect_x is a *NumericProperty* and defaults to 0.

drag_rect_y

Y position of the axis aligned bounding rectangle where dragging is allowed (in window coordinates).

drag_rect_y is a *NumericProperty* and defaults to 0.

drag_rectangle

Position and size of the axis aligned bounding rectangle where dragging is allowed.

drag_rectangle is a *ReferenceListProperty* of (*drag_rect_x*, *drag_rect_y*, *drag_rect_width*, *drag_rect_height*) properties.

drag_timeout

Timeout allowed to trigger the *drag_distance*, in milliseconds. If the user has not moved *drag_distance* within the timeout, dragging will be disabled, and the touch event will be dispatched to the children.

drag_timeout is a *NumericProperty* and defaults to the *scroll_timeout* as defined in the user *Config* (55 milliseconds by default).

class kivy.uix.behaviors.**FocusBehavior**(**kwargs)

Bases: `builtins.object`

Provides keyboard focus behavior. When combined with other *FocusBehavior* widgets it allows one to cycle focus among them by pressing tab. Please see the *focus behavior module documentation* for more information.

New in version 1.9.0.

focus

Whether the instance currently has focus.

Setting it to True will bind to and/or request the keyboard, and input will be forwarded to the instance. Setting it to False will unbind and/or release the keyboard. For a given keyboard, only one widget can have its focus, so focusing one will automatically unfocus the other instance holding its focus.

When using a software keyboard, please refer to the *softinput_mode* property to determine how the keyboard display is handled.

focus is a *BooleanProperty* and defaults to False.

focus_next

The *FocusBehavior* instance to acquire focus when tab is pressed and this instance has focus, if not *None* or *StopIteration*.

When tab is pressed, focus cycles through all the *FocusBehavior* widgets that are linked through *focus_next* and are focusable. If *focus_next* is *None*, it instead walks the children lists to find the next focusable widget. Finally, if *focus_next* is the *StopIteration* class, focus won't move forward, but end here.

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

focus_previous

The *FocusBehavior* instance to acquire focus when shift+tab is pressed on this instance, if not *None* or *StopIteration*.

When shift+tab is pressed, focus cycles through all the *FocusBehavior* widgets that are linked through *focus_previous* and are focusable. If *focus_previous* is *None*, it instead walks the children tree to find the previous focusable widget. Finally, if *focus_previous* is the *StopIteration* class, focus won't move backward, but end here.

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

focused

An alias of *focus*.

focused is a *BooleanProperty* and defaults to False.

Warning: <i>focused</i> is an alias of <i>focus</i> and will be removed in 2.0.0.
--

get_focus_next()

Returns the next focusable widget using either *focus_next* or the children similar to the order when tabbing forwards with the **tab** key.

get_focus_previous()

Returns the previous focusable widget using either *focus_previous* or the children similar to the order when **tab + shift** key are triggered together.

hide_keyboard()

Convenience function to hide the keyboard in managed mode.

ignored_touch = []

A list of touches that should not be used to defocus. After **on_touch_up**, every touch that is not in *ignored_touch* will defocus all the focused widgets if the **config** keyboard mode is not **multi**. Touches on focusable widgets that were used to focus are automatically added here.

Example usage:

```
class Unfocusable(Widget):  
    def on_touch_down(self, touch):  
        if self.collide_point(*touch.pos):  
            FocusBehavior.ignored_touch.append(touch)
```

Notice that you need to access this as a class, not an instance variable.

input_type

The kind of input keyboard to request.

New in version 1.8.0.

input_type is an **OptionsProperty** and defaults to **'text'**. Can be one of **'text'**, **'number'**, **'url'**, **'mail'**, **'datetime'**, **'tel'** or **'address'**.

is_focusable

Whether the instance can become focused. If focused, it'll lose focus when set to **False**.

is_focusable is a **BooleanProperty** and defaults to **True** on a desktop (i.e. *desktop* is **True** in *config*), **False** otherwise.

keyboard

The keyboard to bind to (or bound to the widget) when focused.

When **None**, a keyboard is requested and released whenever the widget comes into and out of focus. If not **None**, it must be a keyboard, which gets bound and unbound from the widget whenever it's in or out of focus. It is useful only when more than one keyboard is available, so it is recommended to be set to **None** when only one keyboard is available.

If more than one keyboard is available, whenever an instance gets focused a new keyboard will be requested if **None**. Unless the other instances lose focus (e.g. if **tab** was used), a new keyboard will appear. When this is undesired, the **keyboard** property can be used. For example, if there are two users with two keyboards, then each keyboard can be assigned to different groups of instances of **FocusBehavior**, ensuring that within each group, only one **FocusBehavior** will have focus, and will receive input from the correct keyboard. See *keyboard_mode* in *config* for more information on the keyboard modes.

Keyboard and focus behavior

When using the keyboard, there are some important default behaviors you should keep in mind.

- When Config's *keyboard_mode* is multi, each new touch is considered a touch by a different user and will set the focus (if clicked on a focusable) with a new keyboard. Already focused elements will not lose their focus (even if an unfocusable widget is touched).
- If the keyboard property is set, that keyboard will be used when the instance gets focused. If widgets with different keyboards are linked through *focus_next* and *focus_previous*, then as they are tabbed through, different keyboards will become active. Therefore, typically it's undesirable to link instances which are assigned different keyboards.
- When a widget has focus, setting its keyboard to None will remove its keyboard, but the widget will then immediately try to get another keyboard. In order to remove its keyboard, rather set its *focus* to False.
- When using a software keyboard, typical on mobile and touch devices, the keyboard display behavior is determined by the *softinput_mode* property. You can use this property to ensure the focused widget is not covered or obscured.

keyboard is an *AliasProperty* and defaults to None.

keyboard_mode

Determines how the keyboard visibility should be managed. 'auto' will result in the standard behaviour of showing/hiding on focus. 'managed' requires setting the keyboard visibility manually, or calling the helper functions *show_keyboard()* and *hide_keyboard()*.

keyboard_mode is an *OptionsProperty* and defaults to 'auto'. Can be one of 'auto' or 'managed'.

keyboard_on_key_down(*window, keycode, text, modifiers*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input press. The parameters are the same as *kivy.core.window.WindowBase.on_key_down()*.

When overwriting the method in the derived widget, super should be called to enable tab cycling. If the derived widget wishes to use tab for its own purposes, it can call super after it has processed the character (if it does not wish to consume the tab).

Similar to other keyboard functions, it should return True if the key was consumed.

keyboard_on_key_up(*window, keycode*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input release. The parameters are the same as *kivy.core.window.WindowBase.on_key_up()*.

When overwriting the method in the derived widget, super should be called to enable defocusing on escape. If the derived widget wishes to use escape for its own purposes, it can call super after it has processed the character (if it does not wish to consume the escape).

See *keyboard_on_key_down()*

show_keyboard()

Convenience function to show the keyboard in managed mode.

unfocus_on_touch

Whether a instance should lose focus when clicked outside the instance.

When a user clicks on a widget that is focus aware and shares the same keyboard as this widget (which in the case of only one keyboard, are all focus aware widgets), then as the other widgets gains focus, this widget loses focus. In addition to that, if this property is *True*, clicking on any widget other than this widget, will remove focus from this widget.

unfocus_on_touch is a *BooleanProperty* and defaults to *False* if the *keyboard_mode* in *Config* is 'multi' or 'systemandmulti', otherwise it defaults to *True*.

class kivy.uix.behaviors.CompoundSelectionBehavior(**kwargs)

Bases: builtins.object

The Selection behavior *mixin* implements the logic behind keyboard and touch selection of selectable widgets managed by the derived widget. Please see the *compound selection behaviors module* documentation for more information.

New in version 1.9.0.

clear_selection()

Deselects all the currently selected nodes.

deselect_node(node)

Deselects a possibly selected node.

It is called by the controller when it deselects a node and can also be called from the outside to deselect a node directly. The derived widget should overwrite this method and change the node to its unselected state when this is called

Parameters

node The node to be deselected.

Warning: This method must be called by the derived widget using *super* if it is over-written.

get_index_of_node(node, selectable_nodes)

(internal) Returns the index of the *node* within the *selectable_nodes* returned by *get_selectable_nodes()*.

get_selectable_nodes()

(internal) Returns a list of the nodes that can be selected. It can be overwritten by the derived widget to return the correct list.

This list is used to determine which nodes to select with group selection. E.g. the last element in the list will be selected when home is pressed, pagedown will move (or add to, if shift is held) the selection from the current position by negative *page_count* nodes starting from the position of the currently selected node in this list and so on. Still, nodes can be selected even if they are not in this list.

Note: It is safe to dynamically change this list including removing, adding, or re-arranging its elements. Nodes can be selected even if they are not on this list. And selected nodes removed from the list will remain selected until *deselect_node()* is called.

Warning: Layouts display their children in the reverse order. That is, the contents of *children* is displayed from right to left, bottom to top. Therefore, internally, the indices of the elements returned by this function are reversed to make it work by default for most layouts so that the final result is consistent e.g. home, although it will select the last element in this list visually, will select the first element when counting from top to bottom and left to right. If this behavior is not desired, a reversed list should be returned instead.

Defaults to returning *children*.

goto_node(*key*, *last_node*, *last_node_idx*)

(internal) Used by the controller to get the node at the position indicated by key. The key can be keyboard inputs, e.g. pageup, or scroll inputs from the mouse scroll wheel, e.g. scrollup. 'last_node' is the last node selected and is used to find the resulting node. For example, if the key is up, the returned node is one node up from the last node.

It can be overwritten by the derived widget.

Parameters

key str, the string used to find the desired node. It can be any of the keyboard keys, as well as the mouse scrollup, scrolldown, scrollright, and scrollleft strings. If letters are typed in quick succession, the letters will be combined before it's passed in as key and can be used to find nodes that have an associated string that starts with those letters.

last_node The last node that was selected.

last_node_idx The cached index of the last node selected in the *get_selectable_nodes()* list. If the list hasn't changed it saves having to look up the index of *last_node* in that list.

Returns tuple, the node targeted by key and its index in the *get_selectable_nodes()* list. Returning (*last_node*, *last_node_idx*) indicates a node wasn't found.

keyboard_select

Determines whether the keyboard can be used for selection. If False, keyboard inputs will be ignored.

keyboard_select is a *BooleanProperty* and defaults to True.

multiselect

Determines whether multiple nodes can be selected. If enabled, keyboard shift and ctrl selection, optionally combined with touch, for example, will be able to select multiple widgets in the normally expected manner. This dominates *touch_multiselect* when False.

multiselect is a *BooleanProperty* and defaults to False.

nodes_order_reversed

(Internal) Indicates whether the order of the nodes as displayed top- down is reversed compared to their order in *get_selectable_nodes()* (e.g. how the children property is reversed compared to how it's displayed).

page_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when pageup (or pagedown) is pressed.

page_count is a *NumericProperty* and defaults to 10.

right_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the right (or left) arrow on the keyboard is pressed.

right_count is a *NumericProperty* and defaults to 1.

scroll_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the mouse scroll wheel is scrolled.

right_count is a *NumericProperty* and defaults to 0.

select_node(*node*)

Selects a node.

It is called by the controller when it selects a node and can be called from the outside to select a node directly. The derived widget should overwrite this method and change the node state to selected when called.

Parameters

node The node to be selected.

Returns bool, True if the node was selected, False otherwise.

Warning: This method must be called by the derived widget using `super` if it is over-written.

select_with_key_down(*keyboard, scancode, codepoint, modifiers, **kwargs*)

Processes a key press. This is called when a key press is to be used for selection. Depending on the keyboard keys pressed and the configuration, it could select or deselect nodes or node ranges from the selectable nodes list, [`get_selectable_nodes\(\)`](#).

The parameters are such that it could be bound directly to the `on_key_down` event of a keyboard. Therefore, it is safe to be called repeatedly when the key is held down as is done by the keyboard.

Returns bool, True if the keypress was used, False otherwise.

select_with_key_up(*keyboard, scancode, **kwargs*)

(internal) Processes a key release. This must be called by the derived widget when a key that [`select_with_key_down\(\)`](#) returned True is released.

The parameters are such that it could be bound directly to the `on_key_up` event of a keyboard.

Returns bool, True if the key release was used, False otherwise.

select_with_touch(*node, touch=None*)

(internal) Processes a touch on the node. This should be called by the derived widget when a node is touched and is to be used for selection. Depending on the keyboard keys pressed and the configuration, it could select or deslect this and other nodes in the selectable nodes list, [`get_selectable_nodes\(\)`](#).

Parameters

node The node that received the touch. Can be None for a scroll type touch.

touch Optionally, the touch. Defaults to None.

Returns bool, True if the touch was used, False otherwise.

selected_nodes

The list of selected nodes.

Note: Multiple nodes can be selected right after one another e.g. using the keyboard. When listening to [`selected_nodes`](#), one should be aware of this.

[`selected_nodes`](#) is a [`ListProperty`](#) and defaults to the empty list, []. It is read-only and should not be modified.

text_entry_timeout

When typing characters in rapid sucession (i.e. the time difference since the last character is less than [`text_entry_timeout`](#)), the keys get concatenated and the combined text is passed as the key argument of [`goto_node\(\)`](#).

New in version 1.10.0.

touch_deselect_last

Determines whether the last selected node can be deselected when *multiselect* or *touch_multiselect* is False.

New in version 1.10.0.

touch_deselect_last is a *BooleanProperty* and defaults to True on mobile, False on desktop platforms.

touch_multiselect

A special touch mode which determines whether touch events, as processed by *select_with_touch()*, will add the currently touched node to the selection, or if it will clear the selection before adding the node. This allows the selection of multiple nodes by simply touching them.

This is different from *multiselect* because when it is True, simply touching an unselected node will select it, even if ctrl is not pressed. If it is False, however, ctrl must be pressed in order to add to the selection when *multiselect* is True.

Note: *multiselect*, when False, will disable *touch_multiselect*.

touch_multiselect is a *BooleanProperty* and defaults to False.

up_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the up (or down) arrow on the keyboard is pressed.

up_count is a *NumericProperty* and defaults to 1.

class kivy.uix.behaviors.CodeNavigationBehavior

Bases: *kivy.event.EventDispatcher*

Code navigation behavior. Modifies the navigation behavior in *TextInput* to work like an IDE instead of a word processor. Please see the *code navigation behaviors module* documentation for more information.

New in version 1.9.1.

class kivy.uix.behaviors.EmacsBehavior(kwargs)**

Bases: *builtins.object*

A *mixin* that enables Emacs-style keyboard shortcuts for the *TextInput* widget. Please see the *Emacs behaviors module* documentation for more information.

New in version 1.9.1.

delete_word_left()

Delete text left of the cursor to the beginning of word

delete_word_right()

Delete text right of the cursor to the end of the word

key_bindings

String name which determines the type of key bindings to use with the *TextInput*. This allows Emacs key bindings to be enabled/disabled programmatically for widgets that inherit from *EmacsBehavior*. If the value is not 'emacs', Emacs bindings will be disabled. Use 'default' for switching to the default key bindings of *TextInput*.

key_bindings is a *StringProperty* and defaults to 'emacs'.

New in version 1.10.0.

class kivy.uix.behaviors.CoverBehavior(kwargs)**

Bases: *builtins.object*

The CoverBehavior **mixin** provides rendering a texture covering full widget size keeping aspect ratio of the original texture.

New in version 1.10.0.

cover_pos

Position of the aspect ratio aware texture. Gets calculated in CoverBehavior.calculate_cover.

cover_pos is a *ListProperty* and defaults to [0, 0].

cover_size

Size of the aspect ratio aware texture. Gets calculated in CoverBehavior.calculate_cover.

cover_size is a *ListProperty* and defaults to [0, 0].

reference_size

Reference size used for aspect ratio approximation calculation.

reference_size is a *ListProperty* and defaults to [].

class kivy.uix.behaviors.TouchRippleBehavior(**kwargs)

Bases: builtins.object

Touch ripple behavior.

Supposed to be used as mixin on widget classes.

Ripple behavior does not trigger automatically, concrete implementation needs to call *ripple_show()* respective *ripple_fade()* manually.

Here we create a Label which renders the touch ripple animation on interaction:

```
class RippleLabel(TouchRippleBehavior, Label):

    def __init__(self, **kwargs):
        super(RippleLabel, self).__init__(**kwargs)

    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
```

ripple_duration_in

Animation duration taken to show the overlay.

ripple_duration_in is a *NumericProperty* and defaults to 0.5.

ripple_duration_out

Animation duration taken to fade the overlay.

ripple_duration_out is a *NumericProperty* and defaults to 0.2.

ripple_fade()

Finish ripple animation on current widget.

ripple_fade_from_alpha

Alpha channel for ripple color the animation starts with.

ripple_fade_from_alpha is a *NumericProperty* and defaults to 0.5.

ripple_fade_to_alpha

Alpha channel for ripple color the animation targets to.

ripple_fade_to_alpha is a *NumericProperty* and defaults to 0.8.

ripple_func_in

Animation callback for showing the overlay.

ripple_func_in is a *StringProperty* and defaults to *in_cubic*.

ripple_func_out

Animation callback for hiding the overlay.

ripple_func_out is a *StringProperty* and defaults to *out_quad*.

ripple_rad_default

Default radius the animation starts from.

ripple_rad_default is a *NumericProperty* and defaults to 10.

ripple_scale

Max scale of the animation overlay calculated from max(width/height) of the decorated widget.

ripple_scale is a *NumericProperty* and defaults to 2.0.

ripple_show(touch)

Begin ripple animation on current widget.

Expects touch event as argument.

class `kivy.uix.behaviors.TouchRippleButtonBehavior(**kwargs)`

Bases: *kivy.uix.behaviors.touchripple.TouchRippleBehavior*

This *mixin* class provides a similar behavior to *ButtonBehavior* but provides touch ripple animation instead of button pressed/released as visual effect.

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).

always_release

This determines whether or not the widget fires an *on_release* event if the touch_up is outside the widget.

always_release is a *BooleanProperty* and defaults to *False*.

last_touch

Contains the last relevant touch received by the Button. This can be used in *on_press* or *on_release* in order to know which touch dispatched the event.

last_touch is a *ObjectProperty* and defaults to *None*.

33.1.3 Button Behavior

The *ButtonBehavior* *mixin* class provides *Button* behavior. You can combine this class with other widgets, such as an *Image*, to provide alternative buttons that preserve Kivy button behavior.

For an overview of behaviors, please refer to the *behaviors* documentation.

Example

The following example adds button behavior to an image to make a checkbox that behaves like a button:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.uix.behaviors import ButtonBehavior

class MyButton(ButtonBehavior, Image):
    def __init__(self, **kwargs):
        super(MyButton, self).__init__(**kwargs)
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'

    def on_press(self):
        self.source = 'atlas://data/images/defaulttheme/checkbox_on'

    def on_release(self):
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'

class SampleApp(App):
    def build(self):
        return MyButton()

SampleApp().run()
```

See [ButtonBehavior](#) for details.

class kivy.uix.behaviors.button.ButtonBehavior(***kwargs*)
Bases: builtins.object

This [mixin](#) class provides [Button](#) behavior. Please see the [button behaviors module](#) documentation for more information.

Events

on_press Fired when the button is pressed.

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

always_release

This determines whether or not the widget fires an *on_release* event if the touch_up is outside the widget.

New in version 1.9.0.

Changed in version 1.10.0: The default value is now False.

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

last_touch

Contains the last relevant touch received by the Button. This can be used in *on_press* or *on_release* in order to know which touch dispatched the event.

New in version 1.8.0.

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

min_state_time

The minimum period of time which the widget must remain in the 'down' state.

New in version 1.9.1.

`min_state_time` is a float and defaults to 0.035. This value is taken from *Config*.

state

The state of the button, must be one of 'normal' or 'down'. The state is 'down' only when the button is currently touched/clicked, otherwise its 'normal'.

`state` is an *OptionProperty* and defaults to 'normal'.

trigger_action(*duration=0.1*)

Trigger whatever action(s) have been bound to the button by calling both the `on_press` and `on_release` callbacks.

This simulates a quick button press without using any touch events.

Duration is the length of the press in seconds. Pass 0 if you want the action to happen instantly.

New in version 1.8.0.

33.1.4 Code Navigation Behavior

The *CodeNavigationBehavior* modifies navigation behavior in the *TextInput*, making it work like an IDE instead of a word processor.

Using this mixin gives the *TextInput* the ability to recognize whitespace, punctuation and case variations (e.g. CamelCase) when moving over text. It is currently used by the *CodeInput* widget.

class `kivy.uix.behaviors.codenavigation.CodeNavigationBehavior`

Bases: *kivy.event.EventDispatcher*

Code navigation behavior. Modifies the navigation behavior in *TextInput* to work like an IDE instead of a word processor. Please see the *code navigation behaviors module* documentation for more information.

New in version 1.9.1.

33.1.5 Compound Selection Behavior

The *CompoundSelectionBehavior* *mixin* class implements the logic behind keyboard and touch selection of selectable widgets managed by the derived widget. For example, it can be combined with a *GridLayout* to add selection to the layout.

Compound selection concepts

At its core, it keeps a dynamic list of widgets that can be selected. Then, as the touches and keyboard input are passed in, it selects one or more of the widgets based on these inputs. For example, it uses the mouse scroll and keyboard up/down buttons to scroll through the list of widgets. Multiselection can also be achieved using the keyboard shift and ctrl keys.

Finally, in addition to the up/down type keyboard inputs, compound selection can also accept letters from the keyboard to be used to select nodes with associated strings that start with those letters, similar to how files are selected by a file browser.

Selection mechanics

When the controller needs to select a node, it calls `select_node()` and `deselect_node()`. Therefore, they must be overwritten in order alter node selection. By default, the class doesn't listen for keyboard or touch events, so the derived widget must call `select_with_touch()`,

`select_with_key_down()`, and `select_with_key_up()` on events that it wants to pass on for selection purposes.

Example

To add selection to a grid layout which will contain `Button` widgets. For each button added to the layout, you need to bind the *on_touch_down* of the button to `select_with_touch()` to pass on the touch events:

```
from kivy.uix.behaviors.compoundselection import CompoundSelectionBehavior
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.behaviors import FocusBehavior
from kivy.core.window import Window
from kivy.app import App

class SelectableGrid(FocusBehavior, CompoundSelectionBehavior, GridLayout):

    def keyboard_on_key_down(self, window, keycode, text, modifiers):
        """Based on FocusBehavior that provides automatic keyboard
        access, key presses will be used to select children.
        """
        if super(SelectableGrid, self).keyboard_on_key_down(
            window, keycode, text, modifiers):
            return True
        if self.select_with_key_down(window, keycode, text, modifiers):
            return True
        return False

    def keyboard_on_key_up(self, window, keycode):
        """Based on FocusBehavior that provides automatic keyboard
        access, key release will be used to select children.
        """
        if super(SelectableGrid, self).keyboard_on_key_up(window, keycode):
            return True
        if self.select_with_key_up(window, keycode):
            return True
        return False

    def add_widget(self, widget):
        """ Override the adding of widgets so we can bind and catch their
        *on_touch_down* events. """
        widget.bind(on_touch_down=self.button_touch_down,
                    on_touch_up=self.button_touch_up)
        return super(SelectableGrid, self).add_widget(widget)

    def button_touch_down(self, button, touch):
        """ Use collision detection to select buttons when the touch occurs
        within their area. """
        if button.collide_point(*touch.pos):
            self.select_with_touch(button, touch)

    def button_touch_up(self, button, touch):
        """ Use collision detection to de-select buttons when the touch
        occurs outside their area and *touch_multiselect* is not True. """
        if not (button.collide_point(*touch.pos) or
                self.touch_multiselect):
```

(continues on next page)


```

        self.deselect_node(button)

    def select_node(self, node):
        node.background_color = (1, 0, 0, 1)
        return super(SelectableGrid, self).select_node(node)

    def deselect_node(self, node):
        node.background_color = (1, 1, 1, 1)
        super(SelectableGrid, self).deselect_node(node)

    def on_selected_nodes(self, gird, nodes):
        print("Selected nodes = {}".format(nodes))

class TestApp(App):
    def build(self):
        grid = SelectableGrid(cols=3, rows=2, touch_multiselect=True,
                               multiselect=True)
        for i in range(0, 6):
            grid.add_widget(Button(text="Button {}".format(i)))
        return grid

TestApp().run()

```

Warning: This code is still experimental, and its API is subject to change in a future version.

class `kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior`(**kwargs)
 Bases: `builtins.object`

The Selection behavior `mixin` implements the logic behind keyboard and touch selection of selectable widgets managed by the derived widget. Please see the [compound selection behaviors module](#) documentation for more information.

New in version 1.9.0.

clear_selection()

Deselects all the currently selected nodes.

deselect_node(node)

Deselects a possibly selected node.

It is called by the controller when it deselects a node and can also be called from the outside to deselect a node directly. The derived widget should overwrite this method and change the node to its unselected state when this is called

Parameters

node The node to be deselected.

Warning: This method must be called by the derived widget using `super` if it is overwritten.

get_index_of_node(node, selectable_nodes)

(internal) Returns the index of the *node* within the *selectable_nodes* returned by [get_selectable_nodes\(\)](#).

get_selectable_nodes()

(internal) Returns a list of the nodes that can be selected. It can be overwritten by the derived widget to return the correct list.

This list is used to determine which nodes to select with group selection. E.g. the last element in the list will be selected when home is pressed, pagedown will move (or add to, if shift is held) the selection from the current position by negative *page_count* nodes starting from the position of the currently selected node in this list and so on. Still, nodes can be selected even if they are not in this list.

Note: It is safe to dynamically change this list including removing, adding, or re-arranging its elements. Nodes can be selected even if they are not on this list. And selected nodes removed from the list will remain selected until *deselect_node()* is called.

Warning: Layouts display their children in the reverse order. That is, the contents of *children* is displayed from right to left, bottom to top. Therefore, internally, the indices of the elements returned by this function are reversed to make it work by default for most layouts so that the final result is consistent e.g. home, although it will select the last element in this list visually, will select the first element when counting from top to bottom and left to right. If this behavior is not desired, a reversed list should be returned instead.

Defaults to returning *children*.

goto_node(key, last_node, last_node_idx)

(internal) Used by the controller to get the node at the position indicated by key. The key can be keyboard inputs, e.g. pageup, or scroll inputs from the mouse scroll wheel, e.g. scrollup. 'last_node' is the last node selected and is used to find the resulting node. For example, if the key is up, the returned node is one node up from the last node.

It can be overwritten by the derived widget.

Parameters

key str, the string used to find the desired node. It can be any of the keyboard keys, as well as the mouse scrollup, scrolldown, scrollright, and scrollleft strings. If letters are typed in quick succession, the letters will be combined before it's passed in as key and can be used to find nodes that have an associated string that starts with those letters.

last_node The last node that was selected.

last_node_idx The cached index of the last node selected in the *get_selectable_nodes()* list. If the list hasn't changed it saves having to look up the index of *last_node* in that list.

Returns tuple, the node targeted by key and its index in the *get_selectable_nodes()* list. Returning (*last_node*, *last_node_idx*) indicates a node wasn't found.

keyboard_select

Determines whether the keyboard can be used for selection. If False, keyboard inputs will be ignored.

keyboard_select is a *BooleanProperty* and defaults to True.

multiselect

Determines whether multiple nodes can be selected. If enabled, keyboard shift and ctrl se-

lection, optionally combined with touch, for example, will be able to select multiple widgets in the normally expected manner. This dominates *touch_multiselect* when False.

multiselect is a *BooleanProperty* and defaults to False.

nodes_order_reversed

(Internal) Indicates whether the order of the nodes as displayed top- down is reversed compared to their order in *get_selectable_nodes()* (e.g. how the children property is reversed compared to how it's displayed).

page_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when pageup (or pagedown) is pressed.

page_count is a *NumericProperty* and defaults to 10.

right_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the right (or left) arrow on the keyboard is pressed.

right_count is a *NumericProperty* and defaults to 1.

scroll_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the mouse scroll wheel is scrolled.

right_count is a *NumericProperty* and defaults to 0.

select_node(*node*)

Selects a node.

It is called by the controller when it selects a node and can be called from the outside to select a node directly. The derived widget should overwrite this method and change the node state to selected when called.

Parameters

node The node to be selected.

Returns bool, True if the node was selected, False otherwise.

Warning: This method must be called by the derived widget using super if it is overwritten.

select_with_key_down(*keyboard*, *scancode*, *codepoint*, *modifiers*, *kwargs*)**

Processes a key press. This is called when a key press is to be used for selection. Depending on the keyboard keys pressed and the configuration, it could select or deselect nodes or node ranges from the selectable nodes list, *get_selectable_nodes()*.

The parameters are such that it could be bound directly to the *on_key_down* event of a keyboard. Therefore, it is safe to be called repeatedly when the key is held down as is done by the keyboard.

Returns bool, True if the keypress was used, False otherwise.

select_with_key_up(*keyboard*, *scancode*, *kwargs*)**

(internal) Processes a key release. This must be called by the derived widget when a key that *select_with_key_down()* returned True is released.

The parameters are such that it could be bound directly to the *on_key_up* event of a keyboard.

Returns bool, True if the key release was used, False otherwise.

select_with_touch(*node*, *touch*=None)

(internal) Processes a touch on the node. This should be called by the derived widget when a node is touched and is to be used for selection. Depending on the keyboard keys pressed and the configuration, it could select or deslect this and other nodes in the selectable nodes list, *get_selectable_nodes()*.

Parameters

node The node that received the touch. Can be None for a scroll type touch.

touch Optionally, the touch. Defaults to None.

Returns bool, True if the touch was used, False otherwise.

selected_nodes

The list of selected nodes.

Note: Multiple nodes can be selected right after one another e.g. using the keyboard. When listening to *selected_nodes*, one should be aware of this.

selected_nodes is a *ListProperty* and defaults to the empty list, []. It is read-only and should not be modified.

text_entry_timeout

When typing characters in rapid sucession (i.e. the time difference since the last character is less than *text_entry_timeout*), the keys get concatenated and the combined text is passed as the key argument of *goto_node()*.

New in version 1.10.0.

touch_deselect_last

Determines whether the last selected node can be deselected when *multiselect* or *touch_multiselect* is False.

New in version 1.10.0.

touch_deselect_last is a *BooleanProperty* and defaults to True on mobile, False on desktop platforms.

touch_multiselect

A special touch mode which determines whether touch events, as processed by *select_with_touch()*, will add the currently touched node to the selection, or if it will clear the selection before adding the node. This allows the selection of multiple nodes by simply touching them.

This is different from *multiselect* because when it is True, simply touching an unselected node will select it, even if ctrl is not pressed. If it is False, however, ctrl must be pressed in order to add to the selection when *multiselect* is True.

Note: *multiselect*, when False, will disable *touch_multiselect*.

touch_multiselect is a *BooleanProperty* and defaults to False.

up_count

Determines by how much the selected node is moved up or down, relative to the position of the last selected node, when the up (or down) arrow on the keyboard is pressed.

up_count is a *NumericProperty* and defaults to 1.

33.1.6 Cover Behavior

The *CoverBehavior* mixin is intended for rendering textures to full widget size keeping the aspect ratio of the original texture.

Use cases are i.e. rendering full size background images or video content in a dynamic layout.

For an overview of behaviors, please refer to the *behaviors* documentation.

Example

The following examples add cover behavior to an image:

In python:

```
from kivy.app import App
from kivy.uix.behaviors import CoverBehavior
from kivy.uix.image import Image

class CoverImage(CoverBehavior, Image):

    def __init__(self, **kwargs):
        super(CoverImage, self).__init__(**kwargs)
        texture = self._coreimage.texture
        self.reference_size = texture.size
        self.texture = texture

class MainApp(App):

    def build(self):
        return CoverImage(source='image.jpg')

MainApp().run()
```

In Kivy Language:

```
CoverImage:
    source: 'image.png'

<CoverImage@CoverBehavior+Image>:
    reference_size: self.texture_size
```

See *CoverBehavior* for details.

class kivy.uix.behaviors.cover.CoverBehavior(**kwargs)
Bases: builtins.object

The CoverBehavior mixin provides rendering a texture covering full widget size keeping aspect ratio of the original texture.

New in version 1.10.0.

cover_pos

Position of the aspect ratio aware texture. Gets calculated in CoverBehavior.
calculate_cover.

cover_pos is a *ListProperty* and defaults to [0, 0].

cover_size

Size of the aspect ratio aware texture. Gets calculated in CoverBehavior.
calculate_cover.

cover_size is a *ListProperty* and defaults to *[0, 0]*.

reference_size

Reference size used for aspect ratio approximation calculation.

reference_size is a *ListProperty* and defaults to *[]*.

33.1.7 Drag Behavior

The *DragBehavior* mixin class provides Drag behavior. When combined with a widget, dragging in the rectangle defined by the *drag_rectangle* will drag the widget.

Example

The following example creates a draggable label:

```
from kivy.uix.label import Label
from kivy.app import App
from kivy.uix.behaviors import DragBehavior
from kivy.lang import Builder

# You could also put the following in your kv file...
kv = '''
<DragLabel>:
    # Define the properties for the DragLabel
    drag_rectangle: self.x, self.y, self.width, self.height
    drag_timeout: 10000000
    drag_distance: 0

FloatLayout:
    # Define the root widget
    DragLabel:
        size_hint: 0.25, 0.2
        text: 'Drag me'
...

class DragLabel(DragBehavior, Label):
    pass

class TestApp(App):
    def build(self):
        return Builder.load_string(kv)

TestApp().run()
```

```
class kivy.uix.behaviors.drag.DragBehavior(**kwargs)
```

Bases: builtins.object

The DragBehavior *mixin* provides Drag behavior. When combined with a widget, dragging in the rectangle defined by *drag_rectangle* will drag the widget. Please see the *drag behaviors module* documentation for more information.

New in version 1.8.0.

drag_distance

Distance to move before dragging the *DragBehavior*, in pixels. As soon as the distance has been traveled, the *DragBehavior* will start to drag, and no touch event will be dispatched to the children. It is advisable that you base this value on the dpi of your target device's screen.

drag_distance is a *NumericProperty* and defaults to the *scroll_distance* as defined in the user *Config* (20 pixels by default).

drag_rect_height

Height of the axis aligned bounding rectangle where dragging is allowed.

drag_rect_height is a *NumericProperty* and defaults to 100.

drag_rect_width

Width of the axis aligned bounding rectangle where dragging is allowed.

drag_rect_width is a *NumericProperty* and defaults to 100.

drag_rect_x

X position of the axis aligned bounding rectangle where dragging is allowed (in window coordinates).

drag_rect_x is a *NumericProperty* and defaults to 0.

drag_rect_y

Y position of the axis aligned bounding rectangle where dragging is allowed (in window coordinates).

drag_rect_y is a *NumericProperty* and defaults to 0.

drag_rectangle

Position and size of the axis aligned bounding rectangle where dragging is allowed.

drag_rectangle is a *ReferenceListProperty* of (*drag_rect_x*, *drag_rect_y*, *drag_rect_width*, *drag_rect_height*) properties.

drag_timeout

Timeout allowed to trigger the *drag_distance*, in milliseconds. If the user has not moved *drag_distance* within the timeout, dragging will be disabled, and the touch event will be dispatched to the children.

drag_timeout is a *NumericProperty* and defaults to the *scroll_timeout* as defined in the user *Config* (55 milliseconds by default).

33.1.8 Emacs Behavior

The *EmacsBehavior* mixin allows you to add *Emacs* keyboard shortcuts for basic movement and editing to the *TextInput* widget. The shortcuts currently available are listed below:

Emacs shortcuts

Shortcut	Description
Control + a	Move cursor to the beginning of the line
Control + e	Move cursor to the end of the line
Control + f	Move cursor one character to the right
Control + b	Move cursor one character to the left
Alt + f	Move cursor to the end of the word to the right
Alt + b	Move cursor to the start of the word to the left
Alt + Backspace	Delete text left of the cursor to the beginning of word
Alt + d	Delete text right of the cursor to the end of the word
Alt + w	Copy selection
Control + w	Cut selection
Control + y	Paste selection

Warning: If you have the *inspector* module enabled, the shortcut for opening the inspector (Control + e) conflicts with the Emacs shortcut to move to the end of the line (it will still move the cursor to the end of the line, but the inspector will open as well).

class `kivy.uix.behaviors.emacs.EmacsBehavior`(**kwargs)

Bases: `builtins.object`

A *mixin* that enables Emacs-style keyboard shortcuts for the *TextInput* widget. Please see the *Emacs behaviors module* documentation for more information.

New in version 1.9.1.

delete_word_left()

Delete text left of the cursor to the beginning of word

delete_word_right()

Delete text right of the cursor to the end of the word

key_bindings

String name which determines the type of key bindings to use with the *TextInput*. This allows Emacs key bindings to be enabled/disabled programmatically for widgets that inherit from *EmacsBehavior*. If the value is not 'emacs', Emacs bindings will be disabled. Use 'default' for switching to the default key bindings of *TextInput*.

key_bindings is a *StringProperty* and defaults to 'emacs'.

New in version 1.10.0.

33.1.9 Focus Behavior

The *FocusBehavior* *mixin* class provides keyboard focus behavior. When combined with other *FocusBehavior* widgets it allows one to cycle focus among them by pressing tab. In addition, upon gaining focus, the instance will automatically receive keyboard input.

Focus, very different from selection, is intimately tied with the keyboard; each keyboard can focus on zero or one widgets, and each widget can only have the focus of one keyboard. However, multiple keyboards can focus simultaneously on different widgets. When escape is hit, the widget having the focus of that keyboard will de-focus.

Managing focus

In essence, focus is implemented as a doubly linked list, where each node holds a (weak) reference to the instance before it and after it, as visualized when cycling through the nodes using `tab` (forward) or `shift+tab` (backward). If a previous or next widget is not specified, `focus_next` and `focus_previous` defaults to `None`. This means that the *children* list and *parents* are walked to find the next focusable widget, unless `focus_next` or `focus_previous` is set to the *StopIteration* class, in which case focus stops there.

For example, to cycle focus between *Button* elements of a *GridLayout*:

```
class FocusButton(FocusBehavior, Button):
    pass

grid = GridLayout(cols=4)
for i in range(40):
    grid.add_widget(FocusButton(text=str(i)))
# clicking on a widget will activate focus, and tab can now be used
# to cycle through
```

When using a software keyboard, typical on mobile and touch devices, the keyboard display behavior is determined by the *softinput_mode* property. You can use this property to ensure the focused widget is not covered or obscured by the keyboard.

Initializing focus

Widgets need to be visible before they can receive the focus. This means that setting their *focus* property to `True` before they are visible will have no effect. To initialize focus, you can use the `'on_parent'` event:

```
from kivy.app import App
from kivy.uix.textinput import TextInput

class MyTextInput(TextInput):
    def on_parent(self, widget, parent):
        self.focus = True

class SampleApp(App):
    def build(self):
        return MyTextInput()

SampleApp().run()
```

If you are using a *popup*, you can use the `'on_open'` event.

For an overview of behaviors, please refer to the *behaviors* documentation.

Warning: This code is still experimental, and its API is subject to change in a future version.

```
class kivy.uix.behaviors.focus.FocusBehavior(**kwargs)
    Bases: builtins.object
```

Provides keyboard focus behavior. When combined with other *FocusBehavior* widgets it allows one to cycle focus among them by pressing `tab`. Please see the *focus behavior module documentation* for more information.

New in version 1.9.0.

focus

Whether the instance currently has focus.

Setting it to True will bind to and/or request the keyboard, and input will be forwarded to the instance. Setting it to False will unbind and/or release the keyboard. For a given keyboard, only one widget can have its focus, so focusing one will automatically unfocus the other instance holding its focus.

When using a software keyboard, please refer to the *softinput_mode* property to determine how the keyboard display is handled.

focus is a *BooleanProperty* and defaults to False.

focus_next

The *FocusBehavior* instance to acquire focus when tab is pressed and this instance has focus, if not *None* or *StopIteration*.

When tab is pressed, focus cycles through all the *FocusBehavior* widgets that are linked through *focus_next* and are focusable. If *focus_next* is *None*, it instead walks the children lists to find the next focusable widget. Finally, if *focus_next* is the *StopIteration* class, focus won't move forward, but end here.

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

focus_previous

The *FocusBehavior* instance to acquire focus when shift+tab is pressed on this instance, if not *None* or *StopIteration*.

When shift+tab is pressed, focus cycles through all the *FocusBehavior* widgets that are linked through *focus_previous* and are focusable. If *focus_previous* is *None*, it instead walks the children tree to find the previous focusable widget. Finally, if *focus_previous* is the *StopIteration* class, focus won't move backward, but end here.

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

focused

An alias of *focus*.

focused is a *BooleanProperty* and defaults to False.

Warning: *focused* is an alias of *focus* and will be removed in 2.0.0.

get_focus_next()

Returns the next focusable widget using either *focus_next* or the children similar to the order when tabbing forwards with the tab key.

get_focus_previous()

Returns the previous focusable widget using either *focus_previous* or the children similar to the order when tab + shift key are triggered together.

hide_keyboard()

Convenience function to hide the keyboard in managed mode.

ignored_touch = []

A list of touches that should not be used to defocus. After *on_touch_up*, every touch that is not in *ignored_touch* will defocus all the focused widgets if the config keyboard mode is not multi. Touches on focusable widgets that were used to focus are automatically added here.

Example usage:

```
class Unfocusable(Widget):

    def on_touch_down(self, touch):
        if self.collide_point(*touch.pos):
            FocusBehavior.ignored_touch.append(touch)
```

Notice that you need to access this as a class, not an instance variable.

input_type

The kind of input keyboard to request.

New in version 1.8.0.

input_type is an *OptionsProperty* and defaults to 'text'. Can be one of 'text', 'number', 'url', 'mail', 'datetime', 'tel' or 'address'.

is_focusable

Whether the instance can become focused. If focused, it'll lose focus when set to False.

is_focusable is a *BooleanProperty* and defaults to True on a desktop (i.e. *desktop* is True in *config*), False otherwise.

keyboard

The keyboard to bind to (or bound to the widget) when focused.

When None, a keyboard is requested and released whenever the widget comes into and out of focus. If not None, it must be a keyboard, which gets bound and unbound from the widget whenever it's in or out of focus. It is useful only when more than one keyboard is available, so it is recommended to be set to None when only one keyboard is available.

If more than one keyboard is available, whenever an instance gets focused a new keyboard will be requested if None. Unless the other instances lose focus (e.g. if tab was used), a new keyboard will appear. When this is undesired, the keyboard property can be used. For example, if there are two users with two keyboards, then each keyboard can be assigned to different groups of instances of FocusBehavior, ensuring that within each group, only one FocusBehavior will have focus, and will receive input from the correct keyboard. See *keyboard_mode* in *config* for more information on the keyboard modes.

Keyboard and focus behavior

When using the keyboard, there are some important default behaviors you should keep in mind.

- When Config's *keyboard_mode* is multi, each new touch is considered a touch by a different user and will set the focus (if clicked on a focusable) with a new keyboard. Already focused elements will not lose their focus (even if an unfocusable widget is touched).
- If the keyboard property is set, that keyboard will be used when the instance gets focused. If widgets with different keyboards are linked through *focus_next* and *focus_previous*, then as they are tabbed through, different keyboards will become active. Therefore, typically it's undesirable to link instances which are assigned different keyboards.
- When a widget has focus, setting its keyboard to None will remove its keyboard, but the widget will then immediately try to get another keyboard. In order to remove its keyboard, rather set its *focus* to False.
- When using a software keyboard, typical on mobile and touch devices, the keyboard display behavior is determined by the *softinput_mode* property. You can use this property to ensure the focused widget is not covered or obscured.

keyboard is an *AliasProperty* and defaults to None.

keyboard_mode

Determines how the keyboard visibility should be managed. 'auto' will result in the standard behaviour of showing/hiding on focus. 'managed' requires setting the keyboard visibility manually, or calling the helper functions `show_keyboard()` and `hide_keyboard()`.

`keyboard_mode` is an `OptionsProperty` and defaults to 'auto'. Can be one of 'auto' or 'managed'.

keyboard_on_key_down(*window, keycode, text, modifiers*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input press. The parameters are the same as `kivy.core.window.WindowBase.on_key_down()`.

When overwriting the method in the derived widget, super should be called to enable tab cycling. If the derived widget wishes to use tab for its own purposes, it can call super after it has processed the character (if it does not wish to consume the tab).

Similar to other keyboard functions, it should return True if the key was consumed.

keyboard_on_key_up(*window, keycode*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input release. The parameters are the same as `kivy.core.window.WindowBase.on_key_up()`.

When overwriting the method in the derived widget, super should be called to enable de-focusing on escape. If the derived widget wishes to use escape for its own purposes, it can call super after it has processed the character (if it does not wish to consume the escape).

See `keyboard_on_key_down()`

show_keyboard()

Convenience function to show the keyboard in managed mode.

unfocus_on_touch

Whether a instance should lose focus when clicked outside the instance.

When a user clicks on a widget that is focus aware and shares the same keyboard as this widget (which in the case of only one keyboard, are all focus aware widgets), then as the other widgets gains focus, this widget loses focus. In addition to that, if this property is `True`, clicking on any widget other than this widget, will remove focus from this widget.

`unfocus_on_touch` is a `BooleanProperty` and defaults to `False` if the `keyboard_mode` in `Config` is 'multi' or 'systemandmulti', otherwise it defaults to `True`.

33.1.10 Kivy Namespaces

New in version 1.9.1.

Warning: This code is still experimental, and its API is subject to change in a future version.

The `KNSpaceBehavior` mixin class provides namespace functionality for Kivy objects. It allows kivy objects to be named and then accessed using namespaces.

`KNSpace` instances are the namespaces that store the named objects in Kivy `ObjectProperty` instances. In addition, when inheriting from `KNSpaceBehavior`, if the derived object is named, the name will automatically be added to the associated namespace and will point to a `proxy_ref` of the derived object.

Basic examples

By default, there's only a single namespace: the *knospace* namespace. The simplest example is adding a widget to the namespace:

```
from kivy.uix.behaviors.knospace import knospace
widget = Widget()
knospace.my_widget = widget
```

This adds a kivy *ObjectProperty* with *rebind=True* and *allownone=True* to the *knospace* namespace with a property name *my_widget*. And the property now also points to this widget.

This can be done automatically with:

```
class MyWidget(KNSpaceBehavior, Widget):
    pass

widget = MyWidget(knsname='my_widget')
```

Or in kv:

```
<MyWidget@KNSpaceBehavior+Widget>

MyWidget:
    knsname: 'my_widget'
```

Now, *knospace.my_widget* will point to that widget.

When one creates a second widget with the same name, the namespace will also change to point to the new widget. E.g.:

```
widget = MyWidget(knsname='my_widget')
# knospace.my_widget now points to widget
widget2 = MyWidget(knsname='my_widget')
# knospace.my_widget now points to widget2
```

Setting the namespace

One can also create one's own namespace rather than using the default *knospace* by directly setting *KNSpaceBehavior.knospace*:

```
class MyWidget(KNSpaceBehavior, Widget):
    pass

widget = MyWidget(knsname='my_widget')
my_new_namespace = KNSpace()
widget.knospace = my_new_namespace
```

Initially, *my_widget* is added to the default namespace, but when the widget's namespace is changed to *my_new_namespace*, the reference to *my_widget* is moved to that namespace. We could have also of course first set the namespace to *my_new_namespace* and then have named the widget *my_widget*, thereby avoiding the initial assignment to the default namespace.

Similarly, in kv:

```
<MyWidget@KNSpaceBehavior+Widget>

MyWidget:
```

(continues on next page)

```

knspace: KNSpace()
knsname: 'my_widget'

```

Inheriting the namespace

In the previous example, we directly set the namespace we wished to use. In the following example, we inherit it from the parent, so we only have to set it once:

```

<MyWidget@KNSpaceBehavior+Widget>
<MyLabel@KNSpaceBehavior+Label>

<MyComplexWidget@MyWidget>:
    knsname: 'my_complex'
    MyLabel:
        knsname: 'label1'
    MyLabel:
        knsname: 'label2'

```

Then, we do:

```

widget = MyComplexWidget()
new_knspace = KNSpace()
widget.knspace = new_knspace

```

The rule is that if no *knspace* has been assigned to a widget, it looks for a namespace in its parent and parent's parent and so on until it find one to use. If none are found, it uses the default *knspace*.

When *MyComplexWidget* is created, it still used the default namespace. However, when we assigned the root widget its new namespace, all its children switched to using that new namespace as well. So *new_knspace* now contains *label1* and *label2* as well as *my_complex*.

If we had first done:

```

widget = MyComplexWidget()
new_knspace = KNSpace()
knspace.label1.knspace = knspace
widget.knspace = new_knspace

```

Then *label1* would remain stored in the default *knspace* since it was directly set, but *label2* and *my_complex* would still be added to the new namespace.

One can customize the attribute used to search the parent tree by changing *KNSpaceBehavior.knspace_key*. If the desired *knspace* is not reachable through a widget's parent tree, e.g. in a popup that is not a widget's child, *KNSpaceBehavior.knspace_key* can be used to establish a different search order.

Accessing the namespace

As seen in the previous example, if not directly assigned, the namespace is found by searching the parent tree. Consequently, if a namespace was assigned further up the parent tree, all its children and below could access that namespace through their *KNSpaceBehavior.knspace* property.

This allows the creation of multiple widgets with identically given names if each root widget instance is assigned a new namespace. For example:

```

<MyComplexWidget@KNSpaceBehavior+Widget>:
  Label:
    text: root.knspace.pretty.text if root.knspace.pretty else ''

<MyPrettyWidget@KNSpaceBehavior+TextInput>:
  knsname: 'pretty'
  text: 'Hello'

<MyCompositeWidget@KNSpaceBehavior+BoxLayout>:
  MyComplexWidget
  MyPrettyWidget

```

Now, when we do:

```

knspace1, knspace2 = KNSpace(), KNSpace()
composite1 = MyCompositeWidget()
composite1.knspace = knspace1

composite2 = MyCompositeWidget()
composite2.knspace = knspace2

knspace1.pretty = "Here's the ladder, now fix the roof!"
knspace2.pretty = "Get that raccoon off me!"

```

Because each of the *MyCompositeWidget* instances have a different namespace their children also use different namespaces. Consequently, the pretty and complex widgets of each instance will have different text.

Further, because both the namespace *ObjectProperty* references, and *KNSpaceBehavior.knspace* have *rebind=True*, the text of the *MyComplexWidget* label is rebound to match the text of *MyPrettyWidget* when either the root's namespace changes or when the *root.knspace.pretty* property changes, as expected.

Forking a namespace

Forking a namespace provides the opportunity to create a new namespace from a parent namespace so that the forked namespace will contain everything in the origin namespace, but the origin namespace will not have access to anything added to the forked namespace.

For example:

```

child = knspace.fork()
grandchild = child.fork()

child.label = Label()
grandchild.button = Button()

```

Now label is accessible by both child and grandchild, but not by knspace. And button is only accessible by the grandchild but not by the child or by knspace. Finally, doing *grandchild.label = Label()* will leave *grandchild.label* and *child.label* pointing to different labels.

A motivating example is the example from above:

```

<MyComplexWidget@KNSpaceBehavior+Widget>:
  Label:
    text: root.knspace.pretty.text if root.knspace.pretty else ''

<MyPrettyWidget@KNSpaceBehavior+TextInput>:

```

(continues on next page)

```

    knsname: 'pretty'
    text: 'Hello'

<MyCompositeWidget@KNSpaceBehavior+BoxLayout>:
    knspace: 'fork'
    MyComplexWidget
    MyPrettyWidget

```

Notice the addition of *knspace: 'fork'*. This is identical to doing *knspace: self.knspace.fork()*. However, doing that would lead to infinite recursion as that kv rule would be executed recursively because *self.knspace* will keep on changing. However, allowing *knspace: 'fork'* circumvents that. See [*KNSpaceBehavior.knspace*](#).

Now, having forked, we just need to do:

```

composite1 = MyCompositeWidget()
composite2 = MyCompositeWidget()

composite1.knspace.pretty = "Here's the ladder, now fix the roof!"
composite2.knspace.pretty = "Get that raccoon off me!"

```

Since by forking we automatically created a unique namespace for each *MyCompositeWidget* instance.

class *kivy.uix.behaviors.knspace.KNSpace*(*parent=None, keep_ref=False, **kwargs*)
 Bases: [*kivy.event.EventDispatcher*](#)

Each *KNSpace* instance is a namespace that stores the named Kivy objects associated with this namespace. Each named object is stored as the value of a Kivy *ObjectProperty* of this instance whose property name is the object's given name. Both *rebind* and *allownone* are set to *True* for the property.

See [*KNSpaceBehavior.knspace*](#) for details on how a namespace is associated with a named object.

When storing an object in the namespace, the object's *proxy_ref* is stored if the object has such an attribute.

Parameters

parent: (internal) A *KNSpace* instance or None. If specified, it's a parent namespace, in which case, the current namespace will have in its namespace all its named objects as well as the named objects of its parent and parent's parent etc. See [*fork\(\)*](#) for more details.

fork()

Returns a new *KNSpace* instance which will have access to all the named objects in the current namespace but will also have a namespace of its own that is unique to it.

For example:

```

forked_knspace1 = knspace.fork()
forked_knspace2 = knspace.fork()

```

Now, any names added to *knspace* will be accessible by the *forked_knspace1* and *forked_knspace2* namespaces by the normal means. However, any names added to *forked_knspace1* will not be accessible from *knspace* or *forked_knspace2*. Similar for *forked_knspace2*.

keep_ref = False

Whether a direct reference should be kept to the stored objects. If *True*, we use the direct object, otherwise we use *proxy_ref* when present.

Defaults to *False*.

parent = None

(internal) The parent namespace instance, *KNSpace*, or None. See *fork()*.

property (*self, name, quiet=False*)

Get a property instance from the property name. If quiet is True, None is returned instead of raising an exception when *name* is not a property. Defaults to *False*.

New in version 1.0.9.

Returns A *Property* derived instance corresponding to the name.

Changed in version 1.9.0: quiet was added.

class kivy.uix.behaviors.namespace.KNSpaceBehavior (*knspace=None, **kwargs*)

Bases: *builtins.object*

Inheriting from this class allows naming of the inherited objects, which are then added to the associated namespace *knspace* and accessible through it.

Please see the *knspace behaviors module* documentation for more information.

knsname

The name given to this instance. If named, the name will be added to the associated *knspace* namespace, which will then point to the *proxy_ref* of this instance.

When named, one can access this object by e.g. *self.knspace.name*, where *name* is the given name of this instance. See *knspace* and the module description for more details.

knspace

The namespace instance, *KNSpace*, associated with this widget. The *knspace* namespace stores this widget when naming this widget with *knsname*.

If the namespace has been set with a *KNSpace* instance, e.g. with *self.knspace = KNSpace()*, then that instance is returned (setting with *None* doesn't count). Otherwise, if *knspace_key* is not None, we look for a namespace to use in the object that is stored in the property named *knspace_key*, of this instance. I.e. *object = getattr(self, self.knspace_key)*.

If that object has a *knspace* property, then we return its value. Otherwise, we go further up, e.g. with *getattr(object, self.knspace_key)* and look for its *knspace* property.

Finally, if we reach a value of *None*, or *knspace_key* was *None*, the default *knspace* namespace is returned.

If *knspace* is set to the string 'fork', the current namespace in *knspace* will be forked with *KNSpace.fork()* and the resulting namespace will be assigned to this instance's *knspace*. See the module examples for a motivating example.

Both *rebind* and *allownone* are *True*.

knspace_key

The name of the property of this instance, to use to search upwards for a namespace to use by this instance. Defaults to 'parent' so that we'll search the parent tree. See *knspace*.

When *None*, we won't search the parent tree for the namespace. *allownone* is *True*.

kivy.uix.behaviors.namespace.knspace = <kivy.uix.behaviors.namespace.KNSpace object>

The default *KNSpace* namespace. See *KNSpaceBehavior.knspace* for more details.

33.1.11 ToggleButton Behavior

The *ToggleButtonBehavior* mixin class provides *ToggleButton* behavior. You can combine this class with other widgets, such as an *Image*, to provide alternative togglebuttons that preserve Kivy togglebutton behavior.

For an overview of behaviors, please refer to the *behaviors* documentation.

Example

The following example adds togglebutton behavior to an image to make a checkbox that behaves like a togglebutton:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.uix.behaviors import ToggleButtonBehavior

class MyButton(ToggleButtonBehavior, Image):
    def __init__(self, **kwargs):
        super(MyButton, self).__init__(**kwargs)
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'

    def on_state(self, widget, value):
        if value == 'down':
            self.source = 'atlas://data/images/defaulttheme/checkbox_on'
        else:
            self.source = 'atlas://data/images/defaulttheme/checkbox_off'

class SampleApp(App):
    def build(self):
        return MyButton()

SampleApp().run()
```

class `kivy.uix.behaviors.togglebutton.ToggleButtonBehavior`(**kwargs)
Bases: `kivy.uix.behaviors.button.ButtonBehavior`

This *mixin* class provides *togglebutton* behavior. Please see the *togglebutton behaviors module* documentation for more information.

New in version 1.8.0.

allow_no_selection

This specifies whether the widgets in a group allow no selection i.e. everything to be deselected.

New in version 1.9.0.

allow_no_selection is a `BooleanProperty` and defaults to *True*

static get_widgets(groupname)

Return a list of the widgets contained in a specific group. If the group doesn't exist, an empty list will be returned.

Note: Always release the result of this method! Holding a reference to any of these widgets can prevent them from being garbage collected. If in doubt, do:

```
l = ToggleButtonBehavior.get_widgets('mygroup')
# do your job
del l
```

Warning: It's possible that some widgets that you have previously deleted are still in the list. The garbage collector might need to release other objects before flushing them.

group

Group of the button. If *None*, no group will be used (the button will be independent). If specified, *group* must be a hashable object, like a string. Only one button in a group can be in a 'down' state.

group is a *ObjectProperty* and defaults to *None*.

33.1.12 Touch Ripple

New in version 1.10.1.

Warning: This code is still experimental, and its API is subject to change in a future version.

This module contains *mixin* classes to add a touch ripple visual effect known from *Google Material Design* <https://en.wikipedia.org/wiki/Material_Design>_ to widgets.

For an overview of behaviors, please refer to the *behaviors* documentation.

The class *TouchRippleBehavior* provides rendering the ripple animation.

The class *TouchRippleButtonBehavior* basically provides the same functionality as *ButtonBehavior* but rendering the ripple animation instead of default press/release visualization.

class `kivy.uix.behaviors.touchripple.TouchRippleBehavior(**kwargs)`
Bases: `builtins.object`

Touch ripple behavior.

Supposed to be used as mixin on widget classes.

Ripple behavior does not trigger automatically, concrete implementation needs to call *ripple_show()* respective *ripple_fade()* manually.

Here we create a Label which renders the touch ripple animation on interaction:

```
class RippleLabel(TouchRippleBehavior, Label):

    def __init__(self, **kwargs):
        super(RippleLabel, self).__init__(**kwargs)

    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
```

ripple_duration_in

Animation duration taken to show the overlay.

ripple_duration_in is a *NumericProperty* and defaults to 0.5.

ripple_duration_out

Animation duration taken to fade the overlay.

ripple_duration_out is a *NumericProperty* and defaults to 0.2.

ripple_fade()

Finish ripple animation on current widget.

ripple_fade_from_alpha

Alpha channel for ripple color the animation starts with.

ripple_fade_from_alpha is a *NumericProperty* and defaults to 0.5.

ripple_fade_to_alpha

Alpha channel for ripple color the animation targets to.

ripple_fade_to_alpha is a *NumericProperty* and defaults to 0.8.

ripple_func_in

Animation callback for showing the overlay.

ripple_func_in is a *StringProperty* and defaults to *in_cubic*.

ripple_func_out

Animation callback for hiding the overlay.

ripple_func_out is a *StringProperty* and defaults to *out_quad*.

ripple_rad_default

Default radius the animation starts from.

ripple_rad_default is a *NumericProperty* and defaults to 10.

ripple_scale

Max scale of the animation overlay calculated from max(width/height) of the decorated widget.

ripple_scale is a *NumericProperty* and defaults to 2.0.

ripple_show(*touch*)

Begin ripple animation on current widget.

Expects touch event as argument.

class `kivy.uix.behaviors.touchripple.TouchRippleButtonBehavior` (**kwargs)

Bases: *kivy.uix.behaviors.touchripple.TouchRippleBehavior*

This *mixin* class provides a similar behavior to *ButtonBehavior* but provides touch ripple animation instead of button pressed/released as visual effect.

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).

always_release

This determines whether or not the widget fires an *on_release* event if the touch_up is outside the widget.

always_release is a *BooleanProperty* and defaults to *False*.

last_touch

Contains the last relevant touch received by the Button. This can be used in *on_press* or *on_release* in order to know which touch dispatched the event.

last_touch is a *ObjectProperty* and defaults to *None*.

33.2 RecycleView

New in version 1.10.0.

The RecycleView provides a flexible model for viewing selected sections of large data sets. It aims to prevent the performance degradation that can occur when generating large numbers of widgets in order to display many data items.

The view is generated by processing the *data*, essentially a list of dicts, and uses these dicts to generate instances of the *viewclass* as required. Its design is based on the MVC (*Model-view-controller*) pattern.

- Model: The model is formed by *data* you pass in via a list of dicts.
- View: The View is split across layout and views and implemented using adapters.
- Controller: The controller determines the logical interaction and is implemented by *RecycleViewBehavior*.

These are abstract classes and cannot be used directly. The default concrete implementations are the *RecycleDataModel* for the model, the *RecycleLayout* for the view, and the *RecycleView* for the controller.

When a RecycleView is instantiated, it automatically creates the views and data classes. However, one must manually create the layout classes and add them to the RecycleView.

A layout manager is automatically created as a *layout_manager* when added as the child of the RecycleView. Similarly when removed. A requirement is that the layout manager must be contained as a child somewhere within the RecycleView's widget tree so the view port can be found.

A minimal example might look something like this:

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.recycleview import RecycleView

Builder.load_string('''
<RV>:
    viewclass: 'Label'
    RecycleBoxLayout:
        default_size: None, dp(56)
        default_size_hint: 1, None
        size_hint_y: None
        height: self.minimum_height
        orientation: 'vertical'
''')

class RV(RecycleView):
    def __init__(self, **kwargs):
        super(RV, self).__init__(**kwargs)
        self.data = [{'text': str(x)} for x in range(100)]

class TestApp(App):
    def build(self):
        return RV()

if __name__ == '__main__':
    TestApp().run()
```

In order to support selection in the view, you can add the required behaviours as follows:

```

from kivy.app import App
from kivy.lang import Builder
from kivy.uix.recycleview import RecycleView
from kivy.uix.recycleview.views import RecycleDataViewBehavior
from kivy.uix.label import Label
from kivy.properties import BooleanProperty
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.uix.behaviors import FocusBehavior
from kivy.uix.recycleview.layout import LayoutSelectionBehavior

Builder.load_string('''
<SelectableLabel>:
    # Draw a background to indicate selection
    canvas.before:
        Color:
            rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
        Rectangle:
            pos: self.pos
            size: self.size
<RV>:
    viewclass: 'SelectableLabel'
    SelectableRecycleBoxLayout:
        default_size: None, dp(56)
        default_size_hint: 1, None
        size_hint_y: None
        height: self.minimum_height
        orientation: 'vertical'
        multiselect: True
        touch_multiselect: True
''')

class SelectableRecycleBoxLayout(FocusBehavior, LayoutSelectionBehavior,
                                RecycleBoxLayout):
    ''' Adds selection and focus behaviour to the view. '''

class SelectableLabel(RecycleDataViewBehavior, Label):
    ''' Add selection support to the Label '''
    index = None
    selected = BooleanProperty(False)
    selectable = BooleanProperty(True)

    def refresh_view_attrs(self, rv, index, data):
        ''' Catch and handle the view changes '''
        self.index = index
        return super(SelectableLabel, self).refresh_view_attrs(
            rv, index, data)

    def on_touch_down(self, touch):
        ''' Add selection on touch down '''
        if super(SelectableLabel, self).on_touch_down(touch):
            return True
        if self.collide_point(*touch.pos) and self.selectable:
            return self.parent.select_with_touch(self.index, touch)

    def apply_selection(self, rv, index, is_selected):
        ''' Respond to the selection of items in the view. '''
        self.selected = is_selected

```

(continues on next page)

```

    if is_selected:
        print("selection changed to {}".format(rv.data[index]))
    else:
        print("selection removed for {}".format(rv.data[index]))

class RV(RecyclerView):
    def __init__(self, **kwargs):
        super(RV, self).__init__(**kwargs)
        self.data = [{'text': str(x)} for x in range(100)]

class TestApp(App):
    def build(self):
        return RV()

if __name__ == '__main__':
    TestApp().run()

```

Please see the *examples/widgets/recycleview/basic_data.py* file for a more complete example.

TODO:

- Method to clear cached class instances.
- Test when views cannot be found (e.g. viewclass is None).
- Fix selection goto.

Warning: When views are re-used they may not trigger if the data remains the same.

class kivy.uix.recycleview.**RecycleViewBehavior**(**kwargs)

Bases: `builtins.object`

RecycleViewBehavior provides a behavioral model upon which the *RecycleView* is built. Together, they offer an extensible and flexible way to produce views with limited windows over large data sets.

See the module documentation for more information.

data_model

The Data model responsible for maintaining the data set.

data_model is an *AliasProperty* that gets and sets the current data model.

layout_manager

The Layout manager responsible for positioning views within the *RecycleView*.

layout_manager is an *AliasProperty* that gets and sets the layout_manger.

refresh_from_data(*largs, **kwargs)

This should be called when data changes. Data changes typically indicate that everything should be recomputed since the source data changed.

This method is automatically bound to the `on_data_changed` method of the `RecycleDataModelBehavior` class and therefore responds to and accepts the keyword arguments of that event.

It can be called manually to trigger an update.

refresh_from_layout(*largs, **kwargs)

This should be called when the layout changes or needs to change. It is typically called when a layout parameter has changed and therefore the layout needs to be recomputed.

refresh_from_viewport(*largs)

This should be called when the viewport changes and the displayed data must be updated. Neither the data nor the layout will be recomputed.

view_adapter

The adapter responsible for providing views that represent items in a data set.

view_adapter is an *AliasProperty* that gets and sets the current view adapter.

class kivy.uix.recycleview.**RecycleView**(*kwargs)

Bases: *kivy.uix.recycleview.RecycleViewBehavior*, *kivy.uix.scrollview.ScrollView*

RecycleView is a flexible view for providing a limited window into a large data set.

See the module documentation for more information.

add_widget(widget, *largs)

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)
```

data

The data used by the current view adapter. This is a list of dicts whose keys map to the corresponding property names of the *viewclass*.

data is an *AliasProperty* that gets and sets the data used to generate the views.

key_viewclass

key_viewclass is an *AliasProperty* that gets and sets the key viewclass for the current layout_manager.

remove_widget(widget, *largs)

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)
```

viewclass

The viewclass used by the current layout_manager.

viewclass is an *AliasProperty* that gets and sets the class used to generate the individual items presented in the view.

33.2.1 RecycleView Data Model

New in version 1.10.0.

The data model part of the RecycleView model-view-controller pattern.

It defines the models (classes) that store the data associated with a *RecycleViewBehavior*. Each model (class) determines how the data is stored and emits requests to the controller (*RecycleViewBehavior*) when the data is modified.

class kivy.uix.recycleview.datamodel.**RecycleDataModelBehavior**

Bases: builtins.object

RecycleDataModelBehavior is the base class for the models that describes and provides the data for the *RecycleViewBehavior*.

Events

on_data_changed: Fired when the data changes. The event may dispatch keyword arguments specific to each implementation of the data model. When dispatched, the event and keyword arguments are forwarded to *refresh_from_data()*.

attach_recycleview(rv)

Associates a *RecycleViewBehavior* with this data model.

detach_recycleview()

Removes the *RecycleViewBehavior* associated with this data model.

recycleview

The *RecycleViewBehavior* instance associated with this data model.

class kivy.uix.recycleview.datamodel.**RecycleDataModel**(**kwargs)

Bases: *kivy.uix.recycleview.datamodel.RecycleDataModelBehavior*, *kivy.event.EventDispatcher*

An implementation of *RecycleDataModelBehavior* that keeps the data in a indexable list. See *data*.

When data changes this class currently dispatches *on_data_changed* with one of the following additional keyword arguments.

none: no keyword argument With no additional argument it means a generic data change.

removed: a slice or integer The value is a slice or integer indicating the indices removed.

appended: a slice The slice in *data* indicating the first and last new items (i.e. the slice pointing to the new items added at the end).

inserted: a integer The index in *data* where a new data item was inserted.

modified: a slice The slice with the indices where the data has been modified. This currently does not allow changing of size etc.

attach_recycleview(rv)

Associates a *RecycleViewBehavior* with this data model.

data

Stores the model's data using a list.

The data for a item at index *i* can also be accessed with *RecycleDataModel* [*i*].

detach_recycleview()

Removes the *RecycleViewBehavior* associated with this data model.

observable_dict

A dictionary instance, which when modified will trigger a *data* and consequently an *on_data_changed* dispatch.

33.2.2 RecycleView Layouts

New in version 1.10.0.

The Layouts handle the presentation of views for the *RecycleView*.

Warning: This module is highly experimental, its API may change in the future and the documentation is not complete at this time.

exception `kivy.uix.recycleview.layout.LayoutChangeException`

Bases: `Exception`

class `kivy.uix.recycleview.layout.LayoutSelectionBehavior(**kwargs)`

Bases: *kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior*

The *LayoutSelectionBehavior* can be combined with *RecycleLayoutManagerBehavior* to allow its derived classes selection behaviors similarly to how *CompoundSelectionBehavior* can be used to add selection behaviors to normal layout.

RecycleLayoutManagerBehavior manages its children differently than normal layouts or widgets so this class adapts *CompoundSelectionBehavior* based selection to work with *RecycleLayoutManagerBehavior* as well.

Similarly to *CompoundSelectionBehavior*, one can select using the keyboard or touch, which calls *select_node()* or *deselect_node()*, or one can call these methods directly. When a item is selected or deselected *apply_selection()* is called. See *apply_selection()*.

apply_selection(index, view, is_selected)

Applies the selection to the view. This is called internally when a view is displayed and it needs to be shown as selected or as not selected.

It is called when *select_node()* or *deselect_node()* is called or when a view needs to be refreshed. Its function is purely to update the view to reflect the selection state. So the function may be called multiple times even if the selection state may not have changed.

If the view is a instance of *RecycleDataViewBehavior*, its *apply_selection()* method will be called every time the view needs to refresh the selection state. Otherwise, the this method is responsible for applying the selection.

Parameters

index: int The index of the data item that is associated with the view.

view: widget The widget that is the view of this data item.

is_selected: **bool** Whether the item is selected.

deselect_node(*node*)

Deselects a possibly selected node.

It is called by the controller when it deselects a node and can also be called from the outside to deselect a node directly. The derived widget should overwrite this method and change the node to its unselected state when this is called

Parameters

node The node to be deselected.

Warning: This method must be called by the derived widget using `super` if it is overwritten.

get_index_of_node(*node*, *selectable_nodes*)

(internal) Returns the index of the *node* within the *selectable_nodes* returned by `get_selectable_nodes()`.

get_selectable_nodes()

(internal) Returns a list of the nodes that can be selected. It can be overwritten by the derived widget to return the correct list.

This list is used to determine which nodes to select with group selection. E.g. the last element in the list will be selected when home is pressed, pagedown will move (or add to, if shift is held) the selection from the current position by negative **page_count** nodes starting from the position of the currently selected node in this list and so on. Still, nodes can be selected even if they are not in this list.

Note: It is safe to dynamically change this list including removing, adding, or re-arranging its elements. Nodes can be selected even if they are not on this list. And selected nodes removed from the list will remain selected until `deselect_node()` is called.

Warning: Layouts display their children in the reverse order. That is, the contents of `children` is displayed from right to left, bottom to top. Therefore, internally, the indices of the elements returned by this function are reversed to make it work by default for most layouts so that the final result is consistent e.g. home, although it will select the last element in this list visually, will select the first element when counting from top to bottom and left to right. If this behavior is not desired, a reversed list should be returned instead.

Defaults to returning `children`.

goto_node(*key*, *last_node*, *last_node_idx*)

(internal) Used by the controller to get the node at the position indicated by *key*. The *key* can be keyboard inputs, e.g. pageup, or scroll inputs from the mouse scroll wheel, e.g. scrollup. 'last_node' is the last node selected and is used to find the resulting node. For example, if the *key* is up, the returned node is one node up from the last node.

It can be overwritten by the derived widget.

Parameters

key *str*, the string used to find the desired node. It can be any of the keyboard keys, as well as the mouse scrollup, scrolldown, scrollright, and scrollleft

strings. If letters are typed in quick succession, the letters will be combined before it's passed in as key and can be used to find nodes that have an associated string that starts with those letters.

last_node The last node that was selected.

last_node_idx The cached index of the last node selected in the ***get_selectable_nodes()*** list. If the list hasn't changed it saves having to look up the index of ***last_node*** in that list.

Returns tuple, the node targeted by key and its index in the ***get_selectable_nodes()*** list. Returning (***last_node***, ***last_node_idx***) indicates a node wasn't found.

key_selection

The key used to check whether a view of a data item can be selected with touch or the keyboard.

key_selection is the key in data, which if present and True will enable selection for this item from the keyboard or with a touch. When None, the default, not item will be selectable.

key_selection is a StringProperty and defaults to None.

Note: All data items can be selected directly using ***select_node()*** or ***deselect_node()***, even if ***key_selection*** is False.

select_node(*node*)

Selects a node.

It is called by the controller when it selects a node and can be called from the outside to select a node directly. The derived widget should overwrite this method and change the node state to selected when called.

Parameters

node The node to be selected.

Returns bool, True if the node was selected, False otherwise.

Warning: This method must be called by the derived widget using super if it is overwritten.

class kivy.uix.recycleview.layout.RecycleLayoutManagerBehavior

Bases: builtins.object

A RecycleLayoutManagerBehavior is responsible for positioning views into the RecycleView. data within a RecycleView. It adds new views into the data when it becomes visible to the user, and removes them when they leave the visible area.

compute_visible_views(*data*, *viewport*)

viewport is in coordinates of the layout manager.

get_view_index_at(*pos*)

Return the view *index* on which position, *pos*, falls.

pos is in coordinates of the layout manager.

goto_view(*index*)

Moves the views so that the view corresponding to *index* is visible.

key_viewclass

See RecycleView.key_viewclass.

refresh_view_layout(*index, layout, view, viewport*)

See :meth:`~kivy.uix.recycleview.views.RecycleDataAdapter.refresh_view_layout`.

set_visible_views(*indices, data, viewport*)

viewport is in coordinates of the layout manager.

viewclass

See `RecyclerView.viewclass`.

33.2.3 RecycleView Views

New in version 1.10.0.

The adapter part of the RecycleView which together with the layout is the view part of the model-view-controller pattern.

The view module handles converting the data to a view using the adapter class which is then displayed by the layout. A view can be any Widget based class. However, inheriting from `RecycleDataViewBehavior` adds methods for converting the data to a view.

TODO:

- Make view caches specific to each view class type.

class `kivy.uix.recycleview.views.RecycleDataAdapter`(***kwargs*)

Bases: `kivy.event.EventDispatcher`

The class that converts data to a view.

— Internal details — A view can have 3 states.

- It can be completely in sync with the data, which occurs when the view is displayed. These are stored in `views`.
- It can be dirty, which occurs when the view is in sync with the data, except for the size/pos parameters which is controlled by the layout. This occurs when the view is not currently displayed but the data has not changed. These views are stored in `dirty_views`.
- Finally the view can be dead which occurs when the data changes and the view was not updated or when a view is just created. Such views are typically added to the internal cache.

Typically what happens is that the layout manager lays out the data and then asks for views, using `set_visible_views()` for some specific data items that it displays.

These views are gotten from the current views, dirty or global cache. Then depending on the view state `refresh_view_attrs()` is called to bring the view up to date with the data (except for sizing parameters). Finally, the layout manager gets these views, updates their size and displays them.

attach_recycleview(*rv*)

Associates a `RecycleViewBehavior` with this instance. It is stored in `recycleview`.

create_view(*index, data_item, viewclass*)

(internal) Creates and initializes the view for the data at *index*.

The returned view is synced with the data, except for the pos/size information.

detach_recycleview()

Removes the `RecycleViewBehavior` associated with this instance and clears `recycleview`.

get_view(*index, data_item, viewclass*)

(internal) Returns a view instance for the data at *index*

It looks through the various caches and finally creates a view if it doesn't exist. The returned view is synced with the data, except for the pos/size information.

If found in the cache it's removed from the source before returning. It doesn't check the current views.

get_visible_view(*index*)

Returns the currently visible view associated with *index*.

If no view is currently displayed for *index* it returns `None`.

invalidate()

Moves all the current views into the global cache.

As opposed to making a view dirty where the view is in sync with the data except for sizing information, this will completely disconnect the view from the data, as it is assumed the data has gone out of sync with the view.

This is typically called when the data changes.

make_view_dirty(*view*, *index*)

(internal) Used to flag this view as dirty, ready to be used for others. See [*make_views_dirty\(\)*](#).

make_views_dirty()

Makes all the current views dirty.

Dirty views are still in sync with the corresponding data. However, the size information may go out of sync. Therefore a dirty view can be reused by the same index by just updating the sizing information.

Once the underlying data of this index changes, the view should be removed from the dirty views and moved to the global cache with [*invalidate\(\)*](#).

This is typically called when the layout manager needs to re-layout all the data.

recycleview

The [*RecycleViewBehavior*](#) associated with this instance.

refresh_view_attrs(*index*, *data_item*, *view*)

(internal) Syncs the view and brings it up to date with the data.

This method calls [*RecycleDataViewBehavior.refresh_view_attrs\(\)*](#) if the view inherits from [*RecycleDataViewBehavior*](#). See that method for more details.

Note: Any sizing and position info is skipped when syncing with the data.

refresh_view_layout(*index*, *layout*, *view*, *viewport*)

Updates the sizing information of the view.

viewport is in coordinates of the layout manager.

This method calls [*RecycleDataViewBehavior.refresh_view_attrs\(\)*](#) if the view inherits from [*RecycleDataViewBehavior*](#). See that method for more details.

Note: Any sizing and position info is skipped when syncing with the data.

set_visible_views(*indices*, *data*, *viewclasses*)

Gets a 3-tuple of the new, remaining, and old views for the current viewport.

The new views are synced to the data except for the size/pos properties. The old views need to be removed from the layout, and the new views added.

The new views are not necessarily *new*, but are all the currently visible views.

class `kivy.uix.recycleview.views.RecycleDataViewBehavior`

Bases: `builtins.object`

An optional base class for data views (`RecycleView.viewclass`). If a view inherits from this class, the class's functions will be called when the view needs to be updated due to a data change or layout update.

refresh_view_attrs(*rv, index, data*)

Called by the `RecycleAdapter` when the view is initially populated with the values from the *data* dictionary for this item.

Any pos or size info should be removed because they are set subsequently with *refresh_view_layout*.

Parameters

rv: RecycleView instance The `RecycleView` that caused the update.

data: dict The data dict used to populate this view.

refresh_view_layout(*rv, index, layout, viewport*)

Called when the view's size is updated by the layout manager, `RecycleLayoutManagerBehavior`.

Parameters

rv: RecycleView instance The `RecycleView` that caused the update.

viewport: 4-tuple The coordinates of the bottom left and width height in layout manager coordinates. This may be larger than this view item.

Raises *LayoutChangeException*: If the sizing or data changed during a call to this method, raising a *LayoutChangeException* exception will force a refresh. Useful when data changed and we don't want to layout further since it'll be overwritten again soon.

33.3 Accordion

New in version 1.0.8.



The Accordion widget is a form of menu where the options are stacked either vertically or horizontally and the item in focus (when touched) opens up to display its content.

The *Accordion* should contain one or many *AccordionItem* instances, each of which should contain one root content widget. You'll end up with a Tree something like this:

- Accordion
 - AccordionItem

- * YourContent
- AccordionItem
 - * BoxLayout
 - Another user content 1
 - Another user content 2
- AccordionItem
 - * Another user content

The current implementation divides the *AccordionItem* into two parts:

1. One container for the title bar
2. One container for the content

The title bar is made from a Kv template. We'll see how to create a new template to customize the design of the title bar.

Warning: If you see message like:

```
[WARNING] [Accordion] not have enough space for displaying all children
[WARNING] [Accordion] need 440px, got 100px
[WARNING] [Accordion] layout aborted.
```

That means you have too many children and there is no more space to display the content. This is “normal” and nothing will be done. Try to increase the space for the accordion or reduce the number of children. You can also reduce the *Accordion.min_space*.

33.3.1 Simple example

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
    def build(self):
        root = Accordion()
        for x in range(5):
            item = AccordionItem(title='Title %d' % x)
            item.add_widget(Label(text='Very big content\n' * 10))
            root.add_widget(item)
        return root

if __name__ == '__main__':
    AccordionApp().run()
```

33.3.2 Customize the accordion

You can increase the default size of the title bar:

```
root = Accordion(min_space=60)
```

Or change the orientation to vertical:


```
root = Accordion(orientation='vertical')
```

The `AccordionItem` is more configurable and you can set your own title background when the item is collapsed or opened:

```
item = AccordionItem(background_normal='image_when_collapsed.png',  
    background_selected='image_when_selected.png')
```

class `kivy.uix.accordion.Accordion`(**kwargs)

Bases: `kivy.uix.widget.Widget`

Accordion class. See module documentation for more information.

add_widget(widget, *largs)

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)
```

anim_duration

Duration of the animation in seconds when a new accordion item is selected.

anim_duration is a *NumericProperty* and defaults to .25 (250ms).

anim_func

Easing function to use for the animation. Check *kivy.animation.AnimationTransition* for more information about available animation functions.

anim_func is an *ObjectProperty* and defaults to 'out_expo'. You can set a string or a function to use as an easing function.

min_space

Minimum space to use for the title of each item. This value is automatically set for each child every time the layout event occurs.

min_space is a *NumericProperty* and defaults to 44 (px).

orientation

Orientation of the layout.

orientation is an *OptionProperty* and defaults to 'horizontal'. Can take a value of 'vertical' or 'horizontal'.

class kivy.uix.accordion.**AccordionItem**(**kwargs)

Bases: *kivy.uix.floatlayout.FloatLayout*

AccordionItem class that must be used in conjunction with the *Accordion* class. See the module documentation for more information.

accordion

Instance of the *Accordion* that the item belongs to.

accordion is an *ObjectProperty* and defaults to None.

add_widget(widget)

Add a new widget as a child of this widget.

Parameters

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

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

New in version 1.0.5.

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

New in version 1.9.0.

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

background_disabled_normal

Background image of the accordion item used for the default graphical representation when the item is collapsed and disabled.

New in version 1.8.0.

background__disabled_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button_disabled'.

background_disabled_selected

Background image of the accordion item used for the default graphical representation when the item is selected (not collapsed) and disabled.

New in version 1.8.0.

background_disabled_selected is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button_disabled_pressed'.

background_normal

Background image of the accordion item used for the default graphical representation when the item is collapsed.

background_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button'.

background_selected

Background image of the accordion item used for the default graphical representation when the item is selected (not collapsed).

`background_normal` is a `StringProperty` and defaults to `'atlas://data/images/defaulttheme/button_pressed'`.

collapse

Boolean to indicate if the current item is collapsed or not.

`collapse` is a `BooleanProperty` and defaults to `True`.

collapse_alpha

Value between 0 and 1 to indicate how much the item is collapsed (1) or whether it is selected (0). It's mostly used for animation.

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

container

(internal) Property that will be set to the container of children inside the `AccordionItem` representation.

container_title

(internal) Property that will be set to the container of title inside the `AccordionItem` representation.

content_size

(internal) Set by the `Accordion` to the size allocated for the content.

min_space

Link to the `Accordion.min_space` property.

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.

orientation

Link to the `Accordion.orientation` property.

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)
```

title

Title string of the item. The title might be used in conjunction with the `AccordionItemTitle` template. If you are using a custom template, you can use that property as a text entry, or not. By default, it's used for the title text. See `title_template` and the example below.

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

title_args

Default arguments that will be passed to the `kivy.lang.Builder.template()` method.

`title_args` is a `DictProperty` and defaults to `{}`.

title_template

Template to use for creating the title part of the accordion item. The default template is a simple Label, not customizable (except the text) that supports vertical and horizontal orientation and different backgrounds for collapse and selected mode.

It's better to create and use your own template if the default template does not suffice.

title is a *StringProperty* and defaults to 'AccordionItemTitle'. The current default template lives in the *kivy/data/style.kv* file.

Here is the code if you want to build your own template:

```
[AccordionItemTitle@Label]:
    text: ctx.title
    canvas.before:
        Color:
            rgb: 1, 1, 1
        BorderImage:
            source:
                ctx.item.background_normal
            if ctx.
→item.collapse
                else ctx.item.background_selected
        pos: self.pos
        size: self.size
    PushMatrix
    Translate:
        xy: self.center_x, self.center_y
    Rotate:
        angle: 90 if ctx.item.orientation == 'horizontal' else 0
        axis: 0, 0, 1
    Translate:
        xy: -self.center_x, -self.center_y
    canvas.after:
        PopMatrix
```

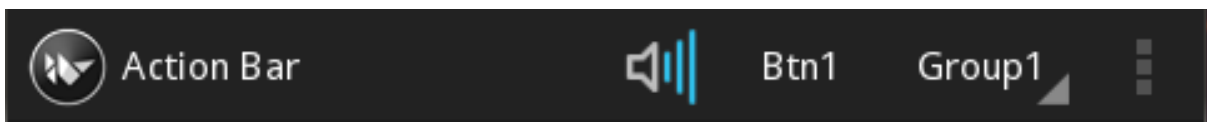
exception kivy.uix.accordion.AccordionException

Bases: Exception

AccordionException class.

33.4 Action Bar

New in version 1.8.0.



The ActionBar widget is like Android's *ActionBar*, where items are stacked horizontally. When the area becomes too small, widgets are moved into the *ActionOverflow* area.

An *ActionBar* contains an *ActionView* with various *ContextualActionViews*. An *ActionView* will contain an *ActionPrevious* having title, app_icon and previous_icon properties. An *ActionView* will contain subclasses of *ActionItems*. Some predefined ones include an *ActionButton*, an *ActionToggleButton*, an *ActionCheck*, an *ActionSeparator* and an *ActionGroup*.

An *ActionGroup* is used to display *ActionItems* in a group. An *ActionView* will always display an *ActionGroup* after other *ActionItems*. An *ActionView* contains an *ActionOverflow*, but

this is only made visible when required i.e. the available area is too small to fit all the widgets. A *ContextualActionView* is a subclass of `android.support.design.widget.ActionView`.

Changed in version 1.10.1: *ActionGroup* core rewritten from *Spinner* to pure *DropDown*

exception `kivy.uix.actionbar.ActionBarException`

Bases: `Exception`

ActionBarException class

class `kivy.uix.actionbar.ActionItem`

Bases: `builtins.object`

ActionItem class, an abstract class for all *ActionBar* widgets. To create a custom widget for an *ActionBar*, inherit from this class. See module documentation for more information.

background_down

Background image of the *ActionItem* used for the default graphical representation when an *ActionItem* is pressed.

background_down is a *StringProperty* and defaults to `'atlas://data/images/defaulttheme/action_item_down'`.

background_normal

Background image of the *ActionItem* used for the default graphical representation when the *ActionItem* is not pressed.

background_normal is a *StringProperty* and defaults to `'atlas://data/images/defaulttheme/action_item'`.

important

Determines if an *ActionItem* is important or not. If an item is important and space is limited, this item will be displayed in preference to others.

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

inside_group

(internal) Determines if an *ActionItem* is displayed inside an *ActionGroup* or not.

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

minimum_width

Minimum Width required by an *ActionItem*.

minimum_width is a *NumericProperty* and defaults to `'90sp'`.

mipmap

Defines whether the image/icon displayed on top of the button uses a mipmap or not.

mipmap is a *BooleanProperty* and defaults to `True`.

pack_width

(read-only) The actual width to use when packing the items. Equal to the greater of `minimum_width` and `width`.

pack_width is an *AliasProperty*.

class `kivy.uix.actionbar.ActionButton(**kwargs)`

Bases: `kivy.uix.button.Button`, `kivy.uix.actionbar.ActionItem`

ActionButton class, see module documentation for more information.

The text color, width and `size_hint_x` are set manually via the Kv language file. It covers a lot of cases: with/without an icon, with/without a group and takes care of the padding between elements.

You don't have much control over these properties, so if you want to customize its appearance, we suggest you create your own button representation. You can do this by creating a class that subclasses an existing widget and an *ActionItem*:

```
class MyOwnActionButton(Button, ActionItem):  
    pass
```

You can then create your own style using the Kv language.

icon

Source image to use when the Button is part of the ActionBar. If the Button is in a group, the text will be preferred.

icon is a *StringProperty* and defaults to None.

class `kivy.uix.actionbar.ActionToggleButton(**kwargs)`

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.togglebutton.ToggleButton*

ActionToggleButton class, see module documentation for more information.

icon

Source image to use when the Button is part of the ActionBar. If the Button is in a group, the text will be preferred.

class `kivy.uix.actionbar.ActionCheck(**kwargs)`

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.checkbox.CheckBox*

ActionCheck class, see module documentation for more information.

class `kivy.uix.actionbar.ActionSeparator(**kwargs)`

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.widget.Widget*

ActionSeparator class, see module documentation for more information.

background_image

Background image for the separators default graphical representation.

background_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/separator'.

class `kivy.uix.actionbar.ActionDropDown(**kwargs)`

Bases: *kivy.uix.dropdown.DropDown*

ActionDropDown class, see module documentation for more information.

class `kivy.uix.actionbar.ActionGroup(**kwargs)`

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.button.Button*

ActionGroup class, see module documentation for more information.

add_widget(item)

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)
```

clear_widgets()

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

dropdown_width

If non zero, provides the width for the associated DropDown. This is useful when some items in the ActionGroup's DropDown are wider than usual and you don't want to make the ActionGroup widget itself wider.

dropdown_width is a *NumericProperty* and defaults to 0.

New in version 1.10.0.

is_open

By default, the DropDown is not open. Set to True to open it.

is_open is a *BooleanProperty* and defaults to False.

mode

Sets the current mode of an ActionGroup. If mode is 'normal', the ActionGroups children will be displayed normally if there is enough space, otherwise they will be displayed in a spinner. If mode is 'spinner', then the children will always be displayed in a spinner.

mode is an *OptionProperty* and defaults to 'normal'.

separator_image

Background Image for an ActionSeparator in an ActionView.

separator_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/separator'.

separator_width

Width of the ActionSeparator in an ActionView.

separator_width is a *NumericProperty* and defaults to 0.

use_separator

Specifies whether to use a separator after/before this group or not.

use_separator is a *BooleanProperty* and defaults to False.

class kivy.uix.actionbar.ActionOverflow(**kwargs)

Bases: *kivy.uix.actionbar.ActionGroup*

ActionOverflow class, see module documentation for more information.

add_widget(action_item, 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)
```

overflow_image

Image to be used as an Overflow Image.

overflow_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/overflow'.

class kivy.uix.actionbar.ActionView(**kwargs)

Bases: *kivy.uix.boxlayout.BoxLayout*

ActionView class, see module documentation for more information.

action_previous

Previous button for an ActionView.

action_previous is an *ObjectProperty* and defaults to None.

add_widget(*action_item*, *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)
```

background_color

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

background_color is a *ListProperty* and defaults to [1, 1, 1, 1].

background_image

Background image of an ActionViews default graphical representation.

background_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/action_view'.

overflow_group

Widget to be used for the overflow.

overflow_group is an *ObjectProperty* and defaults to an instance of *ActionOverflow*.

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)
```

use_separator

Specify whether to use a separator before every ActionGroup or not.

use_separator is a *BooleanProperty* and defaults to False.

class kivy.uix.actionbar.ContextualActionView(kwargs)**

Bases: *kivy.uix.actionbar.ActionView*

ContextualActionView class, see the module documentation for more information.

class kivy.uix.actionbar.ActionPrevious(kwargs)**

Bases: *kivy.uix.boxlayout.BoxLayout*, *kivy.uix.actionbar.ActionItem*

ActionPrevious class, see module documentation for more information.

app_icon

Application icon for the ActionView.

app_icon is a *StringProperty* and defaults to the window icon if set, otherwise 'data/logo/kivy-icon-32.png'.

app_icon_height

Height of app_icon image.

app_icon_height is a *NumericProperty* and defaults to 0.

app_icon_width

Width of app_icon image.

app_icon_width is a *NumericProperty* and defaults to 0.

color

Text color, in the format (r, g, b, a)

color is a *ListProperty* and defaults to [1, 1, 1, 1].

markup

If True, the text will be rendered using the *MarkupLabel*: you can change the style of the text using tags. Check the *Text Markup* documentation for more information.

markup is a *BooleanProperty* and defaults to False.

previous_image

Image for the 'previous' ActionButtons default graphical representation.

previous_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/previous_normal'.

previous_image_height

Height of previous_image image.

app_icon_width is a *NumericProperty* and defaults to 0.

previous_image_width

Width of previous_image image.

width is a *NumericProperty* and defaults to 0.

title

Title for ActionView.

title is a *StringProperty* and defaults to "".

with_previous

Specifies whether the previous_icon will be shown or not. Note that it is up to the user to implement the desired behavior using the *on_press* or similar events.

with_previous is a *BooleanProperty* and defaults to True.

class `kivy.uix.actionbar.ActionBar(**kwargs)`

Bases: *kivy.uix.boxlayout.BoxLayout*

ActionBar class, which acts as the main container for an *ActionView* instance. The ActionBar determines the overall styling aspects of the bar. *ActionItems* are not added to this class directly, but to the contained *ActionView* instance.

Events

on_previous Fired when action_previous of action_view is pressed.

Please see the module documentation for more information.

action_view

action_view of the ActionBar.

action_view is an *ObjectProperty* and defaults to None or the last ActionView instance added to the ActionBar.

add_widget(view)

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)
```

background_color

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

background_color is a *ListProperty* and defaults to [1, 1, 1, 1].

background_image

Background image of the ActionBar's default graphical representation.

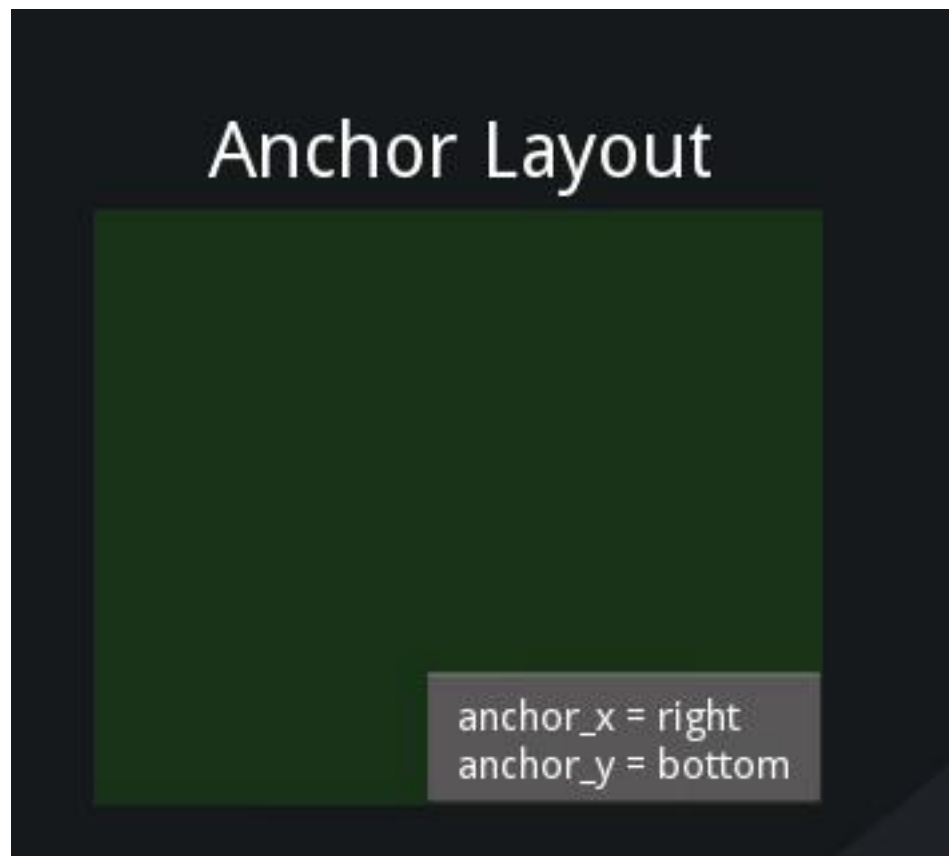
background_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/action_bar'.

border

The border to be applied to the *background_image*.

border is a *ListProperty* and defaults to [2, 2, 2, 2]

33.5 Anchor Layout



The *AnchorLayout* aligns its children to a border (top, bottom, left, right) or center.

To draw a button in the lower-right corner:

```
layout = AnchorLayout(
    anchor_x='right', anchor_y='bottom')
btn = Button(text='Hello World')
layout.add_widget(btn)
```

class `kivy.uix.anchorlayout.AnchorLayout`(**kwargs)

Bases: `kivy.uix.layout.Layout`

Anchor layout class. See the module documentation for more information.

anchor_x

Horizontal anchor.

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

anchor_y

Vertical anchor.

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

do_layout(*args)

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.

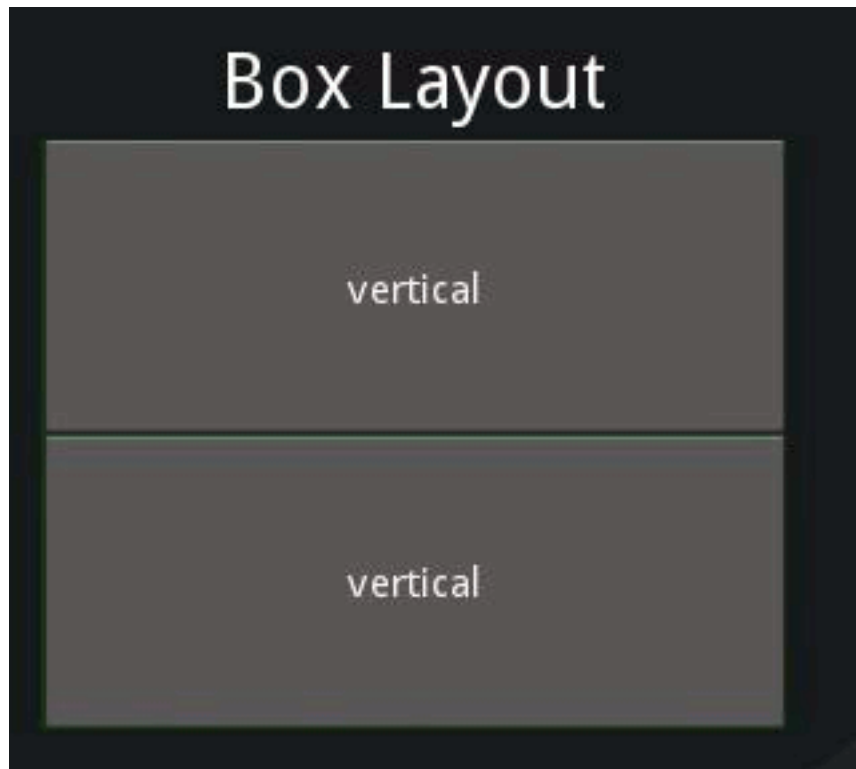
padding

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

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

padding is a *VariableListProperty* and defaults to [0, 0, 0, 0].

33.6 Box Layout



BoxLayout arranges children in a vertical or horizontal box.

To position widgets above/below each other, use a vertical *BoxLayout*:

```
layout = BoxLayout(orientation='vertical')
btn1 = Button(text='Hello')
btn2 = Button(text='World')
layout.add_widget(btn1)
layout.add_widget(btn2)
```

To position widgets next to each other, use a horizontal *BoxLayout*. In this example, we use 10 pixel spacing between children; the first button covers 70% of the horizontal space, the second covers 30%:

```
layout = BoxLayout(spacing=10)
btn1 = Button(text='Hello', size_hint=(.7, 1))
btn2 = Button(text='World', size_hint=(.3, 1))
layout.add_widget(btn1)
layout.add_widget(btn2)
```

Position hints are partially working, depending on the orientation:

- If the orientation is *vertical*: *x*, *right* and *center_x* will be used.
- If the orientation is *horizontal*: *y*, *top* and *center_y* will be used.

Kv Example:

```
BoxLayout:
    orientation: 'vertical'
    Label:
        text: 'this on top'
    Label:
```

(continues on next page)

```

    text: 'this right aligned'
    size_hint_x: None
    size: self.texture_size
    pos_hint: {'right': 1}
Label:
    text: 'this on bottom'

```

You can check the *examples/widgets/boxlayout_poshint.py* for a live example.

Note: The *size_hint* uses the available space after subtracting all the fixed-size widgets. For example, if you have a layout that is 800px wide, and add three buttons like this:

```

btn1 = Button(text='Hello', size=(200, 100), size_hint=(None, None))
btn2 = Button(text='Kivy', size_hint=(.5, 1))
btn3 = Button(text='World', size_hint=(.5, 1))

```

The first button will be 200px wide as specified, the second and third will be 300px each, e.g. $(800-200) * 0.5$

Changed in version 1.4.1: Added support for *pos_hint*.

class kivy.uix.boxlayout.BoxLayout(**kwargs)

Bases: *kivy.uix.layout.Layout*

Box layout class. See module documentation for more information.

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)

```

do_layout(*larges)

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.

minimum_height

Automatically computed minimum height needed to contain all children.

New in version 1.10.0.

minimum_height is a *NumericProperty* and defaults to 0. It is read only.

minimum_size

Automatically computed minimum size needed to contain all children.

New in version 1.10.0.

minimum_size is a *ReferenceListProperty* of (*minimum_width*, *minimum_height*) properties. It is read only.

minimum_width

Automatically computed minimum width needed to contain all children.

New in version 1.10.0.

minimum_width is a *NumericProperty* and defaults to 0. It is read only.

orientation

Orientation of the layout.

orientation is an *OptionProperty* and defaults to 'horizontal'. Can be 'vertical' or 'horizontal'.

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 1.7.0: Replaced *NumericProperty* with *VariableListProperty*.

padding is a *VariableListProperty* and defaults to [0, 0, 0, 0].

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)
```

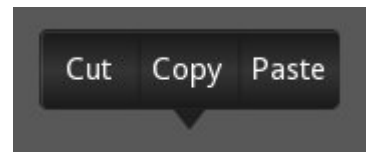
spacing

Spacing between children, in pixels.

spacing is a *NumericProperty* and defaults to 0.

33.7 Bubble

New in version 1.1.0.



The Bubble widget is a form of menu or a small popup where the menu options are stacked either vertically or horizontally.

The *Bubble* contains an arrow pointing in the direction you choose.

33.7.1 Simple example

```
'''
Bubble
=====

Test of the widget Bubble.
'''

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.button import Button
from kivy.lang import Builder
from kivy.uix.bubble import Bubble

Builder.load_string('''
<cut_copy_paste>
    size_hint: (None, None)
    size: (160, 120)
    pos_hint: {'center_x': .5, 'y': .6}
    BubbleButton:
        text: 'Cut'
    BubbleButton:
        text: 'Copy'
    BubbleButton:
        text: 'Paste'
''')

class cut_copy_paste(Bubble):
    pass

class BubbleShowcase(FloatLayout):

    def __init__(self, **kwargs):
        super(BubbleShowcase, self).__init__(**kwargs)
        self.but_bubble = Button(text='Press to show bubble')
        self.but_bubble.bind(on_release=self.show_bubble)
        self.add_widget(self.but_bubble)

    def show_bubble(self, *l):
        if not hasattr(self, 'bubb'):
            self.bubb = bubb = cut_copy_paste()
            self.add_widget(bubb)
        else:
            values = ('left_top', 'left_mid', 'left_bottom', 'top_left',
                    'top_mid', 'top_right', 'right_top', 'right_mid',
```

(continues on next page)

(continued from previous page)

```
        'right_bottom', 'bottom_left', 'bottom_mid', 'bottom_right')
    index = values.index(self.bubb.arrow_pos)
    self.bubb.arrow_pos = values[(index + 1) % len(values)]

class TestBubbleApp(App):

    def build(self):
        return BubbleShowcase()

if __name__ == '__main__':
    TestBubbleApp().run()
```

33.7.2 Customize the Bubble

You can choose the direction in which the arrow points:

```
Bubble(arrow_pos='top_mid')
```

The widgets added to the Bubble are ordered horizontally by default, like a Boxlayout. You can change that by:

```
orientation = 'vertical'
```

To add items to the bubble:

```
bubble = Bubble(orientation = 'vertical')
bubble.add_widget(your_widget_instance)
```

To remove items:

```
bubble.remove_widget(widget)
or
bubble.clear_widgets()
```

To access the list of children, use `content.children`:

```
bubble.content.children
```

Warning: This is important! Do not use `bubble.children`

To change the appearance of the bubble:

```
bubble.background_color = (1, 0, 0, .5) #50% translucent red
bubble.border = [0, 0, 0, 0]
background_image = 'path/to/background/image'
arrow_image = 'path/to/arrow/image'
```

```
class kivy.uix.bubble.Bubble(**kwargs)
    Bases: kivy.uix.gridlayout.GridLayout
    Bubble class. See module documentation for more information.
    add_widget(*l)
        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)
```

arrow_image

Image of the arrow pointing to the bubble.

arrow_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/bubble_arrow'.

arrow_pos

Specifies the position of the arrow relative to the bubble. Can be one of: left_top, left_mid, left_bottom, top_left, top_mid, top_right, right_top, right_mid, right_bottom, bottom_left, bottom_mid, bottom_right.

arrow_pos is a *OptionProperty* and defaults to 'bottom_mid'.

background_color

Background color, in the format (r, g, b, a). To use it you have to set either *background_image* or *arrow_image* first.

background_color is a *ListProperty* and defaults to [1, 1, 1, 1].

background_image

Background image of the bubble.

background_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/bubble'.

border

Border used for *BorderImage* graphics instruction. Used with the *background_image*. It should be used when using custom backgrounds.

It must be a list of 4 values: (bottom, right, top, left). Read the *BorderImage* instructions for more information about how to use it.

border is a *ListProperty* and defaults to (16, 16, 16, 16)

border_auto_scale

Specifies the *kivy.graphics.BorderImage.auto_scale* value on the background *BorderImage*.

New in version 1.11.0.

border_auto_scale is a *OptionProperty* and defaults to 'both_lower'.

clear_widgets(kwargs)**

Remove all (or the specified) `children` of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

content

This is the object where the main content of the bubble is held.

content is a *ObjectProperty* and defaults to 'None'.

limit_to

Specifies the widget to which the bubbles position is restricted.

New in version 1.6.0.

limit_to is a *ObjectProperty* and defaults to 'None'.

orientation

This specifies the manner in which the children inside bubble are arranged. Can be one of 'vertical' or 'horizontal'.

orientation is a *OptionProperty* and defaults to 'horizontal'.

remove_widget(*l)

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)
```

show_arrow

Indicates whether to show arrow.

New in version 1.8.0.

show_arrow is a *BooleanProperty* and defaults to *True*.

class kivy.uix.bubble.BubbleButton(kwargs)**

Bases: *kivy.uix.button.Button*

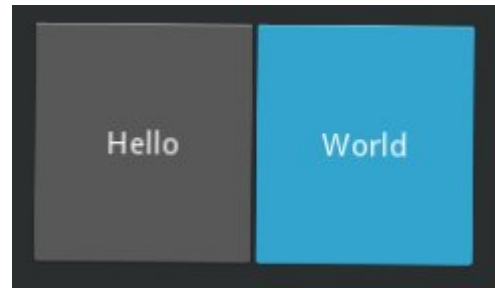
A button intended for use in a Bubble widget. You can use a "normal" button class, but it will not look good unless the background is changed.

Rather use this BubbleButton widget that is already defined and provides a suitable background for you.

class kivy.uix.bubble.BubbleContent(kwargs)**

Bases: *kivy.uix.gridlayout.GridLayout*

33.8 Button



The *Button* is a *Label* with associated actions that are triggered when the button is pressed (or released after a click/touch). To configure the button, the same properties (padding, font_size, etc) and *sizing system* are used as for the *Label* class:

```
button = Button(text='Hello world', font_size=14)
```

To attach a callback when the button is pressed (clicked/touched), use `bind`:

```
def callback(instance):
    print('The button <%s> is being pressed' % instance.text)

btn1 = Button(text='Hello world 1')
btn1.bind(on_press=callback)
btn2 = Button(text='Hello world 2')
btn2.bind(on_press=callback)
```

If you want to be notified every time the button state changes, you can bind to the `Button.state` property:

```
def callback(instance, value):
    print('My button <%s> state is <%s>' % (instance, value))
btn1 = Button(text='Hello world 1')
btn1.bind(state=callback)
```

Kv Example:

```
Button:
    text: 'press me'
    on_press: print("ouch! More gently please")
    on_release: print("ahhh")
    on_state:
        print("my current state is {}".format(self.state))
```

class `kivy.uix.button.Button`(**kwargs)

Bases: `kivy.uix.behaviors.button.ButtonBehavior`, `kivy.uix.label.Label`

Button class, see module documentation for more information.

Changed in version 1.8.0: The behavior / logic of the button has been moved to `ButtonBehaviors`.

background_color

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

This acts as a *multiplier* to the texture colour. The default texture is grey, so just setting the background color will give a darker result. To set a plain color, set the *background_normal* to `''`.

New in version 1.0.8.

The *background_color* is a *ListProperty* and defaults to [1, 1, 1, 1].

background_disabled_down

Background image of the button used for the default graphical representation when the button is disabled and pressed.

New in version 1.8.0.

background_disabled_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button_disabled_pressed'.

background_disabled_normal

Background image of the button used for the default graphical representation when the button is disabled and not pressed.

New in version 1.8.0.

background_disabled_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button_disabled'.

background_down

Background image of the button used for the default graphical representation when the button is pressed.

New in version 1.0.4.

background_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button_pressed'.

background_normal

Background image of the button used for the default graphical representation when the button is not pressed.

New in version 1.0.4.

background_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button'.

border

Border used for *BorderImage* graphics instruction. Used with *background_normal* and *background_down*. Can be used for custom backgrounds.

It must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instruction for more information about how to use it.

border is a *ListProperty* and defaults to (16, 16, 16, 16)

33.9 Camera

The *Camera* widget is used to capture and display video from a camera. Once the widget is created, the texture inside the widget will be automatically updated. Our *CameraBase* implementation is used under the hood:

```
cam = Camera()
```

By default, the first camera found on your system is used. To use a different camera, set the index property:

```
cam = Camera(index=1)
```

You can also select the camera resolution:

```
cam = Camera(resolution=(320, 240))
```

Warning: The camera texture is not updated as soon as you have created the object. The camera initialization is asynchronous, so there may be a delay before the requested texture is created.

class kivy.uix.camera.Camera(**kwargs)

Bases: *kivy.uix.image.Image*

Camera class. See module documentation for more information.

index

Index of the used camera, starting from 0.

index is a *NumericProperty* and defaults to -1 to allow auto selection.

play

Boolean indicating whether the camera is playing or not. You can start/stop the camera by setting this property:

```
# start the camera playing at creation (default)
cam = Camera(play=True)

# create the camera, and start later
cam = Camera(play=False)
# and later
cam.play = True
```

play is a *BooleanProperty* and defaults to True.

resolution

Preferred resolution to use when invoking the camera. If you are using [-1, -1], the resolution will be the default one:

```
# create a camera object with the best image available
cam = Camera()

# create a camera object with an image of 320x240 if possible
cam = Camera(resolution=(320, 240))
```

Warning: Depending on the implementation, the camera may not respect this property.

resolution is a *ListProperty* and defaults to [-1, -1].

33.10 Carousel

New in version 1.4.0.

The *Carousel* widget provides the classic mobile-friendly carousel view where you can swipe between slides. You can add any content to the carousel and have it move horizontally or vertically. The carousel can display pages in a sequence or a loop.

Example:

```

from kivy.app import App
from kivy.uix.carousel import Carousel
from kivy.uix.image import AsyncImage

class CarouselApp(App):
    def build(self):
        carousel = Carousel(direction='right')
        for i in range(10):
            src = "http://placeholder.it/480x270.png&text=slide-%d&.png" % i
            image = AsyncImage(source=src, allow_stretch=True)
            carousel.add_widget(image)
        return carousel

CarouselApp().run()

```

Kv Example:

```

Carousel:
    direction: 'right'
    AsyncImage:
        source: 'http://placeholder.it/480x270.png&text=slide-1.png'
    AsyncImage:
        source: 'http://placeholder.it/480x270.png&text=slide-2.png'
    AsyncImage:
        source: 'http://placeholder.it/480x270.png&text=slide-3.png'
    AsyncImage:
        source: 'http://placeholder.it/480x270.png&text=slide-4.png'

```

Changed in version 1.5.0: The carousel now supports active children, like the *ScrollView*. It will detect a swipe gesture according to the *Carousel.scroll_timeout* and *Carousel.scroll_distance* properties.

In addition, the slide container is no longer exposed by the API. The impacted properties are *Carousel.slides*, *Carousel.current_slide*, *Carousel.previous_slide* and *Carousel.next_slide*.

class kivy.uix.carousel.**Carousel**(**kwargs)
 Bases: *kivy.uix.stencilview.StencilView*

Carousel class. See module documentation for more information.

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)
```

anim_cancel_duration

Defines the duration of the animation when a swipe movement is not accepted. This is generally when the user does not make a large enough swipe. See [min_move](#).

[anim_cancel_duration](#) is a [NumericProperty](#) and defaults to 0.3.

anim_move_duration

Defines the duration of the Carousel animation between pages.

[anim_move_duration](#) is a [NumericProperty](#) and defaults to 0.5.

anim_type

Type of animation to use while animating to the next/previous slide. This should be the name of an [AnimationTransition](#) function.

[anim_type](#) is a [StringProperty](#) and defaults to 'out_quad'.

New in version 1.8.0.

clear_widgets()

Remove all (or the specified) `children` of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

current_slide

The currently shown slide.

[current_slide](#) is an [AliasProperty](#).

Changed in version 1.5.0: The property no longer exposes the slides container. It returns the widget you have added.

direction

Specifies the direction in which the slides are ordered. This corresponds to the direction from which the user swipes to go from one slide to the next. It can be *right*, *left*, *top*, or *bottom*. For example, with the default value of *right*, the second slide is to the right of the first and the user would swipe from the right towards the left to get to the second slide.

[direction](#) is an [OptionProperty](#) and defaults to 'right'.

ignore_perpendicular_swipes

Ignore swipes on axis perpendicular to direction.

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

New in version 1.10.0.

index

Get/Set the current slide based on the index.

[index](#) is an [AliasProperty](#) and defaults to 0 (the first item).

load_next(mode='next')

Animate to the next slide.

New in version 1.7.0.

load_previous()

Animate to the previous slide.

New in version 1.7.0.

load_slide(*slide*)

Animate to the slide that is passed as the argument.

Changed in version 1.8.0.

loop

Allow the Carousel to loop infinitely. If True, when the user tries to swipe beyond last page, it will return to the first. If False, it will remain on the last page.

loop is a *BooleanProperty* and defaults to False.

min_move

Defines the minimum distance to be covered before the touch is considered a swipe gesture and the Carousel content changed. This is expressed as a fraction of the Carousel's width. If the movement doesn't reach this minimum value, the movement is cancelled and the content is restored to its original position.

min_move is a *NumericProperty* and defaults to 0.2.

next_slide

The next slide in the Carousel. It is None if the current slide is the last slide in the Carousel. This ordering reflects the order in which the slides are added: their presentation varies according to the *direction* property.

next_slide is an *AliasProperty*.

Changed in version 1.5.0: The property no longer exposes the slides container. It returns the widget you have added.

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.

previous_slide

The previous slide in the Carousel. It is None if the current slide is the first slide in the Carousel. This ordering reflects the order in which the slides are added: their presentation varies according to the *direction* property.

previous_slide is an *AliasProperty*.

Changed in version 1.5.0: This property no longer exposes the slides container. It returns the widget you have added.

remove_widget(*widget*, *args, **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)
```

scroll_distance

Distance to move before scrolling the *Carousel* in pixels. As soon as the distance has been traveled, the *Carousel* will start to scroll, and no touch event will go to children. It is advisable that you base this value on the dpi of your target device's screen.

scroll_distance is a *NumericProperty* and defaults to 20dp.

New in version 1.5.0.

scroll_timeout

Timeout allowed to trigger the *scroll_distance*, in milliseconds. If the user has not moved *scroll_distance* within the timeout, no scrolling will occur and the touch event will go to the children.

scroll_timeout is a *NumericProperty* and defaults to 200 (milliseconds)

New in version 1.5.0.

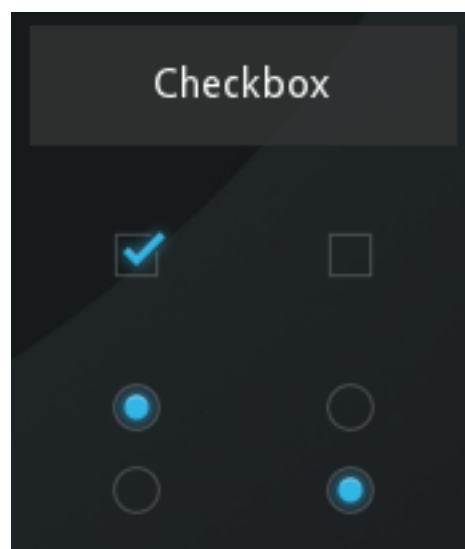
slides

List of slides inside the Carousel. The slides are the widgets added to the Carousel using the *add_widget* method.

slides is a *ListProperty* and is read-only.

33.11 CheckBox

New in version 1.4.0.



CheckBox is a specific two-state button that can be either checked or unchecked. If the *CheckBox* is in a *Group*, it becomes a *Radio* button. As with the *ToggleButton*, only one *Radio* button at a time can be selected when the *CheckBox.group* is set.

An example usage:

```

from kivy.uix.checkbox import CheckBox

# ...

def on_checkbox_active(checkbox, value):
    if value:
        print('The checkbox', checkbox, 'is active')
    else:
        print('The checkbox', checkbox, 'is inactive')

checkbox = CheckBox()
checkbox.bind(active=on_checkbox_active)

```

class kivy.uix.checkbox.**CheckBox**(**kwargs)
 Bases: *kivy.uix.behaviors.togglebutton.ToggleButtonBehavior*, *kivy.uix.widget.Widget*

CheckBox class, see module documentation for more information.

active

Indicates if the switch is active or inactive.

active is a boolean and reflects and sets whether the underlying state is 'down' (True) or 'normal' (False). It is a *AliasProperty*, which accepts boolean values and defaults to False.

Changed in version 1.11.0: It changed from a BooleanProperty to a AliasProperty.

background_checkbox_disabled_down

Background image of the checkbox used for the default graphical representation when the checkbox is disabled and active.

New in version 1.9.0.

background_checkbox_disabled_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_disabled_on'.

background_checkbox_disabled_normal

Background image of the checkbox used for the default graphical representation when the checkbox is disabled and not active.

New in version 1.9.0.

background_checkbox_disabled_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_disabled_off'.

background_checkbox_down

Background image of the checkbox used for the default graphical representation when the checkbox is active.

New in version 1.9.0.

background_checkbox_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_on'.

background_checkbox_normal

Background image of the checkbox used for the default graphical representation when the checkbox is not active.

New in version 1.9.0.

background_checkbox_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_off'.

background_radio_disabled_down

Background image of the radio button used for the default graphical representation when the radio button is disabled and active.

New in version 1.9.0.

background_radio_disabled_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_radio_disabled_on'.

background_radio_disabled_normal

Background image of the radio button used for the default graphical representation when the radio button is disabled and not active.

New in version 1.9.0.

background_radio_disabled_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_radio_disabled_off'.

background_radio_down

Background image of the radio button used for the default graphical representation when the radio button is active.

New in version 1.9.0.

background_radio_down is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_radio_on'.

background_radio_normal

Background image of the radio button used for the default graphical representation when the radio button is not active.

New in version 1.9.0.

background_radio_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/checkbox_radio_off'.

color

Color is used for tinting the default graphical representation of checkbox and radio button (images).

Color is in the format (r, g, b, a).

New in version 1.10.0.

color is a *ListProperty* and defaults to '[1, 1, 1, 1]'.

33.12 Code Input

New in version 1.5.0.

```

if __name__ == '__main__':
    from kivy.app import App
    from kivy.uix.boxlayout import BoxLayout

    class TextInputApp(App):

        def build(self):
            root = BoxLayout(orientation='vertical')
            textinput = TextInput(multiline=True)
            textinput.text = __doc__
            root.add_widget(textinput)
            textinput2 = TextInput(text='monoline textinput',
                                   size_hint=(1, None), height=30)
            root.add_widget(textinput2)
            return root

    TextInputApp().run()

```

```

BoxLayout:
    # Double as a Tabbed Panel Demo!
    TabbedPanel:
        tab_pos: "top_right"
        default_tab_text: "List View"
        default_tab_content: list_view_tab

    TabbedPanelHeader:
        text: 'Icon View'
        content: icon_view_tab

    FileChooserListView:
        id: list_view_tab

    FileChooserIconView:
        id: icon_view_tab
        show_hidden: True

```

Note: This widget requires `pygments` package to run. Install it with `pip`.

The `CodeInput` provides a box of editable highlighted text like the one shown in the image.

It supports all the features provided by the `textinput` as well as code highlighting for languages supported by `pygments` along with `KivyLexer` for `kivy.lang` highlighting.

33.12.1 Usage example

To create a `CodeInput` with highlighting for *KV language*:

```

from kivy.uix.codeinput import CodeInput
from kivy.extras.highlight import KivyLexer
codeinput = CodeInput(lexer=KivyLexer())

```

To create a `CodeInput` with highlighting for *Cython*:

```

from kivy.uix.codeinput import CodeInput
from pygments.lexers import CythonLexer
codeinput = CodeInput(lexer=CythonLexer())

```

class `kivy.uix.codeinput.CodeInput(**kwargs)`

Bases: `kivy.uix.behaviors.codenavigation.CodeNavigationBehavior`, `kivy.uix.textinput.TextInput`

`CodeInput` class, used for displaying highlighted code.

lexer

This holds the selected Lexer used by `pygments` to highlight the code.

`lexer` is an `ObjectProperty` and defaults to `PythonLexer`.

style

The `pygments` style object to use for formatting.

When `style_name` is set, this will be changed to the corresponding style object.

`style` is a `ObjectProperty` and defaults to `None`

style_name

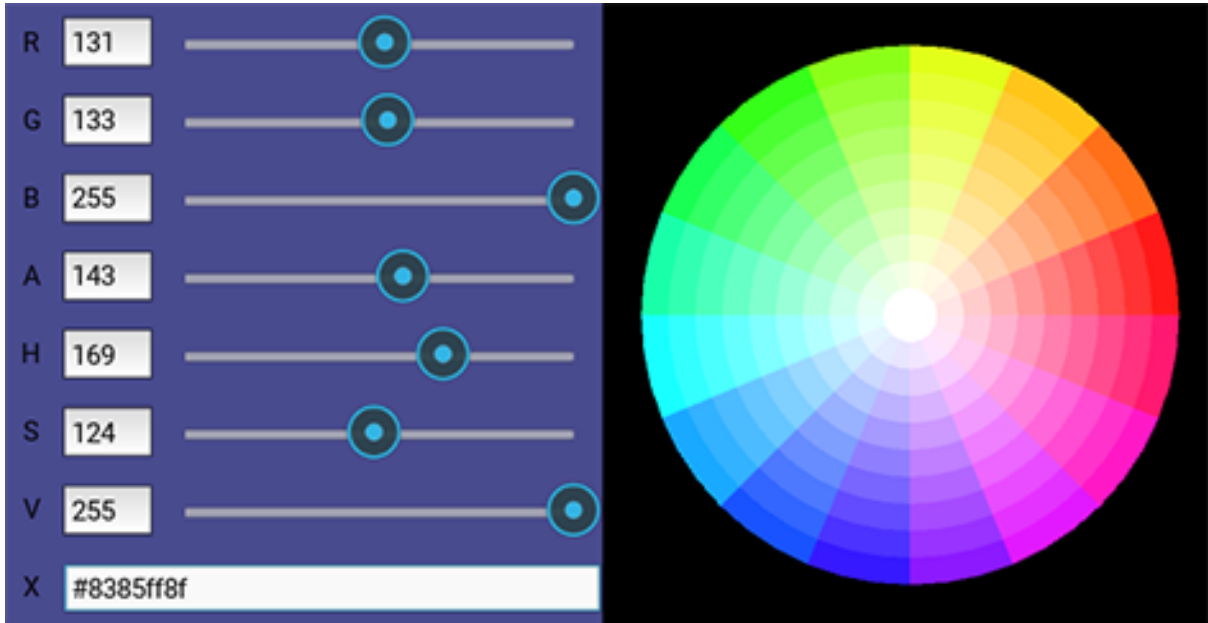
Name of the `pygments` style to use for formatting.

style_name is an *OptionProperty* and defaults to 'default'.

33.13 Color Picker

New in version 1.7.0.

Warning: This widget is experimental. Its use and API can change at any time until this warning is removed.



The ColorPicker widget allows a user to select a color from a chromatic wheel where pinch and zoom can be used to change the wheel's saturation. Sliders and TextInputs are also provided for entering the RGBA/HSV/HEX values directly.

Usage:

```
clr_picker = ColorPicker()
parent.add_widget(clr_picker)

# To monitor changes, we can bind to color property changes
def on_color(instance, value):
    print "RGBA = ", str(value) # or instance.color
    print "HSV = ", str(instance.hsv)
    print "HEX = ", str(instance.hex_color)

clr_picker.bind(color=on_color)
```

```
class kivy.uix.colorpicker.ColorPicker(**kwargs)
    Bases: kivy.uix.relativelayout.RelativeLayout
```

See module documentation.

color

The *color* holds the color currently selected in rgba format.

color is a *ListProperty* and defaults to (1, 1, 1, 1).

font_name

Specifies the font used on the ColorPicker.

font_name is a *StringProperty* and defaults to 'data/fonts/RobotoMono-Regular.ttf'.

hex_color

The *hex_color* holds the currently selected color in hex.

hex_color is an *AliasProperty* and defaults to #ffffff.

hsv

The *hsv* holds the color currently selected in hsv format.

hsv is a *ListProperty* and defaults to (1, 1, 1).

wheel

The *wheel* holds the color wheel.

wheel is an *ObjectProperty* and defaults to None.

class kivy.uix.colorpicker.ColorWheel(**kwargs)

Bases: *kivy.uix.widget.Widget*

Chromatic wheel for the ColorPicker.

Changed in version 1.7.1: *font_size*, *font_name* and *foreground_color* have been removed. The sizing is now the same as others widget, based on 'sp'. Orientation is also automatically determined according to the width/height ratio.

a

The Alpha value of the color currently selected.

a is a *BoundedNumericProperty* and can be a value from 0 to 1.

b

The Blue value of the color currently selected.

b is a *BoundedNumericProperty* and can be a value from 0 to 1.

color

The holds the color currently selected.

color is a *ReferenceListProperty* and contains a list of *r*, *g*, *b*, *a* values.

g

The Green value of the color currently selected.

g is a *BoundedNumericProperty* and can be a value from 0 to 1.

on_touch_down(touch)

Receive a touch down event.

Parameters

touch: **MotionEvent** class Touch received. The touch is in parent coordinates.
See *relative_layout* 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.

r

The Red value of the color currently selected.

r is a *BoundedNumericProperty* and can be a value from 0 to 1. It defaults to 0.

33.14 Drop-Down List

New in version 1.4.0.

A versatile drop-down list that can be used with custom widgets. It allows you to display a list of widgets under a displayed widget. Unlike other toolkits, the list of widgets can contain any type of widget: simple buttons, images etc.

The positioning of the drop-down list is fully automatic: we will always try to place the dropdown list in a way that the user can select an item in the list.

33.14.1 Basic example

A button with a dropdown list of 10 possible values. All the buttons within the dropdown list will trigger the dropdown *DropDown.select()* method. After being called, the main button text will display the selection of the dropdown.

```
from kivy.uix.dropdown import DropDown
from kivy.uix.button import Button
from kivy.base import runTouchApp

# create a dropdown with 10 buttons
dropdown = DropDown()
for index in range(10):
    # When adding widgets, we need to specify the height manually
    # (disabling the size_hint_y) so the dropdown can calculate
    # the area it needs.

    btn = Button(text='Value %d' % index, size_hint_y=None, height=44)

    # for each button, attach a callback that will call the select() method
    # on the dropdown. We'll pass the text of the button as the data of the
    # selection.
    btn.bind(on_release=lambda btn: dropdown.select(btn.text))

    # then add the button inside the dropdown
    dropdown.add_widget(btn)

# create a big main button
mainbutton = Button(text='Hello', size_hint=(None, None))

# show the dropdown menu when the main button is released
# note: all the bind() calls pass the instance of the caller (here, the
# mainbutton instance) as the first argument of the callback (here,
# dropdown.open.).
mainbutton.bind(on_release=dropdown.open)

# one last thing, listen for the selection in the dropdown list and
# assign the data to the button text.
dropdown.bind(on_select=lambda instance, x: setattr(mainbutton, 'text', x))
```

(continues on next page)


```
runTouchApp(mainbutton)
```

33.14.2 Extending dropdown in Kv

You could create a dropdown directly from your kv:

```
#:kivy 1.4.0
<CustomDropDown>:
    Button:
        text: 'My first Item'
        size_hint_y: None
        height: 44
        on_release: root.select('item1')
    Label:
        text: 'Unselectable item'
        size_hint_y: None
        height: 44
    Button:
        text: 'My second Item'
        size_hint_y: None
        height: 44
        on_release: root.select('item2')
```

And then, create the associated python class and use it:

```
class CustomDropDown(DropDown):
    pass

dropdown = CustomDropDown()
mainbutton = Button(text='Hello', size_hint=(None, None))
mainbutton.bind(on_release=dropdown.open)
dropdown.bind(on_select=lambda instance, x: setattr(mainbutton, 'text', x))
```

```
class kivy.uix.dropdown.DropDown(**kwargs)
```

Bases: *kivy.uix.scrollview.ScrollView*

DropDown class. See module documentation for more information.

Events

on_select: data Fired when a selection is done. The data of the selection is passed in as the first argument and is what you pass in the *select()* method as the first argument.

on_dismiss: New in version 1.8.0.

Fired when the DropDown is dismissed, either on selection or on touching outside the widget.

```
add_widget(*args)
```

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)
```

attach_to

(internal) Property that will be set to the widget to which the drop down list is attached.

The [open\(\)](#) method will automatically set this property whilst [dismiss\(\)](#) will set it back to None.

auto_dismiss

By default, the dropdown will be automatically dismissed when a touch happens outside of it, this option allows to disable this feature

[auto_dismiss](#) is a [BooleanProperty](#) and defaults to True.

New in version 1.8.0.

auto_width

By default, the width of the dropdown will be the same as the width of the attached widget. Set to False if you want to provide your own width.

[auto_width](#) is a [BooleanProperty](#) and defaults to True.

clear_widgets()

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

container

(internal) Property that will be set to the container of the dropdown list. It is a [GridLayout](#) by default.

dismiss(*args)

Remove the dropdown widget from the window and detach it from the attached widget.

dismiss_on_select

By default, the dropdown will be automatically dismissed when a selection has been done. Set to False to prevent the dismiss.

[dismiss_on_select](#) is a [BooleanProperty](#) and defaults to True.

max_height

Indicate the maximum height that the dropdown can take. If None, it will take the maximum height available until the top or bottom of the screen is reached.

[max_height](#) is a [NumericProperty](#) and defaults to None.

min_state_time

Minimum time before the **DropDown** is dismissed. This is used to allow for the widget inside

the dropdown to display a down state or for the `DropDown` itself to display a animation for closing.

`min_state_time` is a *NumericProperty* and defaults to the *Config* value `min_state_time`.

New in version 1.10.0.

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.

open(*widget*)

Open the dropdown list and attach it to a specific widget. Depending on the position of the widget within the window and the height of the dropdown, the dropdown might be above or below that widget.

remove_widget(**largs*)

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)
```

select(*data*)

Call this method to trigger the *on_select* event with the *data* selection. The *data* can be anything you want.

33.15 EffectWidget

New in version 1.9.0.

The *EffectWidget* is able to apply a variety of fancy graphical effects to its children. It works by rendering to a series of *Fbo* instances with custom opengl fragment shaders. As such, effects can freely do almost anything, from inverting the colors of the widget, to anti-aliasing, to emulating the appearance of a crt monitor!

Warning: This code is still experimental, and its API is subject to change in a future version.

The basic usage is as follows:

```
w = EffectWidget()
w.add_widget(Button(text='Hello!'))
w.effects = [InvertEffect(), HorizontalBlurEffect(size=2.0)]
```

The equivalent in kv would be:

```
#: import ew kivy.uix.effectwidget
EffectWidget:
    effects: ew.InvertEffect(), ew.HorizontalBlurEffect(size=2.0)
    Button:
        text: 'Hello!'
```

The effects can be a list of effects of any length, and they will be applied sequentially.

The module comes with a range of prebuilt effects, but the interface is designed to make it easy to create your own. Instead of writing a full glsl shader, you provide a single function that takes some inputs based on the screen (current pixel color, current widget texture etc.). See the sections below for more information.

33.15.1 Usage Guidelines

It is not efficient to resize an *EffectWidget*, as the *Fbo* is recreated on each resize event. If you need to resize frequently, consider doing things a different way.

Although some effects have adjustable parameters, it is *not* efficient to animate these, as the entire shader is reconstructed every time. You should use glsl uniform variables instead. The *AdvancedEffectBase* may make this easier.

Note: The *EffectWidget* *cannot* draw outside its own widget area (pos -> pos + size). Any child widgets overlapping the boundary will be cut off at this point.

33.15.2 Provided Effects

The module comes with several pre-written effects. Some have adjustable properties (e.g. blur radius). Please see the individual effect documentation for more details.

- *MonochromeEffect* - makes the widget grayscale.
- *InvertEffect* - inverts the widget colors.
- *ChannelMixEffect* - swaps color channels.
- *ScanlinesEffect* - displays flickering scanlines.
- *PixelateEffect* - pixelates the image.
- *HorizontalBlurEffect* - Gaussuan blurs horizontally.
- *VerticalBlurEffect* - Gaussuan blurs vertically.
- *FXAAEffect* - applies a very basic anti-aliasing.

33.15.3 Creating Effects

Effects are designed to make it easy to create and use your own transformations. You do this by creating and using an instance of *EffectBase* with your own custom *EffectBase.glsl* property.

The `glsl` property is a string representing part of a glsl fragment shader. You can include as many functions as you like (the string is simply spliced into the whole shader), but it must implement a function `effect` as below:

```
vec4 effect(vec4 color, sampler2D texture, vec2 tex_coords, vec2 coords)
{
    // ... your code here
    return something; // must be a vec4 representing the new color
}
```

The full shader will calculate the normal pixel color at each point, then call your `effect` function to transform it. The parameters are:

- **color**: The normal color of the current pixel (i.e. texture sampled at `tex_coords`).
- **texture**: The texture containing the widget's normal background.
- **tex_coords**: The normal texture_coords used to access texture.
- **coords**: The pixel indices of the current pixel.

The shader code also has access to two useful uniform variables, `time` containing the time (in seconds) since the program start, and `resolution` containing the shape (x pixels, y pixels) of the widget.

For instance, the following simple string (taken from the *InvertEffect*) would invert the input color but set alpha to 1.0:

```
vec4 effect(vec4 color, sampler2D texture, vec2 tex_coords, vec2 coords)
{
    return vec4(1.0 - color.xyz, 1.0);
}
```

You can also set the `glsl` by automatically loading the string from a file, simply set the *EffectBase.source* property of an effect.

class `kivy.uix.effectwidget.EffectWidget`(*kwargs)
Bases: *kivy.uix.relativelayout.RelativeLayout*

Widget with the ability to apply a series of graphical effects to its children. See the module documentation for more information on setting effects and creating your own.

add_widget(*widget*)

Add a new widget as a child of this widget.

Parameters

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

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

New in version 1.0.5.

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

New in version 1.9.0.

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

(continues on next page)

```
>>> root.add_widget(Button())
>>> slider = Slider()
>>> root.add_widget(slider)
```

background_color

This defines the background color to be used for the fbo in the EffectWidget.

background_color is a *ListProperty* defaults to (0, 0, 0, 0)

clear_widgets (*children=None*)

Remove all (or the specified) *children* of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

effects

List of all the effects to be applied. These should all be instances or subclasses of *EffectBase*.

effects is a *ListProperty* and defaults to [].

fbo_list

(internal) List of all the fbos that are being used to apply the effects.

fbo_list is a *ListProperty* and defaults to [].

refresh_fbo_setup (**args*)

(internal) Creates and assigns one *Fbo* per effect, and makes sure all sizes etc. are correct and consistent.

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)
```

texture

The output texture of the final *Fbo* after all effects have been applied.

texture is an *ObjectProperty* and defaults to None.

class kivy.uix.effectwidget.**EffectBase** (**args, **kwargs*)

Bases: *kivy.event.EventDispatcher*

The base class for GLSL effects. It simply returns its input.

See the module documentation for more details.

fbo

The fbo currently using this effect. The *EffectBase* automatically handles this.

fbo is an *ObjectProperty* and defaults to None.

glsl

The glsl string defining your effect function. See the module documentation for more details.

glsl is a *StringProperty* and defaults to a trivial effect that returns its input.

set_fbo_shader(*args)

Sets the *Fbo*'s shader by splicing the *glsl* string into a full fragment shader.

The full shader is made up of `shader_header + shader_uniforms + self.glsl + shader_footer_effect`.

source

The (optional) filename from which to load the *glsl* string.

source is a *StringProperty* and defaults to "".

class `kivy.uix.effectwidget.AdvancedEffectBase(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

An *EffectBase* with additional behavior to easily set and update uniform variables in your shader.

This class is provided for convenience when implementing your own effects: it is not used by any of those provided with Kivy.

In addition to your base *glsl* string that must be provided as normal, the *AdvancedEffectBase* has an extra property *uniforms*, a dictionary of name-value pairs. Whenever a value is changed, the new value for the uniform variable is uploaded to the shader.

You must still manually declare your uniform variables at the top of your *glsl* string.

set_fbo_shader(*args)

Sets the *Fbo*'s shader by splicing the *glsl* string into a full fragment shader.

The full shader is made up of `shader_header + shader_uniforms + self.glsl + shader_footer_effect`.

uniforms

A dictionary of uniform variable names and their values. These are automatically uploaded to the *fbo* shader if appropriate.

uniforms is a *DictProperty* and defaults to {}.

class `kivy.uix.effectwidget.MonochromeEffect(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

Returns its input colors in monochrome.

class `kivy.uix.effectwidget.InvertEffect(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

Inverts the colors in the input.

class `kivy.uix.effectwidget.ChannelMixEffect(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

Mixes the color channels of the input according to the *order* property. Channels may be arbitrarily rearranged or repeated.

order

The new sorted order of the *rgb* channels.

order is a *ListProperty* and defaults to [1, 2, 0], corresponding to (g, b, r).

class `kivy.uix.effectwidget.ScanlinesEffect(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

Adds scanlines to the input.

class `kivy.uix.effectwidget.PixelateEffect(*args, **kwargs)`

Bases: *kivy.uix.effectwidget.EffectBase*

Pixelates the input according to its *pixel_size*

pixel_size

Sets the size of a new ‘pixel’ in the effect, in terms of number of ‘real’ pixels.

pixel_size is a *NumericProperty* and defaults to 10.

class kivy.uix.effectwidget.**HorizontalBlurEffect**(*args, **kwargs)

Bases: *kivy.uix.effectwidget.EffectBase*

Blurs the input horizontally, with the width given by *size*.

size

The blur width in pixels.

size is a *NumericProperty* and defaults to 4.0.

class kivy.uix.effectwidget.**VerticalBlurEffect**(*args, **kwargs)

Bases: *kivy.uix.effectwidget.EffectBase*

Blurs the input vertically, with the width given by *size*.

size

The blur width in pixels.

size is a *NumericProperty* and defaults to 4.0.

class kivy.uix.effectwidget.**FXAAEffect**(*args, **kwargs)

Bases: *kivy.uix.effectwidget.EffectBase*

Applies very simple anti-aliasing via fxaa.

33.16 FileChooser

The FileChooser module provides various classes for describing, displaying and browsing file systems.

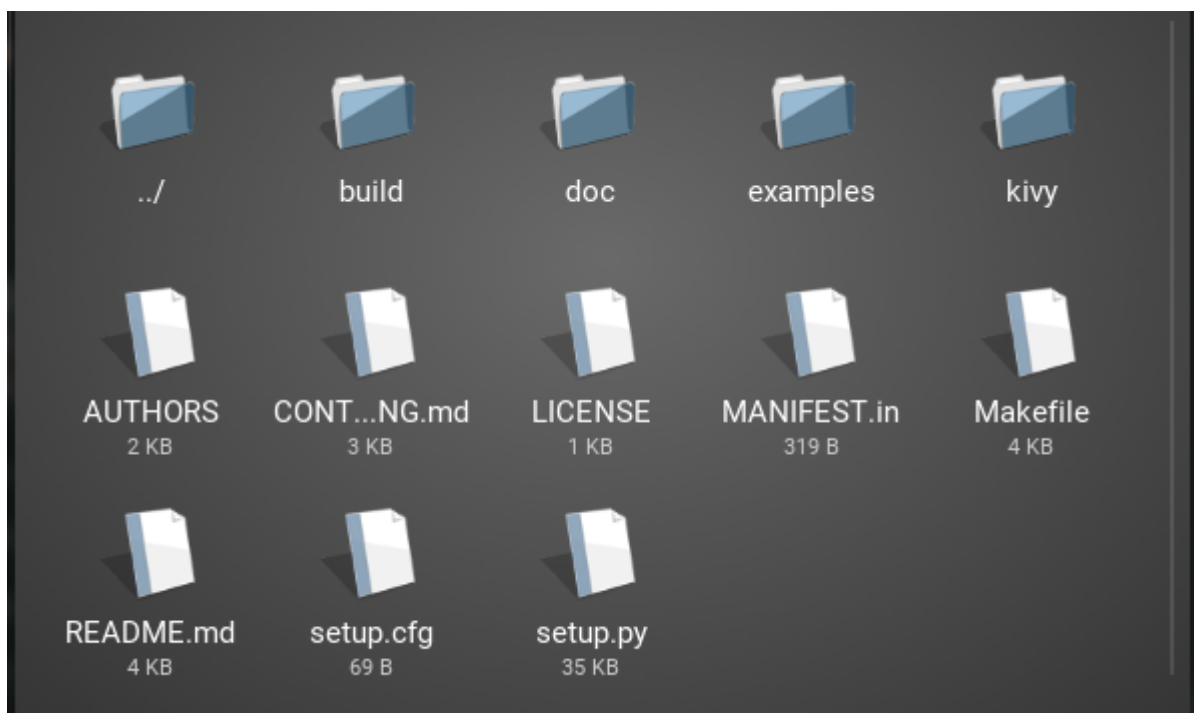
33.16.1 Simple widgets

There are two ready-to-use widgets that provide views of the file system. Each of these present the files and folders in a different style.

The *FileChooserListView* displays file entries as text items in a vertical list, where folders can be collapsed and expanded.

Name	Size
../	
> build	
> doc	
> examples	
> kivy	
AUTHORS	2 KB
CONTRIBUTING.md	3 KB
LICENSE	1 KB
MANIFEST.in	319 B
Makefile	4 KB
README.md	4 KB
setup.cfg	69 B
setup.py	35 KB

The *FileChooserIconView* presents icons and text from left to right, wrapping them as required.



They both provide for scrolling, selection and basic user interaction. Please refer to the *FileChooserController* for details on supported events and properties.

33.16.2 Widget composition

FileChooser classes adopt a *MVC* design. They are exposed so that you to extend and customize your file chooser according to your needs.

The FileChooser classes can be categorized as follows:

- Models are represented by concrete implementations of the *FileSystemAbstract* class, such as the *FileSystemLocal*.
- Views are represented by the *FileChooserListLayout* and *FileChooserIconLayout* classes. These are used by the *FileChooserListView* and *FileChooserIconView* widgets respectively.
- Controllers are represented by concrete implementations of the *FileChooserController*, namely the *FileChooser*, *FileChooserIconView* and *FileChooserListView* classes.

This means you can define your own views or provide *FileSystemAbstract* implementations for alternative file systems for use with these widgets. The *FileChooser* can be used as a controller for handling multiple, synchronized views of the same path. By combining these elements, you can add your own views and file systems and have them easily interact with the existing components.

33.16.3 Usage example

main.py

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.factory import Factory
from kivy.properties import ObjectProperty
from kivy.uix.popup import Popup

import os

class LoadDialog(FloatLayout):
    load = ObjectProperty(None)
    cancel = ObjectProperty(None)

class SaveDialog(FloatLayout):
    save = ObjectProperty(None)
    text_input = ObjectProperty(None)
    cancel = ObjectProperty(None)

class Root(FloatLayout):
    loadfile = ObjectProperty(None)
    savefile = ObjectProperty(None)
    text_input = ObjectProperty(None)

    def dismiss_popup(self):
        self._popup.dismiss()

    def show_load(self):
        content = LoadDialog(load=self.load, cancel=self.dismiss_popup)
        self._popup = Popup(title="Load file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()

    def show_save(self):
        content = SaveDialog(save=self.save, cancel=self.dismiss_popup)
        self._popup = Popup(title="Save file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()
```

(continues on next page)

(continued from previous page)

```
def load(self, path, filename):
    with open(os.path.join(path, filename[0])) as stream:
        self.text_input.text = stream.read()

    self.dismiss_popup()

def save(self, path, filename):
    with open(os.path.join(path, filename), 'w') as stream:
        stream.write(self.text_input.text)

    self.dismiss_popup()

class Editor(App):
    pass

Factory.register('Root', cls=Root)
Factory.register('LoadDialog', cls=LoadDialog)
Factory.register('SaveDialog', cls=SaveDialog)

if __name__ == '__main__':
    Editor().run()
```

editor.kv

```
#:kivy 1.1.0

Root:
    text_input: text_input

    BoxLayout:
        orientation: 'vertical'
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: 'Load'
                on_release: root.show_load()
            Button:
                text: 'Save'
                on_release: root.show_save()

        BoxLayout:
            TextInput:
                id: text_input
                text: ''

            RstDocument:
                text: text_input.text
                show_errors: True

<LoadDialog>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
```

(continues on next page)

```

FileChooserListView:
    id: filechooser

BoxLayout:
    size_hint_y: None
    height: 30
    Button:
        text: "Cancel"
        on_release: root.cancel()

    Button:
        text: "Load"
        on_release: root.load(filechooser.path, filechooser.selection)

<SaveDialog>:
    text_input: text_input
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        FileChooserListView:
            id: filechooser
            on_selection: text_input.text = self.selection and self.selection[0]_
or
    TextInput:
        id: text_input
        size_hint_y: None
        height: 30
        multiline: False

    BoxLayout:
        size_hint_y: None
        height: 30
        Button:
            text: "Cancel"
            on_release: root.cancel()

        Button:
            text: "Save"
            on_release: root.save(filechooser.path, text_input.text)

```

New in version 1.0.5.

Changed in version 1.2.0: In the chooser template, the *controller* is no longer a direct reference but a weak-reference. If you are upgrading, you should change the notation *root.controller.xxx* to *root.controller().xxx*.

class kivy.uix.filechooser.FileChooserListView(**kwargs)

Bases: *kivy.uix.filechooser.FileChooserController*

Implementation of a *FileChooserController* using a list view.

New in version 1.9.0.

class kivy.uix.filechooser.FileChooserIconView(**kwargs)

Bases: *kivy.uix.filechooser.FileChooserController*

Implementation of a *FileChooserController* using an icon view.

New in version 1.9.0.

class kivy.uix.filechooser.**FileChooserListLayout**(**kwargs)

Bases: kivy.uix.filechooser.FileChooserLayout

File chooser layout using a list view.

New in version 1.9.0.

class kivy.uix.filechooser.**FileChooserIconLayout**(**kwargs)

Bases: kivy.uix.filechooser.FileChooserLayout

File chooser layout using an icon view.

New in version 1.9.0.

class kivy.uix.filechooser.**FileChooser**(**kwargs)

Bases: *kivy.uix.filechooser.FileChooserController*

Implementation of a *FileChooserController* which supports switching between multiple, synced layout views.

The FileChooser can be used as follows:

```
BoxLayout:
    orientation: 'vertical'

    BoxLayout:
        size_hint_y: None
        height: sp(52)

        Button:
            text: 'Icon View'
            on_press: fc.view_mode = 'icon'
        Button:
            text: 'List View'
            on_press: fc.view_mode = 'list'

    FileChooser:
        id: fc
        FileChooserIconLayout
        FileChooserListLayout
```

New in version 1.9.0.

add_widget(widget, **kwargs)

Add a new widget as a child of this widget.

Parameters

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

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

New in version 1.0.5.

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

New in version 1.9.0.

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

manager

Reference to the *ScreenManager* instance.

manager is an *ObjectProperty*.

view_list

List of views added to this FileChooser.

view_list is an *AliasProperty* of type list.

view_mode

Current layout view mode.

view_mode is an *AliasProperty* of type str.

class kivy.uix.filechooser.**FileChooserController**(*kwargs)

Bases: *kivy.uix.relativelayout.RelativeLayout*

Base for implementing a FileChooser. Don't use this class directly, but prefer using an implementation such as the *FileChooser*, *FileChooserListView* or *FileChooserIconView*.

Events

on_entry_added: entry, parent Fired when a root-level entry is added to the file list.
If you return True from this event, the entry is not added to FileChooser.

on_entries_cleared Fired when the the entries list is cleared, usually when the root is refreshed.

on_subentry_to_entry: entry, parent Fired when a sub-entry is added to an existing entry or when entries are removed from an entry e.g. when a node is closed.

on_submit: selection, touch Fired when a file has been selected with a double-tap.

cancel(*largs)

Cancel any background action started by filechooser, such as loading a new directory.

New in version 1.2.0.

dirselect

Determines whether directories are valid selections or not.

dirselect is a *BooleanProperty* and defaults to False.

New in version 1.1.0.

entry_released(entry, touch)

(internal) This method must be called by the template when an entry is touched by the user.

New in version 1.1.0.

entry_touched(entry, touch)

(internal) This method must be called by the template when an entry is touched by the user.

file_encodings

Possible encodings for decoding a filename to unicode. In the case that the user has a non-ascii filename, undecodable without knowing its initial encoding, we have no other choice than to guess it.

Please note that if you encounter an issue because of a missing encoding here, we'll be glad to add it to this list.

file_encodings is a *ListProperty* and defaults to ['utf-8', 'latin1', 'cp1252'].

New in version 1.3.0.

Deprecated since version 1.8.0: This property is no longer used as the filechooser no longer decodes the file names.

file_system

The file system object used to access the file system. This should be a subclass of *FileSystemAbstract*.

file_system is an *ObjectProperty* and defaults to *FileSystemLocal()*

New in version 1.8.0.

files

The list of files in the directory specified by path after applying the filters.

files is a read-only *ListProperty*.

filter_dirs

Indicates whether filters should also apply to directories. filter_dirs is a *BooleanProperty* and defaults to False.

filters

filters specifies the filters to be applied to the files in the directory. filters is a *ListProperty* and defaults to []. This is equivalent to '*' i.e. nothing is filtered.

The filters are not reset when the path changes. You need to do that yourself if desired.

There are two kinds of filters: patterns and callbacks.

1. Patterns

e.g. ['*.png']. You can use the following patterns:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

2. Callbacks

You can specify a function that will be called for each file. The callback will be passed the folder and file name as the first and second parameters respectively. It should return True to indicate a match and False otherwise.

Changed in version 1.4.0: Added the option to specify the filter as a callback.

get_nice_size(fn)

Pass the filepath. Returns the size in the best human readable format or '' if it is a directory (Don't recursively calculate size).

layout

Reference to the layout widget instance.

layout is an *ObjectProperty*.

New in version 1.9.0.

multiselect

Determines whether the user is able to select multiple files or not.

multiselect is a *BooleanProperty* and defaults to False.

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.

path

path is a *StringProperty* and defaults to the current working directory as a unicode string. It specifies the path on the filesystem that this controller should refer to.

Warning: If a unicode path is specified, all the files returned will be in unicode, allowing the display of unicode files and paths. If a bytes path is specified, only files and paths with ascii names will be displayed properly: non-ascii filenames will be displayed and listed with questions marks (?) instead of their unicode characters.

progress_cls

Class to use for displaying a progress indicator for filechooser loading.

progress_cls is an *ObjectProperty* and defaults to FileChooserProgress.

New in version 1.2.0.

Changed in version 1.8.0: If set to a string, the *Factory* will be used to resolve the class name.

rootpath

Root path to use instead of the system root path. If set, it will not show a “..” directory to go up to the root path. For example, if you set rootpath to /users/foo, the user will be unable to go to /users or to any other directory not starting with /users/foo.

rootpath is a *StringProperty* and defaults to None.

New in version 1.2.0.

Note: Similarly to *path*, whether *rootpath* is specified as bytes or a unicode string determines the type of the filenames and paths read.

selection

Contains the list of files that are currently selected.

selection is a read-only *ListProperty* and defaults to [].

show_hidden

Determines whether hidden files and folders should be shown.

show_hidden is a *BooleanProperty* and defaults to False.

sort_func

Provides a function to be called with a list of filenames as the first argument and the filesystem implementation as the second argument. It returns a list of filenames sorted for display in the view.

sort_func is an *ObjectProperty* and defaults to a function returning alphanumerically named folders first.

Changed in version 1.8.0: The signature needs now 2 arguments: first the list of files, second the filesystem class to use.

class kivy.uix.filechooser.**FileChooserProgressBase**(*kwargs)

Bases: *kivy.uix.floatlayout.FloatLayout*

Base for implementing a progress view. This view is used when too many entries need to be created and are delayed over multiple frames.

New in version 1.2.0.

cancel(*largs)

Cancel any action from the FileChooserController.

index

Current index of *total* entries to be loaded.

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.

path

Current path of the FileChooser, read-only.

total

Total number of entries to load.

class kivy.uix.filechooser.**FileSystemAbstract**

Bases: *builtins.object*

Class for implementing a File System view that can be used with the *FileChooser*.

New in version 1.8.0.

getsize(fn)

Return the size in bytes of a file

is_dir(fn)

Return True if the argument passed to this method is a directory

is_hidden(fn)

Return True if the file is hidden

listdir(*fn*)

Return the list of files in the directory *fn*

class kivy.uix.filechooser.**FileSystemLocal**

Bases: *kivy.uix.filechooser.FileSystemAbstract*

Implementation of *FileSystemAbstract* for local files.

New in version 1.8.0.

getsize(*fn*)

Return the size in bytes of a file

is_dir(*fn*)

Return True if the argument passed to this method is a directory

is_hidden(*fn*)

Return True if the file is hidden

listdir(*fn*)

Return the list of files in the directory *fn*

33.17 Float Layout

FloatLayout honors the *pos_hint* and the *size_hint* properties of its children.



For example, a FloatLayout with a size of (300, 300) is created:

```
layout = FloatLayout(size=(300, 300))
```

By default, all widgets have their *size_hint*=(1, 1), so this button will adopt the same size as the layout:

```
button = Button(text='Hello world')  
layout.add_widget(button)
```

To create a button 50% of the width and 25% of the height of the layout and positioned at (20, 20), you can do:

```
button = Button(
    text='Hello world',
    size_hint=(.5, .25),
    pos=(20, 20))
```

If you want to create a button that will always be the size of layout minus 20% on each side:

```
button = Button(text='Hello world', size_hint=(.6, .6),
    pos_hint={'x':.2, 'y':.2})
```

Note: This layout can be used for an application. Most of the time, you will use the size of Window.

Warning: If you are not using `pos_hint`, you must handle the positioning of the children: if the float layout is moving, you must handle moving the children too.

class `kivy.uix.floatlayout.FloatLayout` (***kwargs*)

Bases: `kivy.uix.layout.Layout`

Float layout class. See module documentation for more information.

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)
```

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.

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)
```

33.18 Gesture Surface

New in version 1.9.0.

Warning: This is experimental and subject to change as long as this warning notice is present.

See `kivy/examples/demo/multistroke/main.py` for a complete application example.

class `kivy.uix.gesturesurface.GestureSurface`(**kwargs)

Bases: `kivy.uix.floatlayout.FloatLayout`

Simple gesture surface to track/draw touch movements. Typically used to gather user input suitable for `kivy.multistroke.Recognizer`.

Properties

temporal_window Time to wait from the last touch_up event before attempting to recognize the gesture. If you set this to 0, the *on_gesture_complete* event is not fired unless the *max_strokes* condition is met.

temporal_window is a *NumericProperty* and defaults to 2.0

max_strokes Max number of strokes in a single gesture; if this is reached, recognition will start immediately on the final touch_up event. If this is set to 0, the *on_gesture_complete* event is not fired unless the *temporal_window* expires.

max_strokes is a *NumericProperty* and defaults to 2.0

bbox_margin Bounding box margin for detecting gesture collisions, in pixels.

bbox_margin is a *NumericProperty* and defaults to 30

draw_timeout Number of seconds to keep lines/bbox on canvas after the *on_gesture_complete* event is fired. If this is set to 0, gestures are immediately removed from the surface when complete.

draw_timeout is a *NumericProperty* and defaults to 3.0

color Color used to draw the gesture, in RGB. This option does not have an effect if *use_random_color* is True.

draw_timeout is a *ListProperty* and defaults to [1, 1, 1] (white)

use_random_color Set to True to pick a random color for each gesture, if you do this then *color* is ignored. Defaults to False.

use_random_color is a *BooleanProperty* and defaults to False

line_width Line width used for tracing touches on the surface. Set to 0 if you only want to detect gestures without drawing anything. If you use 1.0, OpenGL GL_LINE is used for drawing; values > 1 will use an internal drawing method based on triangles (less efficient), see [kivy.graphics](#).

line_width is a [NumericProperty](#) and defaults to 2

draw_bbox Set to True if you want to draw bounding box behind gestures. This only works if *line_width* >= 1. Default is False.

draw_bbox is a [BooleanProperty](#) and defaults to True

bbox_alpha Opacity for bounding box if *draw_bbox* is True. Default 0.1

bbox_alpha is a [NumericProperty](#) and defaults to 0.1

Events

on_gesture_start [GestureContainer](#) Fired when a new gesture is initiated on the surface, i.e. the first on_touch_down that does not collide with an existing gesture on the surface.

on_gesture_extend [GestureContainer](#) Fired when a touch_down event occurs within an existing gesture.

on_gesture_merge [GestureContainer](#), [GestureContainer](#) Fired when two gestures collide and get merged to one gesture. The first argument is the gesture that has been merged (no longer valid); the second is the combined (resulting) gesture.

on_gesture_complete [GestureContainer](#) Fired when a set of strokes is considered a complete gesture, this happens when *temporal_window* expires or *max_strokes* is reached. Typically you will bind to this event and use the provided *GestureContainer* *get_vectors()* method to match against your gesture database.

on_gesture_cleanup [GestureContainer](#) Fired *draw_timeout* seconds after *on_gesture_complete*, The gesture will be removed from the canvas (if *line_width* > 0 or *draw_bbox* is True) and the internal gesture list before this.

on_gesture_discard [GestureContainer](#) Fired when a gesture does not meet the minimum size requirements for recognition (width/height < 5, or consists only of single- point strokes).

find_colliding_gesture(touch)

Checks if a touch x/y collides with the bounding box of an existing gesture. If so, return it (otherwise returns None)

get_gesture(touch)

Returns *GestureContainer* associated with given touch

init_gesture(touch)

Create a new gesture from touch, i.e. it's the first on surface, or was not close enough to any existing gesture (yet)

merge_gestures(g, other)

Merges two gestures together, the oldest one is retained and the newer one gets the *GestureContainer.was_merged* flag raised.

on_touch_down(touch)

When a new touch is registered, the first thing we do is to test if it collides with the bounding box of another known gesture. If so, it is assumed to be part of that gesture.

on_touch_move(touch)

When a touch moves, we add a point to the line on the canvas so the path is updated. We

must also check if the new point collides with the bounding box of another gesture - if so, they should be merged.

on_touch_up(*touch*)

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

See *on_touch_down()* for more information.

class kivy.uix.gesturesurface.GestureContainer(*touch*, *kwargs*)**

Bases: *kivy.event.EventDispatcher*

Container object that stores information about a gesture. It has various properties that are updated by *GestureSurface* as drawing progresses.

Arguments

touch Touch object (as received by *on_touch_down*) used to initialize the gesture container. Required.

Properties

active Set to False once the gesture is complete (meets *max_stroke* setting or *GestureSurface.temporal_window*)

active is a *BooleanProperty*

active_strokes Number of strokes currently active in the gesture, ie concurrent touches associated with this gesture.

active_strokes is a *NumericProperty*

max_strokes Max number of strokes allowed in the gesture. This is set by *GestureSurface.max_strokes* but can be overridden for example from *on_gesture_start*.

max_strokes is a *NumericProperty*

was_merged Indicates that this gesture has been merged with another gesture and should be considered discarded.

was_merged is a *BooleanProperty*

bbox Dictionary with keys minx, miny, maxx, maxy. Represents the size of the gesture bounding box.

bbox is a *DictProperty*

width Represents the width of the gesture.

width is a *NumericProperty*

height Represents the height of the gesture.

height is a *NumericProperty*

accept_stroke(*count*=1)

Returns True if this container can accept *count* new strokes

add_stroke(*touch*, *line*)

Associate a list of points with a touch.uid; the line itself is created by the caller, but subsequent move/up events look it up via us. This is done to avoid problems during merge.

complete_stroke()

Called on touch up events to keep track of how many strokes are active in the gesture (we only want to dispatch event when the *last* stroke in the gesture is released)

get_vectors(*kwargs*)**

Return strokes in a format that is acceptable for *kivy.multistroke.Recognizer* as a gesture candidate or template. The result is cached automatically; the cache is invalidated at the start and

end of a stroke and if *update_bbox* is called. If you are going to analyze a gesture mid-stroke, you may need to set the *no_cache* argument to True.

handles(*touch*)

Returns True if this container handles the given touch

single_points_test()

Returns True if the gesture consists only of single-point strokes, we must discard it in this case, or an exception will be raised

update_bbox(*touch*)

Update gesture bbox from a touch coordinate

33.19 Grid Layout



New in version 1.0.4.

The *GridLayout* arranges children in a matrix. It takes the available space and divides it into columns and rows, then adds widgets to the resulting “cells”.

Changed in version 1.0.7: The implementation has changed to use the widget *size_hint* for calculating column/row sizes. *uniform_width* and *uniform_height* have been removed and other properties have been added to give you more control.

33.19.1 Background

Unlike many other toolkits, you cannot explicitly place a widget in a specific column/row. Each child is automatically assigned a position determined by the layout configuration and the child’s index in the children list.

A *GridLayout* must always have at least one input constraint: *GridLayout.cols* or *GridLayout.rows*. If you do not specify cols or rows, the Layout will throw an exception.

33.19.2 Column Width and Row Height

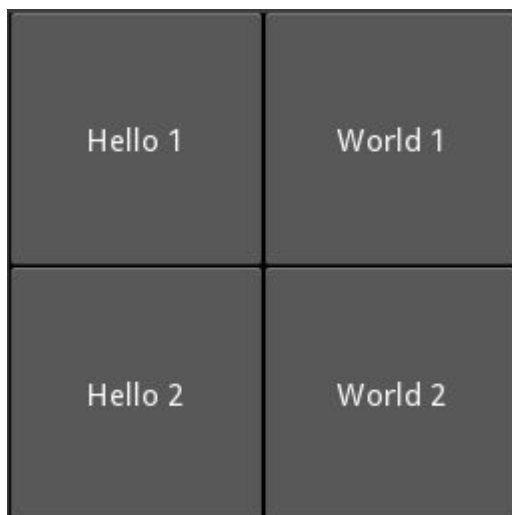
The column width/row height are determined in 3 steps:

- The initial size is given by the `col_default_width` and `row_default_height` properties. To customize the size of a single column or row, use `cols_minimum` or `rows_minimum`.
- The `size_hint_x/size_hint_y` of the children are taken into account. If no widgets have a size hint, the maximum size is used for all children.
- You can force the default size by setting the `col_force_default` or `row_force_default` property. This will force the layout to ignore the `width` and `size_hint` properties of children and use the default size.

33.19.3 Using a GridLayout

In the example below, all widgets will have an equal size. By default, the `size_hint` is (1, 1), so a Widget will take the full size of the parent:

```
layout = GridLayout(cols=2)
layout.add_widget(Button(text='Hello 1'))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2'))
layout.add_widget(Button(text='World 2'))
```



Now, let's fix the size of Hello buttons to 100px instead of using `size_hint_x=1`:

```
layout = GridLayout(cols=2)
layout.add_widget(Button(text='Hello 1', size_hint_x=None, width=100))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2', size_hint_x=None, width=100))
layout.add_widget(Button(text='World 2'))
```




Next, let's fix the row height to a specific size:

```
layout = GridLayout(cols=2, row_force_default=True, row_default_height=40)
layout.add_widget(Button(text='Hello 1', size_hint_x=None, width=100))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2', size_hint_x=None, width=100))
layout.add_widget(Button(text='World 2'))
```



class kivy.uix.gridlayout.**GridLayout**(**kwargs)

Bases: *kivy.uix.layout.Layout*

Grid layout class. See module documentation for more information.

col_default_width

Default minimum size to use for a column.

New in version 1.0.7.

col_default_width is a *NumericProperty* and defaults to 0.

col_force_default

If True, ignore the width and size_hint_x of the child and use the default column width.

New in version 1.0.7.

col_force_default is a *BooleanProperty* and defaults to False.

cols

Number of columns in the grid.

Changed in version 1.0.8: Changed from a NumericProperty to BoundedNumericProperty. You can no longer set this to a negative value.

cols is a *NumericProperty* and defaults to None.

cols_minimum

Dict of minimum width for each column. The dictionary keys are the column numbers, e.g. 0, 1, 2...

New in version 1.0.7.

cols_minimum is a *DictProperty* and defaults to {}.

do_layout(*largs)

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.

minimum_height

Automatically computed minimum height needed to contain all children.

New in version 1.0.8.

minimum_height is a *NumericProperty* and defaults to 0. It is read only.

minimum_size

Automatically computed minimum size needed to contain all children.

New in version 1.0.8.

minimum_size is a *ReferenceListProperty* of (*minimum_width*, *minimum_height*) properties. It is read only.

minimum_width

Automatically computed minimum width needed to contain all children.

New in version 1.0.8.

minimum_width is a *NumericProperty* and defaults to 0. It is read only.

padding

Padding between the layout box and its 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 1.7.0: Replaced *NumericProperty* with *VariableListProperty*.

padding is a *VariableListProperty* and defaults to [0, 0, 0, 0].

row_default_height

Default minimum size to use for row.

New in version 1.0.7.

row_default_height is a *NumericProperty* and defaults to 0.

row_force_default

If True, ignore the height and size_hint_y of the child and use the default row height.

New in version 1.0.7.

row_force_default is a *BooleanProperty* and defaults to False.

rows

Number of rows in the grid.

Changed in version 1.0.8: Changed from a *NumericProperty* to a *BoundedNumericProperty*. You can no longer set this to a negative value.

rows is a *NumericProperty* and defaults to None.

rows_minimum

Dict of minimum height for each row. The dictionary keys are the row numbers, e.g. 0, 1, 2...

New in version 1.0.7.

`rows_minimum` is a *DictProperty* and defaults to {}.

spacing

Spacing between children: [spacing_horizontal, spacing_vertical].

spacing also accepts a one argument form [spacing].

spacing is a *VariableListProperty* and defaults to [0, 0].

exception kivy.uix.gridlayout.GridLayoutException

Bases: Exception

Exception for errors if the grid layout manipulation fails.

33.20 Image

The *Image* widget is used to display an image:

Example in python:

```
wimg = Image(source='mylogo.png')
```

Kv Example:

```
Image:
    source: 'mylogo.png'
    size: self.texture_size
```

33.20.1 Asynchronous Loading

To load an image asynchronously (for example from an external webserver), use the *AsyncImage* subclass:

```
aimg = AsyncImage(source='http://mywebsite.com/logo.png')
```

This can be useful as it prevents your application from waiting until the image is loaded. If you want to display large images or retrieve them from URL's, using *AsyncImage* will allow these resources to be retrieved on a background thread without blocking your application.

33.20.2 Alignment

By default, the image is centered and fits inside the widget bounding box. If you don't want that, you can set *allow_stretch* to True and *keep_ratio* to False.

You can also inherit from Image and create your own style. For example, if you want your image to be greater than the size of your widget, you could do:

```
class FullImage(Image):
    pass
```

And in your kivy language file:

```
<-FullImage>:
    canvas:
        Color:
```

(continues on next page)

```

    rgb: (1, 1, 1)
Rectangle:
    texture: self.texture
    size: self.width + 20, self.height + 20
    pos: self.x - 10, self.y - 10

```

class kivy.uix.image.**Image**(**kwargs)

Bases: *kivy.uix.widget.Widget*

Image class, see module documentation for more information.

allow_stretch

If True, the normalized image size will be maximized to fit in the image box. Otherwise, if the box is too tall, the image will not be stretched more than 1:1 pixels.

New in version 1.0.7.

allow_stretch is a *BooleanProperty* and defaults to False.

anim_delay

Delay the animation if the image is sequenced (like an animated gif). If *anim_delay* is set to -1, the animation will be stopped.

New in version 1.0.8.

anim_delay is a *NumericProperty* and defaults to 0.25 (4 FPS).

anim_loop

Number of loops to play then stop animating. 0 means keep animating.

New in version 1.9.0.

anim_loop is a *NumericProperty* and defaults to 0.

color

Image color, in the format (r, g, b, a). This attribute can be used to 'tint' an image. Be careful: if the source image is not gray/white, the color will not really work as expected.

New in version 1.0.6.

color is a *ListProperty* and defaults to [1, 1, 1, 1].

image_ratio

Ratio of the image (width / float(height)).

image_ratio is an *AliasProperty* and is read-only.

keep_data

If True, the underlying *_coreimage* will store the raw image data. This is useful when performing pixel based collision detection.

New in version 1.3.0.

keep_data is a *BooleanProperty* and defaults to False.

keep_ratio

If False along with *allow_stretch* being True, the normalized image size will be maximized to fit in the image box and ignores the aspect ratio of the image. Otherwise, if the box is too tall, the image will not be stretched more than 1:1 pixels.

New in version 1.0.8.

keep_ratio is a *BooleanProperty* and defaults to True.

mipmap

Indicate if you want OpenGL mipmapping to be applied to the texture. Read [Mipmapping](#) for more information.

New in version 1.0.7.

mipmap is a *BooleanProperty* and defaults to False.

nocache

If this property is set True, the image will not be added to the internal cache. The cache will simply ignore any calls trying to append the core image.

New in version 1.6.0.

nocache is a *BooleanProperty* and defaults to False.

norm_image_size

Normalized image size within the widget box.

This size will always fit the widget size and will preserve the image ratio.

norm_image_size is an *AliasProperty* and is read-only.

reload()

Reload image from disk. This facilitates re-loading of images from disk in case the image content changes.

New in version 1.3.0.

Usage:

```
im = Image(source = '1.jpg')
# -- do something --
im.reload()
# image will be re-loaded from disk
```

source

Filename / source of your image.

source is a *StringProperty* and defaults to None.

texture

Texture object of the image. The texture represents the original, loaded image texture. It is stretched and positioned during rendering according to the *allow_stretch* and *keep_ratio* properties.

Depending of the texture creation, the value will be a *Texture* or a *TextureRegion* object.

texture is an *ObjectProperty* and defaults to None.

texture_size

Texture size of the image. This represents the original, loaded image texture size.

Warning: The texture size is set after the texture property. So if you listen to the change on *texture*, the property texture_size will not be up-to-date. Use self.texture.size instead.

class kivy.uix.image.AsyncImage(**kwargs)

Bases: *kivy.uix.image.Image*

Asynchronous Image class. See the module documentation for more information.

Note: The `AsyncImage` is a specialized form of the `Image` class. You may want to refer to the [loader](#) documentation and in particular, the [ProxyImage](#) for more detail on how to handle events around asynchronous image loading.

reload()

Reload image from disk. This facilitates re-loading of images from disk in case the image content changes.

New in version 1.3.0.

Usage:

```
im = Image(source = '1.jpg')
# -- do something --
im.reload()
# image will be re-loaded from disk
```

33.21 Label



The [Label](#) widget is for rendering text. It supports ascii and unicode strings:

```
# hello world text
l = Label(text='Hello world')

# unicode text; can only display glyphs that are available in the font
l = Label(text=u'Hello world ' + unichr(2764))

# multiline text
l = Label(text='Multi\nLine')

# size
l = Label(text='Hello world', font_size='20sp')
```

33.21.1 Sizing and text content

By default, the size of [Label](#) is not affected by [text](#) content and the text is not affected by the size. In order to control sizing, you must specify [text_size](#) to constrain the text and/or bind [size](#) to [texture_size](#) to grow with the text.

For example, this label's size will be set to the text content (plus [padding](#)):

```
Label:
    size: self.texture_size
```

This label's text will wrap at the specified width and be clipped to the height:

```
Label:
    text_size: cm(6), cm(4)
```

Note: The *shorten* and *max_lines* attributes control how overflowing text behaves.

Combine these concepts to create a Label that can grow vertically but wraps the text at a certain width:

```
Label:
    text_size: root.width, None
    size: self.texture_size
```

How to have a custom background color in the label:

```
# Define your background color Template
<BackgroundColor@Widget>
    background_color: 1, 1, 1, 1
    canvas.before:
        Color:
            rgba: root.background_color
        Rectangle:
            size: self.size
            pos: self.pos
# Now you can simply Mix the `BackgroundColor` class with almost
# any other widget... to give it a background.
<BackgroundLabel@Label+BackgroundColor>
    background_color: 0, 0, 0, 0
    # Default the background color for this label
    # to r 0, g 0, b 0, a 0
# Use the BackgroundLabel any where in your kv code like below
BackgroundLabel
    text: 'Hello'
    background_color: 1, 0, 0, 1
```

33.21.2 Text alignment and wrapping

The *Label* has *halign* and *valign* properties to control the alignment of its text. However, by default the text image (*texture*) is only just large enough to contain the characters and is positioned in the center of the Label. The *valign* property will have no effect and *halign* will only have an effect if your text has newlines; a single line of text will appear to be centered even though *halign* is set to left (by default).

In order for the alignment properties to take effect, set the *text_size*, which specifies the size of the bounding box within which text is aligned. For instance, the following code binds this size to the size of the Label, so text will be aligned within the widget bounds. This will also automatically wrap the text of the Label to remain within this area.

```
Label:
    text_size: self.size
    halign: 'right'
    valign: 'middle'
```

33.21.3 Markup text

New in version 1.1.0.

You can change the style of the text using *Text Markup*. The syntax is similar to the bbcode syntax but only the inline styling is allowed:

```
# hello world with world in bold
l = Label(text='Hello [b]World[/b]', markup=True)

# hello in red, world in blue
l = Label(text='[color=ff3333]Hello[/color][color=3333ff]World[/color]',
          markup = True)
```

If you need to escape the markup from the current text, use *kivy.utils.escape_markup()*:

```
text = 'This is an important message [1]'
l = Label(text='[b]' + escape_markup(text) + '[/b]', markup=True)
```

The following tags are available:

[b][b] Activate bold text

[i][i] Activate italic text

[u][u] Underlined text

[s][s] Strikethrough text

[font=<str>][font] Change the font (note: this refers to a TTF file or registered alias)

[font_context=<str>][font_context] Change context for the font, use string value “none” for isolated context (this is equivalent to *None*; if you created a font context named ‘none’, it cannot be referred to using markup)

[font_family=<str>][font_family] Font family to request for drawing. This is only valid when using a font context, see *kivy.uix.label.Label* for details.

[font_features=<str>][font_features] OpenType font features, in CSS format, this is passed straight through to Pango. The effects of requesting a feature depends on loaded fonts, library versions, etc. Pango only, requires v1.38 or later.

[size=<integer>][size] Change the font size

[color=#<color>][color] Change the text color

[ref=<str>][ref] Add an interactive zone. The reference + bounding box inside the reference will be available in *Label.refs*

[anchor=<str>] Put an anchor in the text. You can get the position of your anchor within the text with *Label.anchors*

[sub][sub] Display the text at a subscript position relative to the text before it.

[sup][sup] Display the text at a superscript position relative to the text before it.

[text_language=<str>][text_language] Language of the text, this is an RFC-3066 format language tag (as string), for example “en_US”, “zh_CN”, “fr” or “ja”. This can impact font selection and metrics. Use the string “None” to revert to locale detection. Pango only.

If you want to render the markup text with a [or] or & character, you need to escape them. We created a simple syntax:


```
[ -> &bl;  
] -> &br;  
& -> &amp;
```

Then you can write:

```
"[size=24>Hello &bl;World&br;[/size]"
```

33.21.4 Interactive zone in text

New in version 1.1.0.

You can now have definable “links” using text markup. The idea is to be able to detect when the user clicks on part of the text and to react. The tag `[ref=xxx]` is used for that.

In this example, we are creating a reference on the word “World”. When this word is clicked, the function `print_it` will be called with the name of the reference:

```
def print_it(instance, value):  
    print('User clicked on', value)  
widget = Label(text='Hello [ref=world]World[/ref]', markup=True)  
widget.bind(on_ref_press=print_it)
```

For prettier rendering, you could add a color for the reference. Replace the `text=` in the previous example with:

```
'Hello [ref=world][color=0000ff]World[/color][[/ref]]'
```

33.21.5 Catering for Unicode languages

The font kivy uses does not contain all the characters required for displaying all languages. When you use the built-in widgets, this results in a block being drawn where you expect a character.

If you want to display such characters, you can chose a font that supports them and deploy it universally via kv:

```
<Label>:  
    font_name: '/<path>/<to>/<font>'
```

Note that this needs to be done before your widgets are loaded as kv rules are only applied at load time.

33.21.6 Usage example

The following example marks the anchors and references contained in a label:

```
from kivy.app import App  
from kivy.uix.label import Label  
from kivy.clock import Clock  
from kivy.graphics import Color, Rectangle  
  
class TestApp(App):  
    @staticmethod  
    def get_x(label, ref_x):
```

(continues on next page)

(continued from previous page)

```
    """ Return the x value of the ref/anchor relative to the canvas """
    return label.center_x - label.texture_size[0] * 0.5 + ref_x

    @staticmethod
    def get_y(label, ref_y):
        """ Return the y value of the ref/anchor relative to the canvas """
        # Note the inversion of direction, as y values start at the top of
        # the texture and increase downwards
        return label.center_y + label.texture_size[1] * 0.5 - ref_y

    def show_marks(self, label):

        # Indicate the position of the anchors with a red top marker
        for name, anc in label.anchors.items():
            with label.canvas:
                Color(1, 0, 0)
                Rectangle(pos=(self.get_x(label, anc[0]),
                               self.get_y(label, anc[1])),
                           size=(3, 3))

        # Draw a green surround around the refs. Note the sizes y inversion
        for name, boxes in label.refs.items():
            for box in boxes:
                with label.canvas:
                    Color(0, 1, 0, 0.25)
                    Rectangle(pos=(self.get_x(label, box[0]),
                                   self.get_y(label, box[1])),
                               size=(box[2] - box[0],
                                      box[1] - box[3]))

    def build(self):
        label = Label(
            text='[anchor=a]a\nChars [anchor=b]b\n[ref=myref]ref[/ref]',
            markup=True)
        Clock.schedule_once(lambda dt: self.show_marks(label), 1)
        return label

TestApp().run()
```

class kivy.uix.label.Label(**kwargs)

Bases: *kivy.uix.widget.Widget*

Label class, see module documentation for more information.

Events

on_ref_press Fired when the user clicks on a word referenced with a [ref] tag in a text markup.

anchors

New in version 1.1.0.

Position of all the [anchor=xxx] markup in the text. These co-ordinates are relative to the top left corner of the text, with the y value increasing downwards. Anchors names should be unique and only the first occurrence of any duplicate anchors will be recorded.

You can place anchors in your markup text as follows:

```
text = """
    [anchor=title1][size=24]This is my Big title.[/size]
```

(continues on next page)

```
""" [anchor=content]Hello world
    """
```

Then, all the `[anchor=]` references will be removed and you'll get all the anchor positions in this property (only after rendering):

```
>>> widget = Label(text=text, markup=True)
>>> widget.texture_update()
>>> widget.anchors
{"content": (20, 32), "title1": (20, 16)}
```

Note: This works only with markup text. You need *markup* set to True.

base_direction

Base direction of text, this impacts horizontal alignment when *halign* is *auto* (the default). Available options are: None, "ltr" (left to right), "rtl" (right to left) plus "weak_ltr" and "weak_rtl".

Note: This feature requires the Pango text provider.

Note: Weak modes are currently not implemented in Kivy text layout, and have the same effect as setting strong mode.

New in version 1.11.0.

base_direction is an *OptionProperty* and defaults to None (autodetect RTL if possible, otherwise LTR).

bold

Indicates use of the bold version of your font.

Note: Depending of your font, the bold attribute may have no impact on your text rendering.

bold is a *BooleanProperty* and defaults to False.

color

Text color, in the format (r, g, b, a).

color is a *ListProperty* and defaults to [1, 1, 1, 1].

disabled_color

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

New in version 1.8.0.

disabled_color is a *ListProperty* and defaults to [1, 1, 1, .3].

disabled_outline_color

The color of the text outline when the widget is disabled, in the (r, g, b) format.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

disabled_outline_color is a *ListProperty* and defaults to [0, 0, 0].

ellipsis_options

Font options for the ellipsis string('...') used to split the text.

Accepts a dict as option name with the value. Only applied when *markup* is true and text is shortened. All font options which work for *Label* will work for *ellipsis_options*. Defaults for the options not specified are taken from the surrounding text.

```
Label:
    text: 'Some very long line which will be cut'
    markup: True
    shorten: True
    ellipsis_options: {'color': (1, 0.5, 0.5, 1), 'underline': True}
```

New in version 2.0.0.

ellipsis_options is a *DictProperty* and defaults to {} (the empty dict).

font_blended

Whether blended or solid font rendering should be used.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

font_blended is a *BooleanProperty* and defaults to True.

font_context

Font context. *None* means the font is used in isolation, so you are guaranteed to be drawing with the TTF file resolved by *font_name*. Specifying a value here will load the font file into a named context, enabling fallback between all fonts in the same context. If a font context is set, you are not guaranteed that rendering will actually use the specified TTF file for all glyphs (Pango will pick the one it thinks is best).

If Kivy is linked against a system-wide installation of FontConfig, you can load the system fonts by specifying a font context starting with the special string *system://*. This will load the system fontconfig configuration, and add your application-specific fonts on top of it (this imposes a significant risk of family name collision, Pango may not use your custom font file, but pick one from the system)

Note: This feature requires the Pango text provider.

New in version 1.11.0.

font_context is a *StringProperty* and defaults to None.

font_family

Font family, this is only applicable when using *font_context* option. The specified font family will be requested, but note that it may not be available, or there could be multiple fonts registered with the same family. The value can be a family name (string) available in the font context (for example a system font in a *system://* context, or a custom font file added using `kivy.core.text.FontContextManager`). If set to *None*, font selection is controlled by the *font_name* setting.

Note: If using *font_name* to reference a custom font file, you should leave this as *None*. The family name is managed automatically in this case.

Note: This feature requires the Pango text provider.

New in version 1.11.0.

font_family is a *StringProperty* and defaults to None.

font_features

OpenType font features, in CSS format, this is passed straight through to Pango. The effects of requesting a feature depends on loaded fonts, library versions, etc. For a complete list of features, see:

https://en.wikipedia.org/wiki/List_of_typographic_features

Note: This feature requires the Pango text provider, and Pango library v1.38 or later.

New in version 1.11.0.

font_features is a *StringProperty* and defaults to an empty string.

font_hinting

What hinting option to use for font rendering. Can be one of 'normal', 'light', 'mono' or None.

Note: This feature requires SDL2 or Pango text provider.

New in version 1.10.0.

font_hinting is an *OptionProperty* and defaults to 'normal'.

font_kerning

Whether kerning is enabled for font rendering. You should normally only disable this if rendering is broken with a particular font file.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

font_kerning is a *BooleanProperty* and defaults to True.

font_name

Filename of the font to use. The path can be absolute or relative. Relative paths are resolved by the *resource_find()* function.

Warning: Depending of your text provider, the font file can be ignored. However, you can mostly use this without problems.

If the font used lacks the glyphs for the particular language/symbols you are using, you will see '[]' blank box characters instead of the actual glyphs. The solution is to use a font that has the glyphs you need to display. For example, to display क, use a font such as freesans.ttf that has the glyph.

font_name is a *StringProperty* and defaults to 'Roboto'. This value is taken from *Config*.

font_size

Font size of the text, in pixels.

font_size is a *NumericProperty* and defaults to 15sp.

halign

Horizontal alignment of the text.

halign is an *OptionProperty* and defaults to 'auto'. Available options are : auto, left, center, right and justify. Auto will attempt to autodetect horizontal alignment for RTL text (Pango only), otherwise it behaves like *left*.

Warning: This doesn't change the position of the text texture of the Label (centered), only the position of the text in this texture. You probably want to bind the size of the Label to the *texture_size* or set a *text_size*.

Changed in version 1.10.1: Added *auto* option

Changed in version 1.6.0: A new option was added to *halign*, namely *justify*.

is_shortened

This property indicates if *text* was rendered with or without shortening when *shorten* is True.

New in version 1.10.0.

is_shortened is a *BooleanProperty* and defaults to False.

italic

Indicates use of the italic version of your font.

Note: Depending of your font, the italic attribute may have no impact on your text rendering.

italic is a *BooleanProperty* and defaults to False.

line_height

Line Height for the text. e.g. *line_height* = 2 will cause the spacing between lines to be twice the size.

line_height is a *NumericProperty* and defaults to 1.0.

New in version 1.5.0.

markup

New in version 1.1.0.

If True, the text will be rendered using the *MarkupLabel*: you can change the style of the text using tags. Check the *Text Markup* documentation for more information.

markup is a *BooleanProperty* and defaults to False.

max_lines

Maximum number of lines to use, defaults to 0, which means unlimited. Please note that *shorten* take over this property. (with shorten, the text is always one line.)

New in version 1.8.0.

max_lines is a *NumericProperty* and defaults to 0.

mipmap

Indicates whether OpenGL mipmapping is applied to the texture or not. Read *Mipmapping* for more information.

New in version 1.0.7.

mipmap is a *BooleanProperty* and defaults to False.

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.

outline_color

The color of the text outline, in the (r, g, b) format.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

outline_color is a *ListProperty* and defaults to [0, 0, 0].

outline_width

Width in pixels for the outline around the text. No outline will be rendered if the value is None.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

outline_width is a *NumericProperty* and defaults to None.

padding

Padding of the text in the format (padding_x, padding_y)

padding is a *ReferenceListProperty* of (*padding_x*, *padding_y*) properties.

padding_x

Horizontal padding of the text inside the widget box.

padding_x is a *NumericProperty* and defaults to 0.

Changed in version 1.9.0: *padding_x* has been fixed to work as expected. In the past, the text was padded by the negative of its values.

padding_y

Vertical padding of the text inside the widget box.

padding_y is a *NumericProperty* and defaults to 0.

Changed in version 1.9.0: *padding_y* has been fixed to work as expected. In the past, the text was padded by the negative of its values.

refs

New in version 1.1.0.

List of [ref=xxx] markup items in the text with the bounding box of all the words contained in a ref, available only after rendering.

For example, if you wrote:

Check out my [ref=hello]link[/ref]

The refs will be set with:

```
{'hello': ((64, 0, 78, 16), )}
```

The references marked “hello” have a bounding box at (x1, y1, x2, y2). These co-ordinates are relative to the top left corner of the text, with the y value increasing downwards. You can define multiple refs with the same name: each occurrence will be added as another (x1, y1, x2, y2) tuple to this list.

The current Label implementation uses these references if they exist in your markup text, automatically doing the collision with the touch and dispatching an *on_ref_press* event.

You can bind a ref event like this:

```
def print_it(instance, value):  
    print('User click on', value)  
widget = Label(text='Hello [ref=world]World[/ref]', markup=True)  
widget.on_ref_press(print_it)
```

Note: This works only with markup text. You need *markup* set to True.

shorten

Indicates whether the label should attempt to shorten its textual contents as much as possible if a *text_size* is given. Setting this to True without an appropriately set *text_size* will lead to unexpected results.

shorten_from and *split_str* control the direction from which the *text* is split, as well as where in the *text* we are allowed to split.

shorten is a *BooleanProperty* and defaults to False.

shorten_from

The side from which we should shorten the text from, can be left, right, or center.

For example, if left, the ellipsis will appear towards the left side and we will display as much text starting from the right as possible. Similar to *shorten*, this option only applies when *text_size* [0] is not None, In this case, the string is shortened to fit within the specified width.

New in version 1.9.0.

shorten_from is a *OptionProperty* and defaults to *center*.

split_str

The string used to split the *text* while shortening the string when *shorten* is True.

For example, if it's a space, the string will be broken into words and as many whole words that can fit into a single line will be displayed. If *split_str* is the empty string, "", we split on every character fitting as much text as possible into the line.

New in version 1.9.0.

split_str is a *StringProperty* and defaults to "" (the empty string).

strikethrough

Adds a strikethrough line to the text.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

strikethrough is a *BooleanProperty* and defaults to False.

strip

Whether leading and trailing spaces and newlines should be stripped from each displayed line. If True, every line will start at the right or left edge, depending on *halign*. If *halign* is *justify* it is implicitly True.

New in version 1.9.0.

strip is a *BooleanProperty* and defaults to False.

text

Text of the label.

Creation of a simple hello world:

```
widget = Label(text='Hello world')
```

If you want to create the widget with an unicode string, use:

```
widget = Label(text=u'My unicode string')
```

text is a *StringProperty* and defaults to "".

text_language

Language of the text, if None Pango will determine it from locale. This is an RFC-3066 format language tag (as a string), for example "en_US", "zh_CN", "fr" or "ja". This can impact font selection, metrics and rendering. For example, the same bytes of text can look different for *ur* and *ar* languages, though both use Arabic script.

Note: This feature requires the Pango text provider.

New in version 1.11.0.

text_language is a *StringProperty* and defaults to None.

text_size

By default, the label is not constrained to any bounding box. You can set the size constraint of the label with this property. The text will autoflow into the constraints. So although the font size will not be reduced, the text will be arranged to fit into the box as best as possible, with any text still outside the box clipped.

This sets and clips *texture_size* to text_size if not None.

New in version 1.0.4.

For example, whatever your current widget size is, if you want the label to be created in a box with width=200 and unlimited height:

```
Label(text='Very big big line', text_size=(200, None))
```

Note: This text_size property is the same as the usersize property in the Label class. (It is named size= in the constructor.)

text_size is a *ListProperty* and defaults to (None, None), meaning no size restriction by default.

texture

Texture object of the text. The text is rendered automatically when a property changes. The OpenGL texture created in this operation is stored in this property. You can use this *texture* for any graphics elements.

Depending on the texture creation, the value will be a *Texture* or *TextureRegion* object.

Warning: The *texture* update is scheduled for the next frame. If you need the texture immediately after changing a property, you have to call the *texture_update()* method before accessing *texture*:

```
l = Label(text='Hello world')
# l.texture is good
l.font_size = '50sp'
# l.texture is not updated yet
l.texture_update()
# l.texture is good now.
```

texture is an *ObjectProperty* and defaults to None.

texture_size

Texture size of the text. The size is determined by the font size and text. If *text_size* is [None, None], the texture will be the size required to fit the text, otherwise it's clipped to fit *text_size*.

When *text_size* is [None, None], one can bind to *texture_size* and rescale it proportionally to fit the size of the label in order to make the text fit maximally in the label.

Warning: The *texture_size* is set after the *texture* property. If you listen for changes to *texture*, *texture_size* will not be up-to-date in your callback. Bind to *texture_size* instead.

texture_update(*args)

Force texture recreation with the current Label properties.

After this function call, the *texture* and *texture_size* will be updated in this order.

underline

Adds an underline to the text.

Note: This feature requires the SDL2 text provider.

New in version 1.10.0.

underline is a *BooleanProperty* and defaults to False.

unicode_errors

How to handle unicode decode errors. Can be 'strict', 'replace' or 'ignore'.

New in version 1.9.0.

unicode_errors is an *OptionProperty* and defaults to 'replace'.

valign

Vertical alignment of the text.

valign is an *OptionProperty* and defaults to 'bottom'. Available options are : 'bottom', 'middle' (or 'center') and 'top'.

Changed in version 1.10.0: The 'center' option has been added as an alias of 'middle'.

Warning: This doesn't change the position of the text texture of the Label (centered), only the position of the text within this texture. You probably want to bind the size of the Label to the `texture_size` or set a `text_size` to change this behavior.

33.22 Layout

Layouts are used to calculate and assign widget positions.

The `Layout` class itself cannot be used directly. You should use one of the following layout classes:

- Anchor layout: `kivy.uix.anchorlayout.AnchorLayout`
- Box layout: `kivy.uix.boxlayout.BoxLayout`
- Float layout: `kivy.uix.floatlayout.FloatLayout`
- Grid layout: `kivy.uix.gridlayout.GridLayout`
- Page Layout: `kivy.uix.pagelayout.PageLayout`
- Relative layout: `kivy.uix.relativelayout.RelativeLayout`
- Scatter layout: `kivy.uix.scatterlayout.ScatterLayout`
- Stack layout: `kivy.uix.stacklayout.StackLayout`

33.22.1 Understanding the `size_hint` Property in `Widget`

The `size_hint` is a tuple of values used by layouts to manage the sizes of their children. It indicates the size relative to the layout's size instead of an absolute size (in pixels/points/cm/etc). The format is:

```
widget.size_hint = (width_percent, height_percent)
```

The percent is specified as a floating point number in the range 0-1. For example, 0.5 is 50%, 1 is 100%.

If you want a widget's width to be half of the parent's width and the height to be identical to the parent's height, you would do:

```
widget.size_hint = (0.5, 1.0)
```

If you don't want to use a `size_hint` for either the width or height, set the value to `None`. For example, to make a widget that is 250px wide and 30% of the parent's height, do:

```
widget.size_hint = (None, 0.3)
widget.width = 250
```

Being *Kivy properties*, these can also be set via constructor arguments:

```
widget = Widget(size_hint=(None, 0.3), width=250)
```

Changed in version 1.4.1: The `reposition_child` internal method (made public by mistake) has been removed.

class `kivy.uix.layout.Layout`(*kwargs)

Bases: `kivy.uix.widget.Widget`

Layout interface class, used to implement every layout. See module documentation for more information.

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)
```

do_layout(*largs)

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.

layout_hint_with_bounds(*sh_sum*, *available_space*, *min_bounded_size*, *sh_min_vals*, *sh_max_vals*, *hint*)

(internal) Computes the appropriate (size) hint for all the widgets given (potential) min or max bounds on the widgets' size. The hint list is updated with appropriate sizes.

It walks through the hints and for any widgets whose hint will result in violating min or max constraints, it fixes the hint. Any remaining or missing space after all the widgets are fixed get distributed to the widgets making them smaller or larger according to their size hint.

This algorithm knows nothing about the widgets other than what is passed through the input params, so it's fairly generic for laying things out according to constraints using size hints.

Parameters

sh_sum: float The sum of the size hints (basically `sum(size_hint)`).

available_space: float The amount of pixels available for all the widgets whose size hint is not None. Cannot be zero.

min_bounded_size: float The minimum amount of space required according to the `size_hint_min` of the widgets (basically `sum(size_hint_min)`).

sh_min_vals: list or iterable Items in the iterable are the `size_hint_min` for each widget. Can be None. The length should be the same as `hint`

sh_max_vals: list or iterable Items in the iterable are the `size_hint_max` for each widget. Can be None. The length should be the same as `hint`

hint: list A list whose size is the same as the length of `sh_min_vals` and `sh_min_vals` whose each element is the corresponding size hint value of that element. This list is updated in place with correct size hints that ensure the constraints are not violated.

Returns Nothing. `hint` is updated in place.

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)
```

33.23 ModalView

New in version 1.4.0.

The **ModalView** widget is used to create modal views. By default, the view will cover the whole “parent” window.

Remember that the default size of a `Widget` is `size_hint=(1, 1)`. If you don’t want your view to be fullscreen, either use size hints with values lower than 1 (for instance `size_hint=(.8, .8)`) or deactivate the `size_hint` and use fixed size attributes.

33.23.1 Examples

Example of a simple 400x400 Hello world view:

```
view = ModalView(size_hint=(None, None), size=(400, 400))
view.add_widget(Label(text='Hello world'))
```

By default, any click outside the view will dismiss it. If you don’t want that, you can set **ModalView.auto_dismiss** to `False`:

```
view = ModalView(auto_dismiss=False)
view.add_widget(Label(text='Hello world'))
view.open()
```

To manually dismiss/close the view, use the **ModalView.dismiss()** method of the `ModalView` instance:

```
view.dismiss()
```

Both **ModalView.open()** and **ModalView.dismiss()** are bindable. That means you can directly bind the function to an action, e.g. to a button’s `on_press`

```
# create content and add it to the view
content = Button(text='Close me!')
view = ModalView(auto_dismiss=False)
view.add_widget(content)
```

(continues on next page)

```
# bind the on_press event of the button to the dismiss function
content.bind(on_press=view.dismiss)

# open the view
view.open()
```

33.23.2 ModalView Events

There are two events available: *on_open* which is raised when the view is opening, and *on_dismiss* which is raised when the view is closed. For *on_dismiss*, you can prevent the view from closing by explicitly returning True from your callback.

```
def my_callback(instance):
    print('ModalView', instance, 'is being dismissed, but is prevented!')
    return True
view = ModalView()
view.add_widget(Label(text='Hello world'))
view.bind(on_dismiss=my_callback)
view.open()
```

Changed in version 1.5.0: The ModalView can be closed by hitting the escape key on the keyboard if the *ModalView.auto_dismiss* property is True (the default).

class kivy.uix.modalview.**ModalView**(**kwargs)
 Bases: *kivy.uix.anchorlayout.AnchorLayout*

ModalView class. See module documentation for more information.

Events

on_pre_open: Fired before the ModalView is opened. When this event is fired ModalView is not yet added to window.

on_open: Fired when the ModalView is opened.

on_pre_dismiss: Fired before the ModalView is closed.

on_dismiss: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

Changed in version 1.11.0: Added events *on_pre_open* and *on_pre_dismiss*.

Changed in version 2.0.0: Added property 'overlay_color'.

attach_to

If a widget is set on *attach_to*, the view will attach to the nearest parent window of the widget. If none is found, it will attach to the main/global Window.

attach_to is an *ObjectProperty* and defaults to None.

auto_dismiss

This property determines if the view is automatically dismissed when the user clicks outside it.

auto_dismiss is a *BooleanProperty* and defaults to True.

background

Background image of the view used for the view background.

background is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/modalview-background'.

background_color

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

This acts as a *multiplier* to the texture colour. The default texture is grey, so just setting the background color will give a darker result. To set a plain color, set the `background_normal` to `' '`.

The `background_color` is a *ListProperty* and defaults to [1, 1, 1, 1].

Changed in version 2.0.0: Changed behavior to affect the background of the widget itself, not the overlay dimming.

border

Border used for *BorderImage* graphics instruction. Used for the `background_normal` and the `background_down` properties. Can be used when using custom backgrounds.

It must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instructions for more information about how to use it.

`border` is a *ListProperty* and defaults to (16, 16, 16, 16).

dismiss(*largs, **kwargs)

Close the view if it is open. If you really want to close the view, whatever the `on_dismiss` event returns, you can use the *force* argument:

```
view = ModalView()
view.dismiss(force=True)
```

When the view is dismissed, it will be faded out before being removed from the parent. If you don't want animation, use:

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

on_touch_down(touch)

Receive a touch down event.

Parameters

touch: *MotionEvent* class Touch received. The touch is in parent coordinates.
See *relative layout* 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.

open(*largs, **kwargs)

Show the view window from the *attach_to* widget. If set, it will attach to the nearest window. If the widget is not attached to any window, the view will attach to the global Window.

When the view is opened, it will be faded in with an animation. If you don't want the animation, use:

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

overlay_color

Overlay color in the format (r, g, b, a). Used for dimming the window behind the modal view.

overlay_color is a *ListProperty* and defaults to [0, 0, 0, .7].

New in version 2.0.0.

33.24 PageLayout

The *PageLayout* class is used to create a simple multi-page layout, in a way that allows easy flipping from one page to another using borders.

PageLayout does not currently honor the *size_hint*, *size_hint_min*, *size_hint_max*, or *pos_hint* properties.

New in version 1.8.0.

Example:

```
PageLayout:
    Button:
        text: 'page1'
    Button:
        text: 'page2'
    Button:
        text: 'page3'
```

Transitions from one page to the next are made by swiping in from the border areas on the right or left hand side. If you wish to display multiple widgets in a page, we suggest you use a containing layout. Ideally, each page should consist of a single *layout* widget that contains the remaining widgets on that page.

class `kivy.uix.pagelayout.PageLayout`(**kwargs)

Bases: *kivy.uix.layout.Layout*

PageLayout class. See module documentation for more information.

anim_kwargs

The animation kwargs used to construct the animation

anim_kwargs is a *DictProperty* and defaults to {'d': .5, 't': 'in_quad'}.

New in version 1.11.0.

border

The width of the border around the current page used to display the previous/next page swipe areas when needed.

border is a *NumericProperty* and defaults to 50dp.

do_layout(*args)

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.

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.

page

The currently displayed page.

page is a *NumericProperty* and defaults to 0.

swipe_threshold

The threshold used to trigger swipes as percentage of the widget size.

swipe_threshold is a *NumericProperty* and defaults to .5.

33.25 Popup

New in version 1.0.7.



The *Popup* widget is used to create modal popups. By default, the popup will cover the whole “parent” window. When you are creating a popup, you must at least set a *Popup.title* and *Popup.content*.

Remember that the default size of a Widget is *size_hint*=(1, 1). If you don’t want your popup to be fullscreen, either use size hints with values less than 1 (for instance *size_hint*=(.8, .8)) or deactivate the *size_hint* and use fixed size attributes.

Changed in version 1.4.0: The *Popup* class now inherits from *ModalView*. The *Popup* offers a default layout with a title and a separation bar.

33.25.1 Examples

Example of a simple 400x400 Hello world popup:

```
popup = Popup(title='Test popup',
              content=Label(text='Hello world'),
              size_hint=(None, None), size=(400, 400))
```

By default, any click outside the popup will dismiss/close it. If you don't want that, you can set *auto_dismiss* to False:

```
popup = Popup(title='Test popup', content=Label(text='Hello world'),
              auto_dismiss=False)
popup.open()
```

To manually dismiss/close the popup, use *dismiss*:

```
popup.dismiss()
```

Both *open()* and *dismiss()* are bindable. That means you can directly bind the function to an action, e.g. to a button's *on_press*:

```
# create content and add to the popup
content = Button(text='Close me!')
popup = Popup(content=content, auto_dismiss=False)

# bind the on_press event of the button to the dismiss function
content.bind(on_press=popup.dismiss)

# open the popup
popup.open()
```

Same thing in KV language only with Factory:

```
#:import Factory kivy.factory.Factory
<MyPopup@Popup>:
    auto_dismiss: False
    Button:
        text: 'Close me!'
        on_release: root.dismiss()

    Button:
        text: 'Open popup'
        on_release: Factory.MyPopup().open()
```

Note: Popup is a special widget. Don't try to add it as a child to any other widget. If you do, Popup will be handled like an ordinary widget and won't be created hidden in the background.

```
BoxLayout:
    MyPopup: # bad!
```

33.25.2 Popup Events

There are two events available: *on_open* which is raised when the popup is opening, and *on_dismiss* which is raised when the popup is closed. For *on_dismiss*, you can prevent the popup from closing by explicitly returning True from your callback:

```
def my_callback(instance):
    print('Popup', instance, 'is being dismissed but is prevented!')
    return True
popup = Popup(content=Label(text='Hello world'))
popup.bind(on_dismiss=my_callback)
popup.open()
```

class `kivy.uix.popup.Popup` (**kwargs)

Bases: `kivy.uix.modalview.ModalView`

Popup class. See module documentation for more information.

Events

on_open: Fired when the Popup is opened.

on_dismiss: Fired when the Popup is closed. If the callback returns True, the dismiss will be canceled.

add_widget(*widget*)

Add a new widget as a child of this widget.

Parameters

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

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

New in version 1.0.5.

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

New in version 1.9.0.

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

content

Content of the popup that is displayed just under the title.

content is an *ObjectProperty* and defaults to None.

on_touch_down(*touch*)

Receive a touch down event.

Parameters

touch: MotionEvent class Touch received. The touch is in parent coordinates.

See *relative layout* 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.

separator_color

Color used by the separator between title and content.

New in version 1.1.0.

separator_color is a *ListProperty* and defaults to [47 / 255., 167 / 255., 212 / 255., 1.]

separator_height

Height of the separator.

New in version 1.1.0.

separator_height is a *NumericProperty* and defaults to 2dp.

title

String that represents the title of the popup.

title is a *StringProperty* and defaults to 'No title'.

title_align

Horizontal alignment of the title.

New in version 1.9.0.

title_align is a *OptionProperty* and defaults to 'left'. Available options are left, center, right and justify.

title_color

Color used by the Title.

New in version 1.8.0.

title_color is a *ListProperty* and defaults to [1, 1, 1, 1].

title_font

Font used to render the title text.

New in version 1.9.0.

title_font is a *StringProperty* and defaults to 'Roboto'. This value is taken from *Config*.

title_size

Represents the font size of the popup title.

New in version 1.6.0.

title_size is a *NumericProperty* and defaults to '14sp'.

exception kivy.uix.popup.**PopupException**

Bases: Exception

Popup exception, fired when multiple content widgets are added to the popup.

New in version 1.4.0.

33.26 Progress Bar

New in version 1.0.8.



The **ProgressBar** widget is used to visualize the progress of some task. Only the horizontal mode is currently supported: the vertical mode is not yet available.

The progress bar has no interactive elements and is a display-only widget.

To use it, simply assign a value to indicate the current progress:

```
from kivy.uix.progressbar import ProgressBar
pb = ProgressBar(max=1000)

# this will update the graphics automatically (75% done)
pb.value = 750
```

class kivy.uix.progressbar.**ProgressBar**(**kwargs)

Bases: **kivy.uix.widget.Widget**

Class for creating a progress bar widget.

See module documentation for more details.

max

Maximum value allowed for **value**.

max is a **NumericProperty** and defaults to 100.

value

Current value used for the slider.

value is an **AliasProperty** that returns the value of the progress bar. If the value is < 0 or > **max**, it will be normalized to those boundaries.

Changed in version 1.6.0: The value is now limited to between 0 and **max**.

value_normalized

Normalized value inside the range 0-1:

```
>>> pb = ProgressBar(value=50, max=100)
>>> pb.value
50
>>> pb.value_normalized
0.5
```

value_normalized is an **AliasProperty**.

33.27 RecycleBoxLayout

New in version 1.10.0.

Warning: This module is highly experimental, its API may change in the future and the documentation is not complete at this time.

The RecycleBoxLayout is designed to provide a **BoxLayout** type layout when used with the **RecycleView** widget. Please refer to the **recycleview** module documentation for more information.

class kivy.uix.recycleboxlayout.**RecycleBoxLayout**(**kwargs)

Bases: **kivy.uix.recyclelayout.RecycleLayout**, **kivy.uix.boxlayout.BoxLayout**

compute_visible_views(data, viewport)

viewport is in coordinates of the layout manager.

get_view_index_at(*pos*)

Return the view *index* on which position, *pos*, falls.

pos is in coordinates of the layout manager.

33.28 RecycleGridLayout

New in version 1.10.0.

Warning: This module is highly experimental, its API may change in the future and the documentation is not complete at this time.

The RecycleGridLayout is designed to provide a *GridLayout* type layout when used with the *RecycleView* widget. Please refer to the *recycleview* module documentation for more information.

class kivy.uix.recyclegridlayout.**RecycleGridLayout**(***kwargs*)

Bases: *kivy.uix.recyclelayout.RecycleLayout*, *kivy.uix.gridlayout.GridLayout*

compute_visible_views(*data*, *viewport*)

viewport is in coordinates of the layout manager.

get_view_index_at(*pos*)

Return the view *index* on which position, *pos*, falls.

pos is in coordinates of the layout manager.

33.29 RecycleLayout

New in version 1.10.0.

Warning: This module is highly experimental, its API may change in the future and the documentation is not complete at this time.

class kivy.uix.recyclelayout.**RecycleLayout**(***kwargs*)

Bases: *kivy.uix.recycleview.layout.RecycleLayoutManagerBehavior*, *kivy.uix.layout.Layout*

RecycleLayout provides the default layout for RecycleViews.

default_size

size as in w, h. They each can be None.

do_layout(**args*)

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.

refresh_view_layout(*index*, *layout*, *view*, *viewport*)

See :meth:`~kivy.uix.recycleview.views.RecycleDataAdapter.refresh_view_layout`.

set_visible_views(*indices, data, viewport*)
viewport is in coordinates of the layout manager.

33.30 Relative Layout

New in version 1.4.0.

This layout allows you to set relative coordinates for children. If you want absolute positioning, use the *FloatLayout*.

The *RelativeLayout* class behaves just like the regular *FloatLayout* except that its child widgets are positioned relative to the layout.

When a widget with position = (0,0) is added to a *RelativeLayout*, the child widget will also move when the position of the *RelativeLayout* is changed. The child widgets coordinates remain (0,0) as they are always relative to the parent layout.

33.30.1 Coordinate Systems

Window coordinates

By default, there's only one coordinate system that defines the position of widgets and touch events dispatched to them: the window coordinate system, which places (0, 0) at the bottom left corner of the window. Although there are other coordinate systems defined, e.g. local and parent coordinates, these coordinate systems are identical to the window coordinate system as long as a relative layout type widget is not in the widget's parent stack. When widget.pos is read or a touch is received, the coordinate values are in parent coordinates. But as mentioned, these are identical to window coordinates, even in complex widget stacks as long as there's no relative layout type widget in the widget's parent stack.

For example:

```
BoxLayout:
    Label:
        text: 'Left'
    Button:
        text: 'Middle'
        on_touch_down: print('Middle: {}'.format(args[1].pos))
    BoxLayout:
        on_touch_down: print('Box: {}'.format(args[1].pos))
        Button:
            text: 'Right'
            on_touch_down: print('Right: {}'.format(args[1].pos))
```

When the middle button is clicked and the touch propagates through the different parent coordinate systems, it prints the following:

```
>>> Box: (430.0, 282.0)
>>> Right: (430.0, 282.0)
>>> Middle: (430.0, 282.0)
```

As claimed, the touch has identical coordinates to the window coordinates in every coordinate system. *collide_point()* for example, takes the point in window coordinates.

Parent coordinates

Other *RelativeLayout* type widgets are *Scatter*, *ScatterLayout*, and *ScrollView*. If such a special widget is in the parent stack, only then does the parent and local coordinate system diverge

from the window coordinate system. For each such widget in the stack, a coordinate system with (0, 0) of that coordinate system being at the bottom left corner of that widget is created. **Position and touch coordinates received and read by a widget are in the coordinate system of the most recent special widget in its parent stack (not including itself) or in window coordinates if there are none** (as in the first example). We call these coordinates parent coordinates.

For example:

```
BoxLayout:
    Label:
        text: 'Left'
    Button:
        text: 'Middle'
        on_touch_down: print('Middle: {}'.format(args[1].pos))
    RelativeLayout:
        on_touch_down: print('Relative: {}'.format(args[1].pos))
        Button:
            text: 'Right'
            on_touch_down: print('Right: {}'.format(args[1].pos))
```

Clicking on the middle button prints:

```
>>> Relative: (396.0, 298.0)
>>> Right: (-137.33, 298.0)
>>> Middle: (396.0, 298.0)
```

As the touch propagates through the widgets, for each widget, the touch is received in parent coordinates. Because both the relative and middle widgets don't have these special widgets in their parent stack, the touch is the same as window coordinates. Only the right widget, which has a RelativeLayout in its parent stack, receives the touch in coordinates relative to that RelativeLayout which is different than window coordinates.

Local and Widget coordinates

When expressed in parent coordinates, the position is expressed in the coordinates of the most recent special widget in its parent stack, not including itself. When expressed in local or widget coordinates, the widgets themselves are also included.

Changing the above example to transform the parent coordinates into local coordinates:

```
BoxLayout:
    Label:
        text: 'Left'
    Button:
        text: 'Middle'
        on_touch_down: print('Middle: {}'.format(self.to_local(*args[1].pos)))
    RelativeLayout:
        on_touch_down: print('Relative: {}'.format(self.to_local(*args[1].pos)))
        Button:
            text: 'Right'
            on_touch_down: print('Right: {}'.format(self.to_local(*args[1].pos)))
```

Now, clicking on the middle button prints:

```
>>> Relative: (-135.33, 301.0)
>>> Right: (-135.33, 301.0)
>>> Middle: (398.0, 301.0)
```

This is because now the relative widget also expresses the coordinates relative to itself.

Note: Although all widgets including *RelativeLayout* receive their touch events in `on_touch_xxx` in parent coordinates, these special widgets will transform the touch position to be in local coordinates before it calls `super`. This may only be noticeable in a complex inheritance class.

Coordinate transformations

Widget provides 4 functions to transform coordinates between the various coordinate systems. For now, we assume that the *relative* keyword of these functions is *False*. *to_widget()* takes the coordinates expressed in window coordinates and returns them in local (widget) coordinates. *to_window()* takes the coordinates expressed in local coordinates and returns them in window coordinates. *to_parent()* takes the coordinates expressed in local coordinates and returns them in parent coordinates. *to_local()* takes the coordinates expressed in parent coordinates and returns them in local coordinates.

Each of the 4 transformation functions take a *relative* parameter. When the relative parameter is *True*, the coordinates are returned or originate in true relative coordinates - relative to a coordinate system with its (0, 0) at the bottom left corner of the widget in question.

33.30.2 Common Pitfalls

As all positions within a *RelativeLayout* are relative to the position of the layout itself, the position of the layout should never be used in determining the position of sub-widgets or the layout's canvas.

Take the following kv code for example:

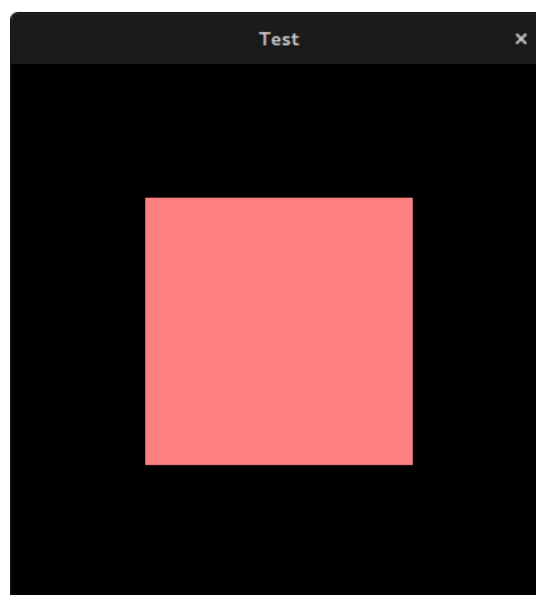


Fig. 1: expected result

```
FloatLayout:
    Widget:
        size_hint: None, None
        size: 200, 200
        pos: 200, 200

    canvas:
```

(continues on next page)

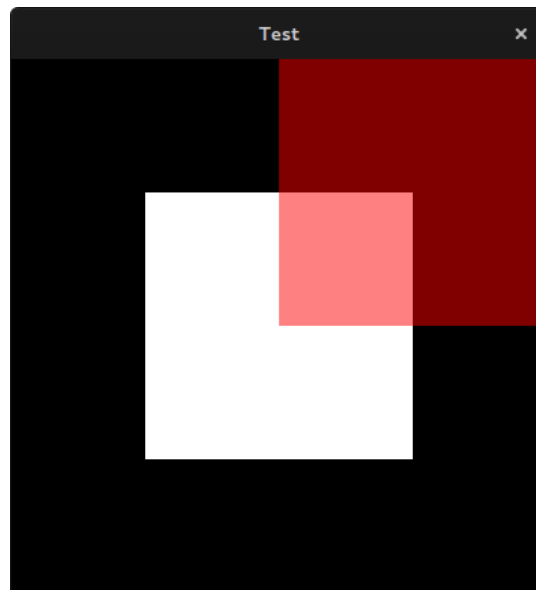


Fig. 2: actual result

(continued from previous page)

```

Color:
    rgba: 1, 1, 1, 1
Rectangle:
    pos: self.pos
    size: self.size

RelativeLayout:
    size_hint: None, None
    size: 200, 200
    pos: 200, 200

    canvas:
        Color:
            rgba: 1, 0, 0, 0.5
        Rectangle:
            pos: self.pos # incorrect
            size: self.size

```

You might expect this to render a single pink rectangle; however, the content of the *RelativeLayout* is already transformed, so the use of *pos: self.pos* will double that transformation. In this case, using *pos: 0, 0* or omitting *pos* completely will provide the expected result.

This also applies to the position of sub-widgets. Instead of positioning a *Widget* based on the layout's own position:

```

RelativeLayout:
    Widget:
        pos: self.parent.pos
    Widget:
        center: self.parent.center

```

use the *pos_hint* property:

```

RelativeLayout:
    Widget:

```

(continues on next page)

```

pos_hint: {'x': 0, 'y': 0}
Widget:
pos_hint: {'center_x': 0.5, 'center_y': 0.5}

```

Changed in version 1.7.0: Prior to version 1.7.0, the *RelativeLayout* was implemented as a *FloatLayout* inside a *Scatter*. This behaviour/widget has been renamed to *ScatterLayout*. The *RelativeLayout* now only supports relative positions (and can't be rotated, scaled or translated on a multitouch system using two or more fingers). This was done so that the implementation could be optimized and avoid the heavier calculations of *Scatter* (e.g. inverse matrix, recalculating multiple properties etc.)

class `kivy.uix.relativelayout.RelativeLayout`(**kw)

Bases: `kivy.uix.floatlayout.FloatLayout`

RelativeLayout class, see module documentation for more information.

do_layout(*args)

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.

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.

to_local(x, y, **k)

Transform parent coordinates to local (current widget) coordinates.

See *RelativeLayout* for details on the coordinate systems.

Parameters

relative: bool, defaults to False Change to True if you want to translate coordinates to relative widget coordinates.

to_parent(x, y, **k)

Transform local (current widget) coordinates to parent coordinates.

See *RelativeLayout* for details on the coordinate systems.

Parameters

relative: **bool**, defaults to **False** Change to **True** if you want to translate relative positions from a widget to its parent coordinates.

33.31 reStructuredText renderer

New in version 1.1.0.

reStructuredText is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system.

Note: This widget requires the `docutils` package to run. Install it with `pip` or include it as one of your deployment requirements.

Warning: This widget is highly experimental. The styling and implementation should not be considered stable until this warning has been removed.

33.31.1 Usage with Text

```
text = """
.. _top:

Hello world
=====

This is an emphased text, some ``interpreted text``.
And this is a reference to top_:

    $ print("Hello world")

"""
document = RstDocument(text=text)
```

The rendering will output:



33.31.2 Usage with Source

You can also render a rst file using the *source* property:

```
document = RstDocument(source='index.rst')
```

You can reference other documents using the role `:doc:`. For example, in the document `index.rst` you can write:

```
Go to my next document: :doc:`moreinfo.rst`
```

It will generate a link that, when clicked, opens the `moreinfo.rst` document.

class `kivy.uix.rst.RstDocument`(**kwargs)

Bases: `kivy.uix.scrollview.ScrollView`

Base widget used to store an Rst document. See module documentation for more information.

background_color

Specifies the `background_color` to be used for the `RstDocument`.

New in version 1.8.0.

`background_color` is an *AliasProperty* for `colors['background']`.

base_font_size

Font size for the biggest title, 31 by default. All other font sizes are derived from this.

New in version 1.8.0.

colors

Dictionary of all the colors used in the RST rendering.

Warning: This dictionary needs special handling. You also need to call `RstDocument.render()` if you change them after loading.

`colors` is a *DictProperty*.

document_root

Root path where `:doc:` will search for rst documents. If no path is given, it will use the directory of the first loaded source file.

`document_root` is a *StringProperty* and defaults to `None`.

goto(ref, *larges)

Scroll to the reference. If it's not found, nothing will be done.

For this text:

```
.. _myref:  
  
This is something I always wanted.
```

You can do:

```
from kivy.clock import Clock  
from functools import partial  
  
doc = RstDocument(...)  
Clock.schedule_once(partial(doc.goto, 'myref'), 0.1)
```

Note: It is preferable to delay the call of the `goto` if you just loaded the document because the layout might not be finished or the size of the `RstDocument` has not yet been determined. In either case, the calculation of the scrolling would be wrong.

You can, however, do a direct call if the document is already loaded.

New in version 1.3.0.

preload(*filename*, *encoding*='utf-8', *errors*='strict')

Preload a rst file to get its toctree and its title.

The result will be stored in *toctrees* with the *filename* as key.

render()

Force document rendering.

resolve_path(*filename*)

Get the path for this filename. If the filename doesn't exist, it returns the *document_root* + *filename*.

show_errors

Indicate whether RST parsers errors should be shown on the screen or not.

show_errors is a *BooleanProperty* and defaults to False.

source

Filename of the RST document.

source is a *StringProperty* and defaults to None.

source_encoding

Encoding to be used for the *source* file.

source_encoding is a *StringProperty* and defaults to *utf-8*.

Note: It is your responsibility to ensure that the value provided is a valid codec supported by python.

source_error

Error handling to be used while encoding the *source* file.

source_error is an *OptionProperty* and defaults to *strict*. Can be one of 'strict', 'ignore', 'replace', 'xmlcharrefreplace' or 'backslashreplac'.

text

RST markup text of the document.

text is a *StringProperty* and defaults to None.

title

Title of the current document.

title is a *StringProperty* and defaults to *''*. It is read-only.

toctrees

Toctree of all loaded or preloaded documents. This dictionary is filled when a rst document is explicitly loaded or where *preload()* has been called.

If the document has no filename, e.g. when the document is loaded from a text file, the key will be *''*.

toctrees is a *DictProperty* and defaults to *{}*.

underline_color

underline color of the titles, expressed in html color notation

underline_color is a *StringProperty* and defaults to '204a9699'.

33.32 Sandbox

New in version 1.8.0.

Warning: This is experimental and subject to change as long as this warning notice is present.

This is a widget that runs itself and all of its children in a Sandbox. That means if a child raises an Exception, it will be caught. The Sandbox itself runs its own Clock, Cache, etc.

The SandBox widget is still experimental and required for the Kivy designer. When the user designs their own widget, if they do something wrong (wrong size value, invalid python code), it will be caught correctly without breaking the whole application. Because it has been designed that way, we are still enhancing this widget and the [kivy.context](#) module. Don't use it unless you know what you are doing.

class kivy.uix.sandbox.Sandbox(**kwargs*)

Bases: [kivy.uix.floatlayout.FloatLayout](#)

Sandbox widget, used to trap all the exceptions raised by child widgets.

add_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)
```

clear_widgets(**args*, ***kwargs*)

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

on_context_created()

Override this method in order to load your kv file or do anything else with the newly created context.

on_exception(*exception*, _*traceback*=None)

Override this method in order to catch all the exceptions from children.

If you return True, it will not reraise the exception. If you return False, the exception will be raised to the parent.

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.

remove_widget(*args, **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)
```

33.33 Scatter

Scatter is used to build interactive widgets that can be translated, rotated and scaled with two or more fingers on a multitouch system.

Scatter has its own matrix transformation: the modelview matrix is changed before the children are drawn and the previous matrix is restored when the drawing is finished. That makes it possible to perform rotation, scaling and translation over the entire children tree without changing any widget properties. That specific behavior makes the scatter unique, but there are some advantages / constraints that you should consider:

1. The children are positioned relative to the scatter similarly to a *RelativeLayout*. So when dragging the scatter, the position of the children don't change, only the position of the scatter does.
2. The scatter size has no impact on the size of its children.
3. If you want to resize the scatter, use scale, not size (read #2). Scale transforms both the scatter and its children, but does not change size.
4. The scatter is not a layout. You must manage the size of the children yourself.

For touch events, the scatter converts from the parent matrix to the scatter matrix automatically in *on_touch_down/move/up* events. If you are doing things manually, you will need to use *to_parent()* and *to_local()*.

33.33.1 Usage

By default, the Scatter does not have a graphical representation: it is a container only. The idea is to combine the Scatter with another widget, for example an *Image*:

```
scatter = Scatter()
image = Image(source='sun.jpg')
scatter.add_widget(image)
```

33.33.2 Control Interactions

By default, all interactions are enabled. You can selectively disable them using the `do_rotation`, `do_translation` and `do_scale` properties.

Disable rotation:

```
scatter = Scatter(do_rotation=False)
```

Allow only translation:

```
scatter = Scatter(do_rotation=False, do_scale=False)
```

Allow only translation on x axis:

```
scatter = Scatter(do_rotation=False, do_scale=False,
                  do_translation_y=False)
```

33.33.3 Automatic Bring to Front

If the *Scatter.auto_bring_to_front* property is `True`, the scatter widget will be removed and re-added to the parent when it is touched (brought to front, above all other widgets in the parent). This is useful when you are manipulating several scatter widgets and don't want the active one to be partially hidden.

33.33.4 Scale Limitation

We are using a 32-bit matrix in double representation. That means we have a limit for scaling. You cannot do infinite scaling down/up with our implementation. Generally, you don't hit the minimum scale (because you don't see it on the screen), but the maximum scale is $9.99506983235e+19$ (2^{66}).

You can also limit the minimum and maximum scale allowed:

```
scatter = Scatter(scale_min=.5, scale_max=3.)
```

33.33.5 Behavior

Changed in version 1.1.0: If no control interactions are enabled, then the touch handler will never return `True`.

class `kivy.uix.scatter.Scatter`(***kwargs*)

Bases: *kivy.uix.widget.Widget*

Scatter class. See module documentation for more information.

Events

on_transform_with_touch: Fired when the scatter has been transformed by user touch or multitouch, such as panning or zooming.

on_bring_to_front: Fired when the scatter is brought to the front.

Changed in version 1.9.0: Event *on_bring_to_front* added.

Changed in version 1.8.0: Event *on_transform_with_touch* added.

apply_transform(*trans*, *post_multiply=False*, *anchor=(0, 0)*)

Transforms the scatter by applying the “trans” transformation matrix (on top of its current transformation state). The resultant matrix can be found in the *transform* property.

Parameters

trans: *Matrix*. Transformation matrix to be applied to the scatter widget.

anchor: *tuple*, defaults to **(0, 0)**. The point to use as the origin of the transformation (uses local widget space).

post_multiply: *bool*, defaults to **False**. If True, the transform matrix is post multiplied (as if applied before the current transform).

Usage example:

```
from kivy.graphics.transformation import Matrix
mat = Matrix().scale(3, 3, 3)
scatter_instance.apply_transform(mat)
```

auto_bring_to_front

If True, the widget will be automatically pushed on the top of parent widget list for drawing.

auto_bring_to_front is a *BooleanProperty* and defaults to True.

bbox

Bounding box of the widget in parent space:

```
((x, y), (w, h))
# x, y = lower left corner
```

bbox is an *AliasProperty*.

collide_point(x, y)

Check if a point (x, y) is inside the widget’s axis aligned bounding box.

Parameters

x: *numeric* x position of the point (in parent coordinates)

y: *numeric* y position of the point (in parent coordinates)

Returns A bool. True if the point is inside the bounding box, False otherwise.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

do_collide_after_children

If True, the collision detection for limiting the touch inside the scatter will be done after dispatching the touch to the children. You can put children outside the bounding box of the scatter and still be able to touch them.

do_collide_after_children is a *BooleanProperty* and defaults to False.

New in version 1.3.0.

do_rotation

Allow rotation.

do_rotation is a *BooleanProperty* and defaults to True.

do_scale

Allow scaling.

do_scale is a *BooleanProperty* and defaults to True.

do_translation

Allow translation on the X or Y axis.

do_translation is an *AliasProperty* of (*do_translation_x* + *do_translation_y*)

do_translation_x

Allow translation on the X axis.

do_translation_x is a *BooleanProperty* and defaults to True.

do_translation_y

Allow translation on Y axis.

do_translation_y is a *BooleanProperty* and defaults to True.

on_bring_to_front(*touch*)

Called when a touch event causes the scatter to be brought to the front of the parent (only if *auto_bring_to_front* is True)

Parameters

touch: The touch object which brought the scatter to front.

New in version 1.9.0.

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_transform_with_touch(*touch*)

Called when a touch event has transformed the scatter widget. By default this does nothing, but can be overridden by derived classes that need to react to transformations caused by user input.

Parameters

touch: The touch object which triggered the transformation.

New in version 1.8.0.

rotation

Rotation value of the scatter in degrees moving in a counterclockwise direction.

rotation is an *AliasProperty* and defaults to 0.0.

scale

Scale value of the scatter.

scale is an *AliasProperty* and defaults to 1.0.

scale_max

Maximum scaling factor allowed.

scale_max is a *NumericProperty* and defaults to 1e20.

scale_min

Minimum scaling factor allowed.

scale_min is a *NumericProperty* and defaults to 0.01.

to_local(*x*, *y*, ***k*)

Transform parent coordinates to local (current widget) coordinates.

See *relativelayout* for details on the coordinate systems.

Parameters

relative: **bool, defaults to False** Change to True if you want to translate coordinates to relative widget coordinates.

to_parent(*x*, *y*, ***k*)

Transform local (current widget) coordinates to parent coordinates.

See *relativelayout* for details on the coordinate systems.

Parameters

relative: **bool, defaults to False** Change to True if you want to translate relative positions from a widget to its parent coordinates.

transform

Transformation matrix.

transform is an *ObjectProperty* and defaults to the identity matrix.

Note: This matrix reflects the current state of the transformation matrix but setting it directly will erase previously applied transformations. To apply a transformation considering context, please use the *apply_transform* method.

transform_inv

Inverse of the transformation matrix.

transform_inv is an *ObjectProperty* and defaults to the identity matrix.

translation_touches

Determine whether translation was triggered by a single or multiple touches. This only has effect when *do_translation* = True.

translation_touches is a *NumericProperty* and defaults to 1.

New in version 1.7.0.

class kivy.uix.scatter.ScatterPlane(***kwargs*)

Bases: *kivy.uix.scatter.Scatter*

This is essentially an unbounded Scatter widget. It's a convenience class to make it easier to handle infinite planes.

collide_point(*x*, *y*)

Check if a point (*x*, *y*) is inside the widget's axis aligned bounding box.

Parameters

x: **numeric** x position of the point (in parent coordinates)

y: **numeric** y position of the point (in parent coordinates)

Returns A bool. True if the point is inside the bounding box, False otherwise.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

33.34 Scatter Layout

New in version 1.6.0.

This layout behaves just like a *RelativeLayout*. When a widget is added with position = (0,0) to a *ScatterLayout*, the child widget will also move when you change the position of the *ScatterLayout*. The child widget's coordinates remain (0,0) as they are relative to the parent layout.

However, since *ScatterLayout* is implemented using a *Scatter* widget, you can also translate, rotate and scale the layout using touches or clicks, just like in the case of a normal Scatter widget, and the child widgets will behave as expected.

In contrast to a Scatter, the Layout favours 'hint' properties, such as *size_hint*, *size_hint_x*, *size_hint_y* and *pos_hint*.

Note: The *ScatterLayout* is implemented as a *FloatLayout* inside a *Scatter*.

Warning: Since the actual *ScatterLayout* is a *Scatter*, its *add_widget* and *remove_widget* functions are overridden to add children to the embedded *FloatLayout* (accessible as the *content* property of *Scatter*) automatically. So if you want to access the added child elements, you need *self.content.children* instead of *self.children*.

Warning: The *ScatterLayout* was introduced in 1.7.0 and was called *RelativeLayout* in prior versions. The *RelativeLayout* is now an optimized implementation that uses only a positional transform to avoid some of the heavier calculation involved for *Scatter*.

class kivy.uix.scatterlayout.**ScatterLayout**(***kw*)

Bases: *kivy.uix.scatter.Scatter*

ScatterLayout class, see module documentation for more information.

add_widget(**l*)

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)
```

clear_widgets()

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

remove_widget(*l)

Remove a widget from the children of this widget.

Parameters

widget: Widget Widget to remove from our children list.

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

class kivy.uix.scatterlayout.ScatterPlaneLayout(kwargs)**

Bases: *kivy.uix.scatter.ScatterPlane*

ScatterPlaneLayout class, see module documentation for more information.

Similar to ScatterLayout, but based on ScatterPlane - so the input is not bounded.

New in version 1.9.0.

collide_point(x, y)

Check if a point (x, y) is inside the widget's axis aligned bounding box.

Parameters

x: numeric x position of the point (in parent coordinates)

y: numeric y position of the point (in parent coordinates)

Returns A bool. True if the point is inside the bounding box, False otherwise.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

33.35 Screen Manager

New in version 1.4.0.

The screen manager is a widget dedicated to managing multiple screens for your application. The default *ScreenManager* displays only one *Screen* at a time and uses a *TransitionBase* to switch from one Screen to another.

Multiple transitions are supported based on changing the screen coordinates / scale or even performing fancy animation using custom shaders.

33.35.1 Basic Usage

Let's construct a Screen Manager with 4 named screens. When you are creating a screen, **you absolutely need to give a name to it**:

```
from kivy.uix.screenmanager import ScreenManager, Screen

# Create the manager
sm = ScreenManager()

# Add few screens
for i in range(4):
    screen = Screen(name='Title %d' % i)
    sm.add_widget(screen)

# By default, the first screen added into the ScreenManager will be
# displayed. You can then change to another screen.

# Let's display the screen named 'Title 2'
# A transition will automatically be used.
sm.current = 'Title 2'
```

The default *ScreenManager.transition* is a *SlideTransition* with options *direction* and *duration*.

Please note that by default, a *Screen* displays nothing: it's just a *RelativeLayout*. You need to use that class as a root widget for your own screen, the best way being to subclass.

Warning: As *Screen* is a *RelativeLayout*, it is important to understand the *Common Pitfalls*.

Here is an example with a 'Menu Screen' and a 'Settings Screen':

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen

# Create both screens. Please note the root.manager.current: this is how
# you can control the ScreenManager from kv. Each screen has by default a
# property manager that gives you the instance of the ScreenManager used.
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'Goto settings'
```

(continues on next page)

(continued from previous page)

```
        on_press: root.manager.current = 'settings'
    Button:
        text: 'Quit'

<SettingsScreen>:
    BoxLayout:
        Button:
            text: 'My settings button'
        Button:
            text: 'Back to menu'
            on_press: root.manager.current = 'menu'
    """

# Declare both screens
class MenuScreen(Screen):
    pass

class SettingsScreen(Screen):
    pass

# Create the screen manager
sm = ScreenManager()
sm.add_widget(MenuScreen(name='menu'))
sm.add_widget(SettingsScreen(name='settings'))

class TestApp(App):

    def build(self):
        return sm

if __name__ == '__main__':
    TestApp().run()
```

33.35.2 Changing Direction

A common use case for *ScreenManager* involves using a *SlideTransition* which slides right to the next screen and slides left to the previous screen. Building on the previous example, this can be accomplished like so:

```
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'Goto settings'
            on_press:
                root.manager.transition.direction = 'left'
                root.manager.current = 'settings'
        Button:
            text: 'Quit'

<SettingsScreen>:
    BoxLayout:
        Button:
            text: 'My settings button'
        Button:
            text: 'Back to menu'
    """)
```

(continues on next page)


```

        on_press:
            root.manager.transition.direction = 'right'
            root.manager.current = 'menu'
    """

```

33.35.3 Advanced Usage

From 1.8.0, you can now switch dynamically to a new screen, change the transition options and remove the previous one by using `switch_to()`:

```

sm = ScreenManager()
screens = [Screen(name='Title {}'.format(i)) for i in range(4)]

sm.switch_to(screens[0])
# later
sm.switch_to(screens[1], direction='right')

```

Note that this method adds the screen to the `ScreenManager` instance and should not be used if your screens have already been added to this instance. To switch to a screen which is already added, you should use the `current` property.

33.35.4 Changing transitions

You have multiple transitions available by default, such as:

- `NoTransition` - switches screens instantly with no animation
- `SlideTransition` - slide the screen in/out, from any direction
- `CardTransition` - new screen slides on the previous or the old one slides off the new one depending on the mode
- `SwapTransition` - implementation of the iOS swap transition
- `FadeTransition` - shader to fade the screen in/out
- `WipeTransition` - shader to wipe the screens from right to left
- `FallOutTransition` - shader where the old screen 'falls' and becomes transparent, revealing the new one behind it.
- `RiseInTransition` - shader where the new screen rises from the screen centre while fading from transparent to opaque.

You can easily switch transitions by changing the `ScreenManager.transition` property:

```

sm = ScreenManager(transition=FadeTransition())

```

Note: Currently, none of Shader based Transitions use anti-aliasing. This is because they use the FBO which doesn't have any logic to handle supersampling. This is a known issue and we are working on a transparent implementation that will give the same results as if it had been rendered on screen.

To be more concrete, if you see sharp edged text during the animation, it's normal.

```

class kivy.uix.screenmanager.Screen(**kw)
    Bases: kivy.uix.relativelayout.RelativeLayout

```

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

Events

on_pre_enter: () Event fired when the screen is about to be used: the entering animation is started.

on_enter: () Event fired when the screen is displayed: the entering animation is complete.

on_pre_leave: () Event fired when the screen is about to be removed: the leaving animation is started.

on_leave: () Event fired when the screen is removed: the leaving animation is finished.

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

manager

ScreenManager object, set when the screen is added to a manager.

manager is an *ObjectProperty* and defaults to None, read-only.

name

Name of the screen which must be unique within a *ScreenManager*. This is the name used for *ScreenManager.current*.

name is a *StringProperty* and defaults to "".

transition_progress

Value that represents the completion of the current transition, if any is occurring.

If a transition is in progress, whatever the mode, the value will change from 0 to 1. If you want to know if it's an entering or leaving animation, check the *transition_state*.

transition_progress is a *NumericProperty* and defaults to 0.

transition_state

Value that represents the state of the transition:

- 'in' if the transition is going to show your screen
- 'out' if the transition is going to hide your screen

After the transition is complete, the state will retain its last value (in or out).

transition_state is an *OptionProperty* and defaults to 'out'.

class kivy.uix.screenmanager.**ScreenManager**(*kwargs)

Bases: *kivy.uix.floatlayout.FloatLayout*

Screen manager. This is the main class that will control your *Screen* stack and memory.

By default, the manager will show only one screen at a time.

add_widget(screen)

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)
```

clear_widgets(*screens=None*)

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

current

Name of the screen currently shown, or the screen to show.

```
from kivy.uix.screenmanager import ScreenManager, Screen

sm = ScreenManager()
sm.add_widget(Screen(name='first'))
sm.add_widget(Screen(name='second'))

# By default, the first added screen will be shown. If you want to
# show another one, just set the 'current' property.
sm.current = 'second'
```

current is a *StringProperty* and defaults to *None*.

current_screen

Contains the currently displayed screen. You must not change this property manually, use *current* instead.

current_screen is an *ObjectProperty* and defaults to *None*, read-only.

get_screen(*name*)

Return the screen widget associated with the name or raise a *ScreenManagerException* if not found.

has_screen(*name*)

Return True if a screen with the *name* has been found.

New in version 1.6.0.

next()

Return the name of the next screen from the screen list.

on_touch_down(*touch*)

Receive a touch down event.

Parameters

touch: *MotionEvent* class Touch received. The touch is in parent coordinates. See *relative layout* 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.

previous()

Return the name of the previous screen from the screen list.

remove_widget(*l)

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)
```

screen_names

List of the names of all the *Screen* widgets added. The list is read only.

screens_names is an *AliasProperty* and is read-only. It is updated if the screen list changes or the name of a screen changes.

screens

List of all the *Screen* widgets added. You should not change this list manually. Use the *add_widget* method instead.

screens is a *ListProperty* and defaults to [], read-only.

switch_to(*screen*, *options*)**

Add a new or existing screen to the ScreenManager and switch to it. The previous screen will be “switched away” from. *options* are the *transition* options that will be changed before the animation happens.

If no previous screens are available, the screen will be used as the main one:

```
sm = ScreenManager()
sm.switch_to(screen1)
# later
sm.switch_to(screen2, direction='left')
# later
sm.switch_to(screen3, direction='right', duration=1.)
```

If any animation is in progress, it will be stopped and replaced by this one: you should avoid this because the animation will just look weird. Use either *switch_to()* or *current* but not both.

The *screen* name will be changed if there is any conflict with the current screen.

transition

Transition object to use for animating the transition from the current screen to the next one being shown.

For example, if you want to use a *WipeTransition* between slides:

```

from kivy.uix.screenmanager import ScreenManager, Screen,
WipeTransition

sm = ScreenManager(transition=WipeTransition())
sm.add_widget(Screen(name='first'))
sm.add_widget(Screen(name='second'))

# by default, the first added screen will be shown. If you want to
# show another one, just set the 'current' property.
sm.current = 'second'

```

transition is an *ObjectProperty* and defaults to a *SlideTransition*.

Changed in version 1.8.0: Default transition has been changed from *SwapTransition* to *SlideTransition*.

exception kivy.uix.screenmanager.ScreenManagerException

Bases: Exception

Exception for the *ScreenManager*.

class kivy.uix.screenmanager.TransitionBase

Bases: *kivy.event.EventDispatcher*

TransitionBase is used to animate 2 screens within the *ScreenManager*. This class acts as a base for other implementations like the *SlideTransition* and *SwapTransition*.

Events

on_progress: Transition object, progression float Fired during the animation of the transition.

on_complete: Transition object Fired when the transition is finished.

add_screen(screen)

(internal) Used to add a screen to the *ScreenManager*.

duration

Duration in seconds of the transition.

duration is a *NumericProperty* and defaults to .4 (= 400ms).

Changed in version 1.8.0: Default duration has been changed from 700ms to 400ms.

is_active

Indicate whether the transition is currently active or not.

is_active is a *BooleanProperty* and defaults to False, read-only.

manager

ScreenManager object, set when the screen is added to a manager.

manager is an *ObjectProperty* and defaults to None, read-only.

remove_screen(screen)

(internal) Used to remove a screen from the *ScreenManager*.

screen_in

Property that contains the screen to show. Automatically set by the *ScreenManager*.

screen_in is an *ObjectProperty* and defaults to None.

screen_out

Property that contains the screen to hide. Automatically set by the *ScreenManager*.

screen_out is an *ObjectProperty* and defaults to None.

start(*manager*)

(internal) Starts the transition. This is automatically called by the *ScreenManager*.

stop()

(internal) Stops the transition. This is automatically called by the *ScreenManager*.

class `kivy.uix.screenmanager.ShaderTransition`

Bases: *kivy.uix.screenmanager.TransitionBase*

Transition class that uses a Shader for animating the transition between 2 screens. By default, this class doesn't assign any fragment/vertex shader. If you want to create your own fragment shader for the transition, you need to declare the header yourself and include the "t", "tex_in" and "tex_out" uniform:

```
# Create your own transition. This shader implements a "fading"
# transition.
fs = """$HEADER
    uniform float t;
    uniform sampler2D tex_in;
    uniform sampler2D tex_out;

    void main(void) {
        vec4 cin = texture2D(tex_in, tex_coord0);
        vec4 cout = texture2D(tex_out, tex_coord0);
        gl_FragColor = mix(cout, cin, t);
    }
"""

# And create your transition
tr = ShaderTransition(fs=fs)
sm = ScreenManager(transition=tr)
```

add_screen(*screen*)

(internal) Used to add a screen to the *ScreenManager*.

clearcolor

Sets the color of Fbo ClearColor.

New in version 1.9.0.

clearcolor is a *ListProperty* and defaults to [0, 0, 0, 1].

fs

Fragment shader to use.

fs is a *StringProperty* and defaults to None.

remove_screen(*screen*)

(internal) Used to remove a screen from the *ScreenManager*.

stop()

(internal) Stops the transition. This is automatically called by the *ScreenManager*.

vs

Vertex shader to use.

vs is a *StringProperty* and defaults to None.

class `kivy.uix.screenmanager.SlideTransition`

Bases: *kivy.uix.screenmanager.TransitionBase*

Slide Transition, can be used to show a new screen from any direction: left, right, up or down.

direction

Direction of the transition.

direction is an *OptionProperty* and defaults to 'left'. Can be one of 'left', 'right', 'up' or 'down'.

class `kivy.uix.screenmanager.SwapTransition(**kwargs)`

Bases: *kivy.uix.screenmanager.TransitionBase*

Swap transition that looks like iOS transition when a new window appears on the screen.

add_screen(*screen*)

(internal) Used to add a screen to the *ScreenManager*.

start(*manager*)

(internal) Starts the transition. This is automatically called by the *ScreenManager*.

class `kivy.uix.screenmanager.FadeTransition`

Bases: *kivy.uix.screenmanager.ShaderTransition*

Fade transition, based on a fragment Shader.

class `kivy.uix.screenmanager.WipeTransition`

Bases: *kivy.uix.screenmanager.ShaderTransition*

Wipe transition, based on a fragment Shader.

class `kivy.uix.screenmanager.FallOutTransition`

Bases: *kivy.uix.screenmanager.ShaderTransition*

Transition where the new screen 'falls' from the screen centre, becoming smaller and more transparent until it disappears, and revealing the new screen behind it. Mimics the popular/standard Android transition.

New in version 1.8.0.

duration

Duration in seconds of the transition, replacing the default of *TransitionBase*.

duration is a *NumericProperty* and defaults to .15 (= 150ms).

class `kivy.uix.screenmanager.RiseInTransition`

Bases: *kivy.uix.screenmanager.ShaderTransition*

Transition where the new screen rises from the screen centre, becoming larger and changing from transparent to opaque until it fills the screen. Mimics the popular/standard Android transition.

New in version 1.8.0.

duration

Duration in seconds of the transition, replacing the default of *TransitionBase*.

duration is a *NumericProperty* and defaults to .2 (= 200ms).

class `kivy.uix.screenmanager.NoTransition`

Bases: *kivy.uix.screenmanager.TransitionBase*

No transition, instantly switches to the next screen with no delay or animation.

New in version 1.8.0.

class `kivy.uix.screenmanager.CardTransition`

Bases: *kivy.uix.screenmanager.SlideTransition*

Card transition that looks similar to Android 4.x application drawer interface animation.

It supports 4 directions like *SlideTransition*: left, right, up and down, and two modes, pop and push. If push mode is activated, the previous screen does not move, and the new one slides in from the given direction. If the pop mode is activated, the previous screen slides out, when the new screen is already on the position of the *ScreenManager*.

New in version 1.10.

mode

Indicates if the transition should push or pop the screen on/off the ScreenManager.

- 'push' means the screen slides in in the given direction
- 'pop' means the screen slides out in the given direction

mode is an *OptionProperty* and defaults to 'push'.

start(*manager*)

(internal) Starts the transition. This is automatically called by the *ScreenManager*.

33.36 ScrollView

New in version 1.0.4.

The *ScrollView* widget provides a scrollable/pannable viewport that is clipped at the scrollview's bounding box.

33.36.1 Scrolling Behavior

The ScrollView accepts only one child and applies a viewport/window to it according to the *scroll_x* and *scroll_y* properties. Touches are analyzed to determine if the user wants to scroll or control the child in some other manner: you cannot do both at the same time. To determine if interaction is a scrolling gesture, these properties are used:

- *scroll_distance*: the minimum distance to travel, defaults to 20 pixels.
- *scroll_timeout*: the maximum time period, defaults to 55 milliseconds.

If a touch travels *scroll_distance* pixels within the *scroll_timeout* period, it is recognized as a scrolling gesture and translation (scroll/pan) will begin. If the timeout occurs, the touch down event is dispatched to the child instead (no translation).

The default value for those settings can be changed in the configuration file:

```
[widgets]
scroll_timeout = 250
scroll_distance = 20
```

New in version 1.1.1: ScrollView now animates scrolling in Y when a mousewheel is used.

33.36.2 Limiting to the X or Y Axis

By default, the ScrollView allows scrolling along both the X and Y axes. You can explicitly disable scrolling on an axis by setting the *do_scroll_x* or *do_scroll_y* properties to False.

33.36.3 Managing the Content Size and Position

The ScrollView manages the position of its children similarly to a *RelativeLayout* but does not use the *size_hint*. You must carefully specify the *size* of your content to get the desired scroll/pan effect.

By default, the *size_hint* is (1, 1), so the content size will fit your ScrollView exactly (you will have nothing to scroll). You must deactivate at least one of the *size_hint* instructions (x or y) of the child to

enable scrolling. Setting `size_hint_min` to not be `None` will also enable scrolling for that dimension when the `ScrollView` is smaller than the minimum size.

To scroll a `GridLayout` on it's Y-axis/vertically, set the child's width to that of the `ScrollView` (`size_hint_x=1`), and set the `size_hint_y` property to `None`:

```
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.scrollview import ScrollView
from kivy.core.window import Window
from kivy.app import runTouchApp

layout = GridLayout(cols=1, spacing=10, size_hint_y=None)
# Make sure the height is such that there is something to scroll.
layout.bind(minimum_height=layout.setter('height'))
for i in range(100):
    btn = Button(text=str(i), size_hint_y=None, height=40)
    layout.add_widget(btn)
root = ScrollView(size_hint=(1, None), size=(Window.width, Window.height))
root.add_widget(layout)

runTouchApp(root)
```

Kv Example:

```
ScrollView:
    do_scroll_x: False
    do_scroll_y: True

    Label:
        size_hint_y: None
        height: self.texture_size[1]
        text_size: self.width, None
        padding: 10, 10
        text:
            'really some amazing text\n' * 100
```

33.36.4 Overscroll Effects

New in version 1.7.0.

When scrolling would exceed the bounds of the `ScrollView`, it uses a `ScrollEffect` to handle the overscroll. These effects can perform actions like bouncing back, changing opacity, or simply preventing scrolling beyond the normal boundaries. Note that complex effects may perform many computations, which can be slow on weaker hardware.

You can change what effect is being used by setting `effect_cls` to any effect class. Current options include:

- `ScrollEffect`: Does not allow scrolling beyond the `ScrollView` boundaries.
- `DampedScrollEffect`: The current default. Allows the user to scroll beyond the normal boundaries, but has the content spring back once the touch/click is released.
- `OpacityScrollEffect`: Similar to the `DampedScrollEffect`, but also reduces opacity during overscroll.

You can also create your own scroll effect by subclassing one of these, then pass it as the `effect_cls` in the same way.

Alternatively, you can set *effect_x* and/or *effect_y* to an *instance* of the effect you want to use. This will override the default effect set in *effect_cls*.

All the effects are located in the *kivy.effects*.

class *kivy.uix.scrollview.ScrollView*(**kwargs)

Bases: *kivy.uix.stencilview.StencilView*

ScrollView class. See module documentation for more information.

Events

on_scroll_start Generic event fired when scrolling starts from touch.

on_scroll_move Generic event fired when scrolling move from touch.

on_scroll_stop Generic event fired when scrolling stops from touch.

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

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

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)
```

bar_color

Color of horizontal / vertical scroll bar, in RGBA format.

New in version 1.2.0.

bar_color is a *ListProperty* and defaults to [.7, .7, .7, .9].

bar_inactive_color

Color of horizontal / vertical scroll bar (in RGBA format), when no scroll is happening.

New in version 1.9.0.

bar_inactive_color is a *ListProperty* and defaults to [.7, .7, .7, .2].

bar_margin

Margin between the bottom / right side of the scrollview when drawing the horizontal / vertical scroll bar.

New in version 1.2.0.

bar_margin is a *NumericProperty*, default to 0

bar_pos

Which side of the scroll view to place each of the bars on.

bar_pos is a *ReferenceListProperty* of (*bar_pos_x*, *bar_pos_y*)

bar_pos_x

Which side of the ScrollView the horizontal scroll bar should go on. Possible values are 'top' and 'bottom'.

New in version 1.8.0.

bar_pos_x is an *OptionProperty*, defaults to 'bottom'.

bar_pos_y

Which side of the ScrollView the vertical scroll bar should go on. Possible values are 'left' and 'right'.

New in version 1.8.0.

bar_pos_y is an *OptionProperty* and defaults to 'right'.

bar_width

Width of the horizontal / vertical scroll bar. The width is interpreted as a height for the horizontal bar.

New in version 1.2.0.

bar_width is a *NumericProperty* and defaults to 2.

convert_distance_to_scroll(dx, dy)

Convert a distance in pixels to a scroll distance, depending on the content size and the scrollview size.

The result will be a tuple of scroll distance that can be added to *scroll_x* and *scroll_y*

do_scroll

Allow scroll on X or Y axis.

do_scroll is a *AliasProperty* of (*do_scroll_x* + *do_scroll_y*)

do_scroll_x

Allow scroll on X axis.

do_scroll_x is a *BooleanProperty* and defaults to True.

do_scroll_y

Allow scroll on Y axis.

do_scroll_y is a *BooleanProperty* and defaults to True.

effect_cls

Class effect to instantiate for X and Y axis.

New in version 1.7.0.

effect_cls is an *ObjectProperty* and defaults to DampedScrollEffect.

Changed in version 1.8.0: If you set a string, the *Factory* will be used to resolve the class.

effect_x

Effect to apply for the X axis. If None is set, an instance of *effect_cls* will be created.

New in version 1.7.0.

effect_x is an *ObjectProperty* and defaults to None.

effect_y

Effect to apply for the Y axis. If None is set, an instance of *effect_cls* will be created.

New in version 1.7.0.

effect_y is an *ObjectProperty* and defaults to None, read-only.

hbar

Return a tuple of (position, size) of the horizontal scrolling bar.

New in version 1.2.0.

The position and size are normalized between 0-1, and represent a percentage of the current scrollview height. This property is used internally for drawing the little horizontal bar when you're scrolling.

vbar is a *AliasProperty*, readonly.

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.

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)
```

scroll_distance

Distance to move before scrolling the *ScrollView*, in pixels. As soon as the distance has been traveled, the *ScrollView* will start to scroll, and no touch event will go to children. It is advisable that you base this value on the dpi of your target device's screen.

scroll_distance is a *NumericProperty* and defaults to 20 (pixels), according to the default value in user configuration.

scroll_timeout

Timeout allowed to trigger the *scroll_distance*, in milliseconds. If the user has not moved *scroll_distance* within the timeout, the scrolling will be disabled, and the touch event will go to the children.

scroll_timeout is a *NumericProperty* and defaults to 55 (milliseconds) according to the default value in user configuration.

Changed in version 1.5.0: Default value changed from 250 to 55.

scroll_to(*widget*, *padding=10*, *animate=True*)

Scrolls the viewport to ensure that the given widget is visible, optionally with padding and animation. If *animate* is True (the default), then the default animation parameters will be used. Otherwise, it should be a dict containing arguments to pass to *Animation* constructor.

New in version 1.9.1.

scroll_type

Sets the type of scrolling to use for the content of the scrollview. Available options are: ['content'], ['bars'], ['bars', 'content'].

['content']	Content is scrolled by dragging or swiping the content directly.
['bars']	Content is scrolled by dragging or swiping the scroll bars.
['bars', 'content']	Content is scrolled by either of the above methods.

New in version 1.8.0.

scroll_type is an *OptionProperty* and defaults to ['content'].

scroll_wheel_distance

Distance to move when scrolling with a mouse wheel. It is advisable that you base this value on the dpi of your target device's screen.

New in version 1.8.0.

scroll_wheel_distance is a *NumericProperty*, defaults to 20 pixels.

scroll_x

X scrolling value, between 0 and 1. If 0, the content's left side will touch the left side of the ScrollView. If 1, the content's right side will touch the right side.

This property is controlled by *ScrollView* only if *do_scroll_x* is True.

scroll_x is a *NumericProperty* and defaults to 0.

scroll_y

Y scrolling value, between 0 and 1. If 0, the content's bottom side will touch the bottom side of the ScrollView. If 1, the content's top side will touch the top side.

This property is controlled by *ScrollView* only if *do_scroll_y* is True.

scroll_y is a *NumericProperty* and defaults to 1.

smooth_scroll_end

Whether smooth scroll end should be used when scrolling with the mouse-wheel and the factor of transforming the scroll distance to velocity. This option also enables velocity addition meaning if you scroll more, you will scroll faster and further. The recommended value is 10. The velocity is calculated as *scroll_wheel_distance* * *smooth_scroll_end*.

New in version 1.11.0.

smooth_scroll_end is a *NumericProperty* and defaults to None.

to_local(*x*, *y*, ***k*)

Transform parent coordinates to local (current widget) coordinates.

See *relativelayout* for details on the coordinate systems.

Parameters

relative: bool, defaults to False Change to True if you want to translate coordinates to relative widget coordinates.

to_parent(*x, y, **k*)

Transform local (current widget) coordinates to parent coordinates.

See *relative layout* for details on the coordinate systems.

Parameters

relative: bool, defaults to False Change to True if you want to translate relative positions from a widget to its parent coordinates.

update_from_scroll(**largs*)

Force the reposition of the content, according to current value of *scroll_x* and *scroll_y*.

This method is automatically called when one of the *scroll_x*, *scroll_y*, *pos* or *size* properties change, or if the size of the content changes.

vbar

Return a tuple of (position, size) of the vertical scrolling bar.

New in version 1.2.0.

The position and size are normalized between 0-1, and represent a percentage of the current scrollview height. This property is used internally for drawing the little vertical bar when you're scrolling.

vbar is a *AliasProperty*, readonly.

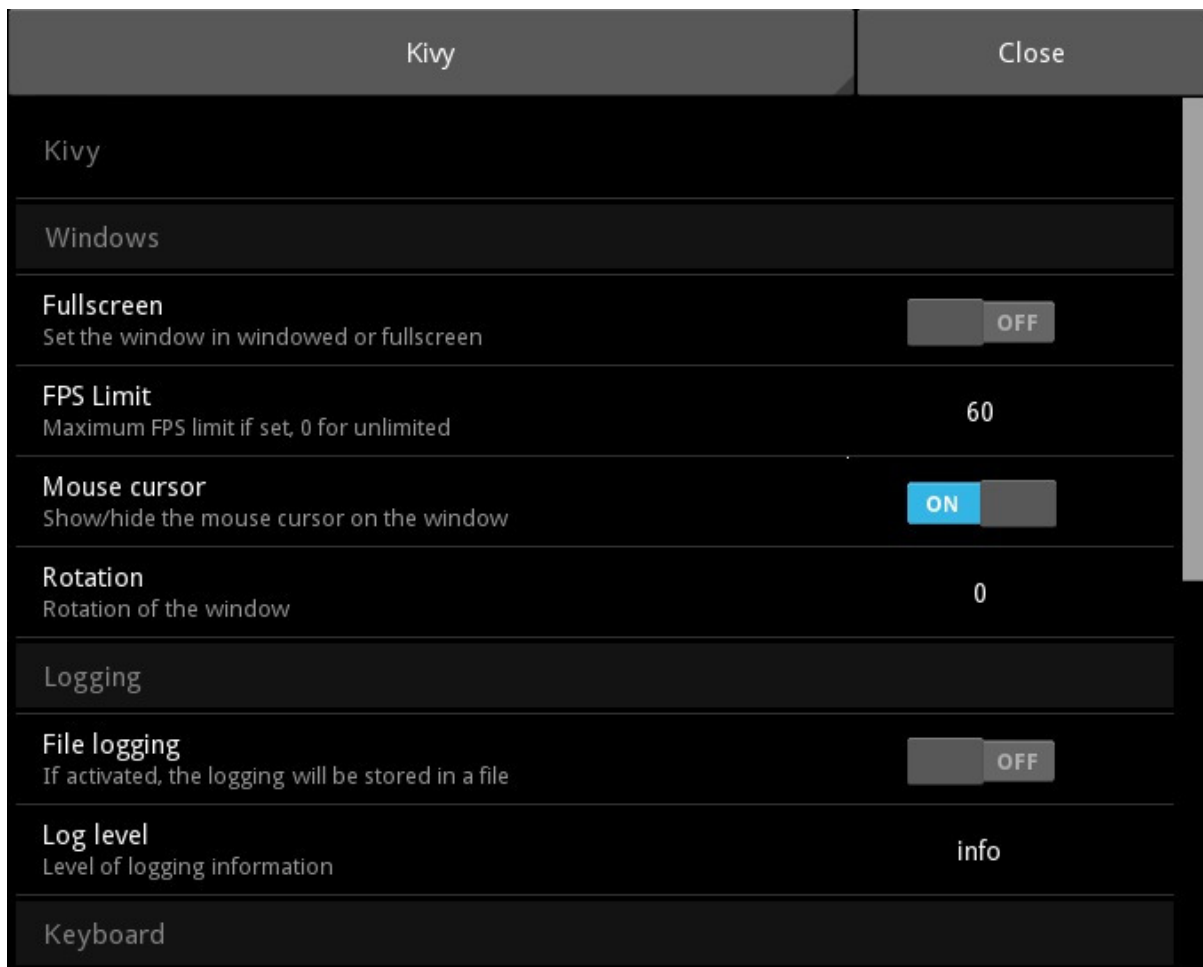
viewport_size

(internal) Size of the internal viewport. This is the size of your only child in the scrollview.

33.37 Settings

New in version 1.0.7.

This module provides a complete and extensible framework for adding a Settings interface to your application. By default, the interface uses a *SettingsWithSpinner*, which consists of a *Spinner* (top) to switch between individual settings panels (bottom). See *Different panel layouts* for some alternatives.



A *SettingsPanel* represents a group of configurable options. The *SettingsPanel.title* property is used by *Settings* when a panel is added: it determines the name of the sidebar button. *SettingsPanel* controls a *ConfigParser* instance.

The panel can be automatically constructed from a JSON definition file: you describe the settings you want and corresponding sections/keys in the *ConfigParser* instance... and you're done!

Settings are also integrated into the *App* class. Use *Settings.add_kivy_panel()* to configure the Kivy core settings in a panel.

33.37.1 Create a panel from JSON

To create a panel from a JSON-file, you need two things:

- a *ConfigParser* instance with default values
- a JSON file

Warning: The *kivy.config.ConfigParser* is required. You cannot use the default *ConfigParser* from Python libraries.

You must create and handle the *ConfigParser* object. *SettingsPanel* will read the values from the associated *ConfigParser* instance. Make sure you have set default values (using *setdefaults*) for all the sections/keys in your JSON file!

The JSON file contains structured information to describe the available settings. Here is an example:

```
[
  {
    "type": "title",
    "title": "Windows"
  },
  {
    "type": "bool",
    "title": "Fullscreen",
    "desc": "Set the window in windowed or fullscreen",
    "section": "graphics",
    "key": "fullscreen"
  }
]
```

Each element in the root list represents a setting that the user can configure. Only the “type” key is mandatory: an instance of the associated class will be created and used for the setting - other keys are assigned to corresponding properties of that class.

Type	Associated class
title	<i>SettingTitle</i>
bool	<i>SettingBoolean</i>
numeric	<i>SettingNumeric</i>
options	<i>SettingOptions</i>
string	<i>SettingString</i>
path	<i>SettingPath</i>

New in version 1.1.0: Added *SettingPath* type

In the JSON example above, the first element is of type “title”. It will create a new instance of *SettingTitle* and apply the rest of the key-value pairs to the properties of that class, i.e. “title”: “Windows” sets the *title* property of the panel to “Windows”.

To load the JSON example to a *Settings* instance, use the *Settings.add_json_panel()* method. It will automatically instantiate a *SettingsPanel* and add it to *Settings*:

```
from kivy.config import ConfigParser

config = ConfigParser()
config.read('myconfig.ini')

s = Settings()
s.add_json_panel('My custom panel', config, 'settings_custom.json')
s.add_json_panel('Another panel', config, 'settings_test2.json')

# then use the s as a widget...
```

33.37.2 Different panel layouts

A kivy *App* can automatically create and display a *Settings* instance. See the *settings_cls* documentation for details on how to choose which settings class to display.

Several pre-built settings widgets are available. All except *SettingsWithNoMenu* include close buttons triggering the on_close event.

- *Settings*: Displays settings with a sidebar at the left to switch between json panels.
- *SettingsWithSidebar*: A trivial subclass of *Settings*.

- **SettingsWithSpinner**: Displays settings with a spinner at the top, which can be used to switch between json panels. Uses **InterfaceWithSpinner** as the *interface_cls*. This is the default behavior from Kivy 1.8.0.
- **SettingsWithTabbedPanel**: Displays json panels as individual tabs in a **TabbedPanel**. Uses **InterfaceWithTabbedPanel** as the *interface_cls*.
- **SettingsWithNoMenu**: Displays a single json panel, with no way to switch to other panels and no close button. This makes it impossible for the user to exit unless *close_settings()* is overridden with a different close trigger! Uses **InterfaceWithNoMenu** as the *interface_cls*.

You can construct your own settings panels with any layout you choose by setting *Settings.interface_cls*. This should be a widget that displays a json settings panel with some way to switch between panels. An instance will be automatically created by *Settings*.

Interface widgets may be anything you like, but *must* have a method *add_panel* that receives newly created json settings panels for the interface to display. See the documentation for *InterfaceWithSideBar* for more information. They may optionally dispatch an *on_close* event, for instance if a close button is clicked. This event is used by *Settings* to trigger its own *on_close* event.

For a complete, working example, please see `kivy/examples/settings/main.py`.

class `kivy.uix.settings.Settings(*args, **kwargs)`

Bases: *kivy.uix.boxlayout.BoxLayout*

Settings UI. Check module documentation for more information on how to use this class.

Events

on_config_change: **ConfigParser instance, section, key, value** Fired when the section's key-value pair of a **ConfigParser** changes.

on_close Fired by the default panel when the Close button is pressed.

add_interface()

(Internal) creates an instance of *Settings.interface_cls*, and sets it to *interface*. When json panels are created, they will be added to this interface which will display them to the user.

add_json_panel(title, config, filename=None, data=None)

Create and add a new *SettingsPanel* using the configuration *config* with the JSON definition *filename*. If *filename* is not set, then the JSON definition is read from the *data* parameter instead.

Check the *Create a panel from JSON* section in the documentation for more information about JSON format and the usage of this function.

add_kivy_panel()

Add a panel for configuring Kivy. This panel acts directly on the kivy configuration. Feel free to include or exclude it in your configuration.

See *use_kivy_settings()* for information on enabling/disabling the automatic kivy panel.

create_json_panel(title, config, filename=None, data=None)

Create new *SettingsPanel*.

New in version 1.5.0.

Check the documentation of *add_json_panel()* for more information.

interface

(internal) Reference to the widget that will contain, organise and display the panel configuration panel widgets.

interface is an *ObjectProperty* and defaults to None.

interface_cls

The widget class that will be used to display the graphical interface for the settings panel. By default, it displays one Settings panel at a time with a sidebar to switch between them.

interface_cls is an *ObjectProperty* and defaults to *InterfaceWithSidebar*.

Changed in version 1.8.0: If you set a string, the *Factory* will be used to resolve the class.

on_touch_down(*touch*)

Receive a touch down event.

Parameters

touch: *MotionEvent* class Touch received. The touch is in parent coordinates. See *relative_layout* 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.

register_type(*tp, cls*)

Register a new type that can be used in the JSON definition.

class kivy.uix.settings.**SettingsPanel**(***kwargs*)

Bases: *kivy.uix.gridlayout.GridLayout*

This class is used to construct panel settings, for use with a *Settings* instance or subclass.

config

A *kivy.config.ConfigParser* instance. See module documentation for more information.

get_value(*section, key*)

Return the value of the section/key from the *config* ConfigParser instance. This function is used by *SettingItem* to get the value for a given section/key.

If you don't want to use a ConfigParser instance, you might want to override this function.

settings

A *Settings* instance that will be used to fire the *on_config_change* event.

title

Title of the panel. The title will be reused by the *Settings* in the sidebar.

class kivy.uix.settings.**SettingItem**(***kwargs*)

Bases: *kivy.uix.floatlayout.FloatLayout*

Base class for individual settings (within a panel). This class cannot be used directly; it is used for implementing the other setting classes. It builds a row with a title/description (left) and a setting control (right).

Look at *SettingBoolean*, *SettingNumeric* and *SettingOptions* for usage examples.

Events

on_release Fired when the item is touched and then released.

add_widget(**largs*)

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)
```

content

(internal) Reference to the widget that contains the real setting. As soon as the content object is set, any further call to `add_widget` will call the `content.add_widget`. This is automatically set.

content is an *ObjectProperty* and defaults to None.

desc

Description of the setting, rendered on the line below the title.

desc is a *StringProperty* and defaults to None.

disabled

Indicate if this setting is disabled. If True, all touches on the setting item will be discarded.

disabled is a *BooleanProperty* and defaults to False.

key

Key of the token inside the *section* in the *ConfigParser* instance.

key is a *StringProperty* and defaults to None.

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.

panel

(internal) Reference to the *SettingsPanel* for this setting. You don't need to use it.

panel is an *ObjectProperty* and defaults to None.

section

Section of the token inside the *ConfigParser* instance.

section is a *StringProperty* and defaults to None.

selected_alpha

(internal) Float value from 0 to 1, used to animate the background when the user touches the item.

selected_alpha is a *NumericProperty* and defaults to 0.

title

Title of the setting, defaults to '<No title set>'.

title is a *StringProperty* and defaults to '<No title set>'.

value

Value of the token according to the *ConfigParser* instance. Any change to this value will trigger a `Settings.on_config_change()` event.

value is an *ObjectProperty* and defaults to None.

class `kivy.uix.settings.SettingString(**kwargs)`

Bases: *kivy.uix.settings.SettingItem*

Implementation of a string setting on top of a *SettingItem*. It is visualized with a *Label* widget that, when clicked, will open a *Popup* with a *TextInput* so the user can enter a custom value.

popup

(internal) Used to store the current popup when it's shown.

popup is an *ObjectProperty* and defaults to None.

textinput

(internal) Used to store the current textinput from the popup and to listen for changes.

textinput is an *ObjectProperty* and defaults to None.

class `kivy.uix.settings.SettingPath(**kwargs)`

Bases: *kivy.uix.settings.SettingItem*

Implementation of a Path setting on top of a *SettingItem*. It is visualized with a *Label* widget that, when clicked, will open a *Popup* with a *FileChooserListView* so the user can enter a custom value.

New in version 1.1.0.

dirselect

Whether to allow selection of directories.

dirselect is a *BooleanProperty* and defaults to True.

New in version 1.10.0.

popup

(internal) Used to store the current popup when it is shown.

popup is an *ObjectProperty* and defaults to None.

show_hidden

Whether to show 'hidden' filenames. What that means is operating-system-dependent.

show_hidden is an *BooleanProperty* and defaults to False.

New in version 1.10.0.

textinput

(internal) Used to store the current textinput from the popup and to listen for changes.

textinput is an *ObjectProperty* and defaults to None.

class `kivy.uix.settings.SettingBoolean(**kwargs)`

Bases: *kivy.uix.settings.SettingItem*

Implementation of a boolean setting on top of a *SettingItem*. It is visualized with a *Switch* widget. By default, 0 and 1 are used for values: you can change them by setting *values*.

values

Values used to represent the state of the setting. If you want to use “yes” and “no” in your ConfigParser instance:

```
SettingBoolean(..., values=['no', 'yes'])
```

Warning: You need a minimum of two values, the index 0 will be used as False, and index 1 as True

values is a *ListProperty* and defaults to ['0', '1']

class kivy.uix.settings.**SettingNumeric**(**kwargs)

Bases: *kivy.uix.settings.SettingString*

Implementation of a numeric setting on top of a *SettingString*. It is visualized with a *Label* widget that, when clicked, will open a *Popup* with a *Textinput* so the user can enter a custom value.

class kivy.uix.settings.**SettingOptions**(**kwargs)

Bases: *kivy.uix.settings.SettingItem*

Implementation of an option list on top of a *SettingItem*. It is visualized with a *Label* widget that, when clicked, will open a *Popup* with a list of options from which the user can select.

options

List of all available options. This must be a list of “string” items. Otherwise, it will crash. :)

options is a *ListProperty* and defaults to [].

popup

(internal) Used to store the current popup when it is shown.

popup is an *ObjectProperty* and defaults to None.

class kivy.uix.settings.**SettingTitle**(**kwargs)

Bases: *kivy.uix.label.Label*

A simple title label, used to organize the settings in sections.

class kivy.uix.settings.**SettingsWithSidebar**(*args, **kwargs)

Bases: *kivy.uix.settings.Settings*

A settings widget that displays settings panels with a sidebar to switch between them. This is the default behaviour of *Settings*, and this widget is a trivial wrapper subclass.

class kivy.uix.settings.**SettingsWithSpinner**(*args, **kwargs)

Bases: *kivy.uix.settings.Settings*

A settings widget that displays one settings panel at a time with a spinner at the top to switch between them.

class kivy.uix.settings.**SettingsWithTabbedPanel**(*args, **kwargs)

Bases: *kivy.uix.settings.Settings*

A settings widget that displays settings panels as pages in a *TabbedPanel*.

class kivy.uix.settings.**SettingsWithNoMenu**(*args, **kwargs)

Bases: *kivy.uix.settings.Settings*

A settings widget that displays a single settings panel with *no* Close button. It will not accept more than one Settings panel. It is intended for use in programs with few enough settings that a full panel switcher is not useful.

Warning: This Settings panel does *not* provide a Close button, and so it is impossible to leave the settings screen unless you also add other behaviour or override `display_settings()` and `close_settings()`.

class `kivy.uix.settings.InterfaceWithSidebar(*args, **kwargs)`

Bases: `kivy.uix.boxlayout.BoxLayout`

The default Settings interface class. It displays a sidebar menu with names of available settings panels, which may be used to switch which one is currently displayed.

See `add_panel()` for information on the method you must implement if creating your own interface.

This class also dispatches an event 'on_close', which is triggered when the sidebar menu's close button is released. If creating your own interface widget, it should also dispatch such an event which will automatically be caught by `Settings` and used to trigger its own 'on_close' event.

add_panel (`panel, name, uid`)

This method is used by Settings to add new panels for possible display. Any replacement for `ContentPanel` *must* implement this method.

Parameters

panel: `SettingsPanel` It should be stored and the interface should provide a way to switch between panels.

name: The name of the panel as a string. It may be used to represent the panel but isn't necessarily unique.

uid: A unique int identifying the panel. It should be used to identify and switch between panels.

content

(internal) A reference to the panel display widget (a `ContentPanel`).

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

menu

(internal) A reference to the sidebar menu widget.

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

class `kivy.uix.settings.ContentPanel(**kwargs)`

Bases: `kivy.uix.scrollview.ScrollView`

A class for displaying settings panels. It displays a single settings panel at a time, taking up the full size and shape of the `ContentPanel`. It is used by `InterfaceWithSidebar` and `InterfaceWithSpinner` to display settings.

add_panel (`panel, name, uid`)

This method is used by Settings to add new panels for possible display. Any replacement for `ContentPanel` *must* implement this method.

Parameters

panel: `SettingsPanel` It should be stored and displayed when requested.

name: The name of the panel as a string. It may be used to represent the panel.

uid: A unique int identifying the panel. It should be stored and used to identify panels when switching.

add_widget(widget)

Add a new widget as a child of this widget.

Parameters

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

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

New in version 1.0.5.

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

New in version 1.9.0.

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

container

(internal) A reference to the GridLayout that contains the settings panel.

container is an *ObjectProperty* and defaults to None.

current_panel

(internal) A reference to the current settings panel.

current_panel is an *ObjectProperty* and defaults to None.

current_uid

(internal) A reference to the uid of the current settings panel.

current_uid is a *NumericProperty* and defaults to 0.

on_current_uid(*args)

The uid of the currently displayed panel. Changing this will automatically change the displayed panel.

Parameters

uid: A panel uid. It should be used to retrieve and display a settings panel that has previously been added with *add_panel()*.

panels

(internal) Stores a dictionary mapping settings panels to their uids.

panels is a *DictProperty* and defaults to {}.

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)
```

class kivy.uix.settings.MenuSidebar(**kwargs)

Bases: *kivy.uix.floatlayout.FloatLayout*

The menu used by *InterfaceWithSidebar*. It provides a sidebar with an entry for each settings panel, which the user may click to select.

add_item(name, uid)

This method is used to add new panels to the menu.

Parameters

name: The name (a string) of the panel. It should be used to represent the panel in the menu.

uid: The name (an int) of the panel. It should be used internally to represent the panel and used to set self.selected_uid when the panel is changed.

buttons_layout

(internal) Reference to the GridLayout that contains individual settings panel menu buttons.

buttons_layout is an *ObjectProperty* and defaults to None.

close_button

(internal) Reference to the widget's Close button.

buttons_layout is an *ObjectProperty* and defaults to None.

on_selected_uid(*args)

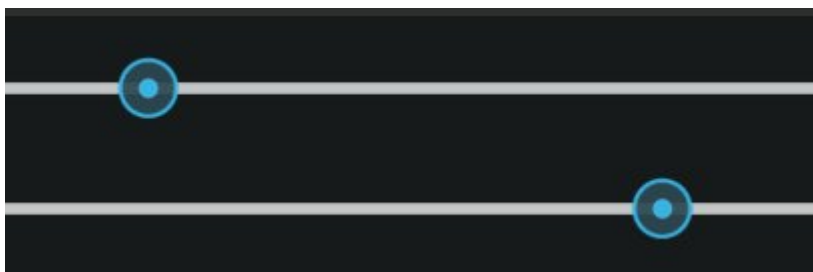
(internal) unselects any currently selected menu buttons, unless they represent the current panel.

selected_uid

The uid of the currently selected panel. This may be used to switch between displayed panels, e.g. by binding it to the *current_uid* of a *ContentPanel*.

selected_uid is a *NumericProperty* and defaults to 0.

33.38 Slider



The *Slider* widget looks like a scrollbar. It supports horizontal and vertical orientations, min/max values and a default value.

To create a slider from -100 to 100 starting from 25:

```
from kivy.uix.slider import Slider
s = Slider(min=-100, max=100, value=25)
```


To create a vertical slider:

```
from kivy.uix.slider import Slider
s = Slider(orientation='vertical')
```

To create a slider with a red line tracking the value:

```
from kivy.uix.slider import Slider
s = Slider(value_track=True, value_track_color=[1, 0, 0, 1])
```

Kv Example:

```
BoxLayout:
    Slider:
        id: slider
        min: 0
        max: 100
        step: 1
        orientation: 'vertical'

    Label:
        text: str(slider.value)
```

class kivy.uix.slider.Slider(**kwargs)

Bases: *kivy.uix.widget.Widget*

Class for creating a Slider widget.

Check module documentation for more details.

background_disabled_horizontal

Background of the disabled slider used in the horizontal orientation.

New in version 1.10.0.

background_disabled_horizontal is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/sliderh_background_disabled*.

background_disabled_vertical

Background of the disabled slider used in the vertical orientation.

New in version 1.10.0.

background_disabled_vertical is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/sliderv_background_disabled*.

background_horizontal

Background of the slider used in the horizontal orientation.

New in version 1.10.0.

background_horizontal is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/sliderh_background*.

background_vertical

Background of the slider used in the vertical orientation.

New in version 1.10.0.

background_vertical is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/sliderv_background*.

background_width

Slider's background's width (thickness), used in both horizontal and vertical orientations.

background_width is a *NumericProperty* and defaults to 36sp.

border_horizontal

Border used to draw the slider background in horizontal orientation.

border_horizontal is a *ListProperty* and defaults to [0, 18, 0, 18].

border_vertical

Border used to draw the slider background in vertical orientation.

border_horizontal is a *ListProperty* and defaults to [18, 0, 18, 0].

cursor_disabled_image

Path of the image used to draw the disabled slider cursor.

cursor_image is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/slider_cursor_disabled*.

cursor_height

Height of the cursor image.

cursor_height is a *NumericProperty* and defaults to 32sp.

cursor_image

Path of the image used to draw the slider cursor.

cursor_image is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/slider_cursor*.

cursor_size

Size of the cursor image.

cursor_size is a *ReferenceListProperty* of (*cursor_width*, *cursor_height*) properties.

cursor_width

Width of the cursor image.

cursor_width is a *NumericProperty* and defaults to 32sp.

max

Maximum value allowed for *value*.

max is a *NumericProperty* and defaults to 100.

min

Minimum value allowed for *value*.

min is a *NumericProperty* and defaults to 0.

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.

orientation

Orientation of the slider.

orientation is an *OptionProperty* and defaults to 'horizontal'. Can take a value of 'vertical' or 'horizontal'.

padding

Padding of the slider. The padding is used for graphical representation and interaction. It prevents the cursor from going out of the bounds of the slider bounding box.

By default, padding is 16sp. The range of the slider is reduced from padding *2 on the screen. It allows drawing the default cursor of 32sp width without having the cursor go out of the widget.

padding is a *NumericProperty* and defaults to 16sp.

range

Range of the slider in the format (minimum value, maximum value):

```
>>> slider = Slider(min=10, max=80)
>>> slider.range
[10, 80]
>>> slider.range = (20, 100)
>>> slider.min
20
>>> slider.max
100
```

range is a *ReferenceListProperty* of (*min*, *max*) properties.

sensitivity

Whether the touch collides with the whole body of the widget or with the slider handle part only.

New in version 1.10.1.

sensitivity is a *OptionProperty* and defaults to 'all'. Can take a value of 'all' or 'handle'.

step

Step size of the slider.

New in version 1.4.0.

Determines the size of each interval or step the slider takes between min and max. If the value range can't be evenly divisible by step the last step will be capped by slider.max

step is a *NumericProperty* and defaults to 1.

value

Current value used for the slider.

value is a *NumericProperty* and defaults to 0.

value_normalized

Normalized value inside the *range* (min/max) to 0-1 range:

```
>>> slider = Slider(value=50, min=0, max=100)
>>> slider.value
50
>>> slider.value_normalized
0.5
>>> slider.value = 0
>>> slider.value_normalized
```

(continues on next page)

(continued from previous page)

```
0
>>> slider.value = 100
>>> slider.value_normalized
1
```

You can also use it for setting the real value without knowing the minimum and maximum:

```
>>> slider = Slider(min=0, max=200)
>>> slider.value_normalized = .5
>>> slider.value
100
>>> slider.value_normalized = 1.
>>> slider.value
200
```

value_normalized is an *AliasProperty*.

value_pos

Position of the internal cursor, based on the normalized value.

value_pos is an *AliasProperty*.

value_track

Decides if slider should draw the line indicating the space between *min* and *value* properties values.

value_track is a *BooleanProperty* and defaults to False.

value_track_color

Color of the *value_line* in rgba format.

value_track_color is a *ListProperty* and defaults to [1, 1, 1, 1].

value_track_width

Width of the track line.

value_track_width is a *NumericProperty* and defaults to 3dp.

33.39 Spinner

New in version 1.4.0.



Spinner is a widget that provides a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all the other available values from which the user can select a new one.

Example:

```
from kivy.base import runTouchApp
from kivy.uix.spinner import Spinner

spinner = Spinner(
    # default value shown
    text='Home',
    # available values
    values=('Home', 'Work', 'Other', 'Custom'),
    # just for positioning in our example
    size_hint=(None, None),
    size=(100, 44),
    pos_hint={'center_x': .5, 'center_y': .5})

def show_selected_value(spinner, text):
    print('The spinner', spinner, 'has text', text)

spinner.bind(text=show_selected_value)

runTouchApp(spinner)
```

Kv Example:

```
FloatLayout:
    Spinner:
        size_hint: None, none
        size: 100, 44
        pos_hint: {'center': (.5, .5)}
        text: 'Home'
        values: 'Home', 'Work', 'Other', 'Custom'
        on_value:
            print: "The spinner {} has text {}".format(self, self.value)
```

```
class kivy.uix.spinner.Spinner(**kwargs)
    Bases: kivy.uix.button.Button
```

Spinner class, see module documentation for more information.

dropdown_cls

Class used to display the dropdown list when the Spinner is pressed.

dropdown_cls is an *ObjectProperty* and defaults to *DropDown*.

Changed in version 1.8.0: If set to a string, the *Factory* will be used to resolve the class name.

is_open

By default, the spinner is not open. Set to True to open it.

is_open is a *BooleanProperty* and defaults to False.

New in version 1.4.0.

option_cls

Class used to display the options within the dropdown list displayed under the Spinner. The *text* property of the class will be used to represent the value.

The option class requires:

- a *text* property, used to display the value.
- an *on_release* event, used to trigger the option when pressed/touched.
- a *size_hint_y* of None.
- the *height* to be set.

option_cls is an *ObjectProperty* and defaults to *SpinnerOption*.

Changed in version 1.8.0: If you set a string, the *Factory* will be used to resolve the class.

sync_height

Each element in a dropdown list uses a default/user-supplied height. Set to True to propagate the Spinner's height value to each dropdown list element.

New in version 1.10.0.

sync_height is a *BooleanProperty* and defaults to False.

text_autoupdate

Indicates if the spinner's *text* should be automatically updated with the first value of the *values* property. Setting it to True will cause the spinner to update its *text* property every time *attr:values* are changed.

New in version 1.10.0.

text_autoupdate is a *BooleanProperty* and defaults to False.

values

Values that can be selected by the user. It must be a list of strings.

values is a *ListProperty* and defaults to [].

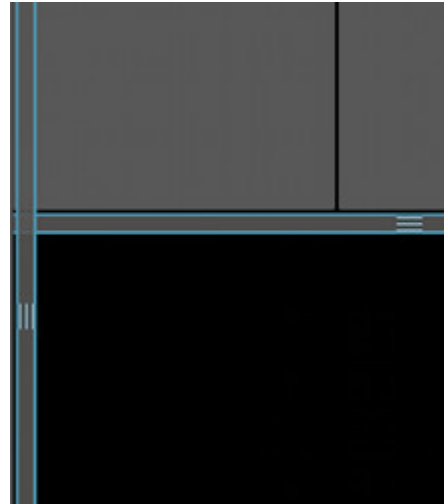
class `kivy.uix.spinner.SpinnerOption(**kwargs)`

Bases: *kivy.uix.button.Button*

Special button used in the *Spinner* dropdown list. By default, this is just a *Button* with a *size_hint_y* of None and a height of *48dp*.

33.40 Splitter

New in version 1.5.0.



The *Splitter* is a widget that helps you re-size its child widget/layout by letting you re-size it via dragging the boundary or double tapping the boundary. This widget is similar to the *ScrollView* in that it allows only one child widget.

Usage:

```
splitter = Splitter(sizable_from = 'right')
splitter.add_widget(layout_or_widget_instance)
splitter.min_size = 100
splitter.max_size = 250
```

To change the size of the strip/border used for resizing:

```
splitter.strip_size = '10pt'
```

To change its appearance:

```
splitter.strip_cls = your_custom_class
```

You can also change the appearance of the *strip_cls*, which defaults to *SplitterStrip*, by overriding the *kv* rule in your app:

```
<SplitterStrip>:
    horizontal: True if self.parent and self.parent.sizable_from[0] in ('t', 'b')_
    → else False
    background_normal: 'path to normal horizontal image' if self.horizontal else
    → 'path to vertical normal image'
    background_down: 'path to pressed horizontal image' if self.horizontal else
    → 'path to vertical pressed image'
```

class kivy.uix.splitter.**Splitter**(**kwargs)

Bases: *kivy.uix.boxlayout.BoxLayout*

See module documentation.

Events

on_press: Fired when the splitter is pressed.

on_release: Fired when the splitter is released.

Changed in version 1.6.0: Added *on_press* and *on_release* events.

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)
```

border

Border used for the *BorderImage* graphics instruction.

This must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instructions for more information about how to use it.

border is a *ListProperty* and defaults to (4, 4, 4, 4).

clear_widgets()

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

keep_within_parent

If True, will limit the splitter to stay within its parent widget.

keep_within_parent is a *BooleanProperty* and defaults to False.

New in version 1.9.0.

max_size

Specifies the maximum size beyond which the widget is not resizable.

max_size is a *NumericProperty* and defaults to 500pt.

min_size

Specifies the minimum size beyond which the widget is not resizable.

min_size is a *NumericProperty* and defaults to 100pt.

remove_widget(widget, *largs)

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()
```

(continues on next page)

(continued from previous page)

```
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

rescale_with_parent

If True, will automatically change size to take up the same proportion of the parent widget when it is resized, while staying within *min_size* and *max_size*. As long as these attributes can be satisfied, this stops the *Splitter* from exceeding the parent size during rescaling.

rescale_with_parent is a *BooleanProperty* and defaults to False.

New in version 1.9.0.

sizable_from

Specifies whether the widget is resizable. Options are: *left*, *right*, *top* or *bottom*

sizable_from is an *OptionProperty* and defaults to *left*.

strip_cls

Specifies the class of the resize Strip.

strip_cls is an *kivy.properties.ObjectProperty* and defaults to *SplitterStrip*, which is of type *Button*.

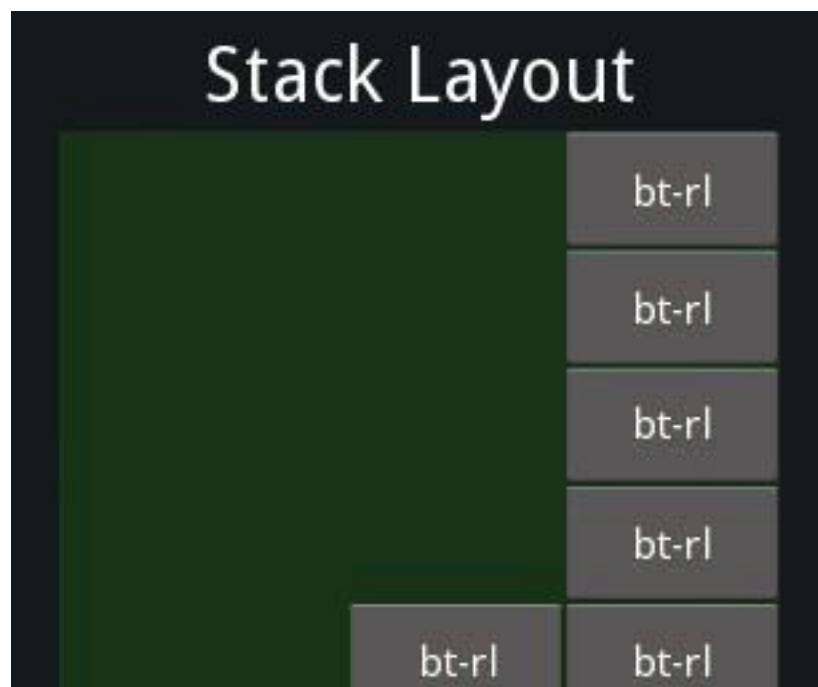
Changed in version 1.8.0: If you set a string, the *Factory* will be used to resolve the class.

strip_size

Specifies the size of resize strip

strip_size is a *NumericProperty* defaults to *10pt*

33.41 Stack Layout



New in version 1.0.5.

The *StackLayout* arranges children vertically or horizontally, as many as the layout can fit. The size of the individual children widgets do not have to be uniform.

For example, to display widgets that get progressively larger in width:

```
root = StackLayout()
for i in range(25):
    btn = Button(text=str(i), width=40 + i * 5, size_hint=(None, 0.15))
    root.add_widget(btn)
```



class `kivy.uix.stacklayout.StackLayout`(**kwargs)

Bases: `kivy.uix.layout.Layout`

Stack layout class. See module documentation for more information.

do_layout(*args)

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.

minimum_height

Minimum height needed to contain all children. It is automatically set by the layout.

New in version 1.0.8.

minimum_height is a `kivy.properties.NumericProperty` and defaults to 0.

minimum_size

Minimum size needed to contain all children. It is automatically set by the layout.

New in version 1.0.8.

minimum_size is a `ReferenceListProperty` of (*minimum_width*, *minimum_height*) properties.

minimum_width

Minimum width needed to contain all children. It is automatically set by the layout.

New in version 1.0.8.

minimum_width is a `kivy.properties.NumericProperty` and defaults to 0.

orientation

Orientation of the layout.

orientation is an *OptionProperty* and defaults to 'lr-tb'.

Valid orientations are 'lr-tb', 'tb-lr', 'rl-tb', 'tb-rl', 'lr-bt', 'bt-lr', 'rl-bt' and 'bt-rl'.

Changed in version 1.5.0: *orientation* now correctly handles all valid combinations of 'lr', 'rl', 'tb', 'bt'. Before this version only 'lr-tb' and 'tb-lr' were supported, and 'tb-lr' was misnamed and placed widgets from bottom to top and from right to left (reversed compared to what was expected).

Note: 'lr' means Left to Right. 'rl' means Right to Left. 'tb' means Top to Bottom. 'bt' means Bottom to Top.

padding

Padding between the layout box and it's children: [padding_left, padding_top, padding_right, padding_bottom].

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

Changed in version 1.7.0: Replaced the *NumericProperty* with a *VariableListProperty*.

padding is a *VariableListProperty* and defaults to [0, 0, 0, 0].

spacing

Spacing between children: [spacing_horizontal, spacing_vertical].

spacing also accepts a single argument form [spacing].

spacing is a *VariableListProperty* and defaults to [0, 0].

33.42 Stencil View

New in version 1.0.4.

StencilView limits the drawing of child widgets to the *StencilView*'s bounding box. Any drawing outside the bounding box will be clipped (trashed).

The *StencilView* uses the stencil graphics instructions under the hood. It provides an efficient way to clip the drawing area of children.

Note: As with the stencil graphics instructions, you cannot stack more than 128 stencil-aware widgets.

Note: *StencilView* is not a layout. Consequently, you have to manage the size and position of its children directly. You can combine (subclass both) a *StencilView* and a *Layout* in order to achieve a layout's behavior. For example:

```
class BoxStencil(BoxLayout, StencilView):
    pass
```

```
class kivy.uix.stencilview.StencilView(**kwargs)
```

Bases: *kivy.uix.widget.Widget*

StencilView class. See module documentation for more information.

33.43 Switch

New in version 1.0.7.



The *Switch* widget is active or inactive, like a mechanical light switch. The user can swipe to the left/right to activate/deactivate it:

```
switch = Switch(active=True)
```

To attach a callback that listens to the activation state:

```
def callback(instance, value):
    print('the switch', instance, 'is', value)

switch = Switch()
switch.bind(active=callback)
```

By default, the representation of the widget is static. The minimum size required is 83x32 pixels (defined by the background image). The image is centered within the widget.

The entire widget is active, not just the part with graphics. As long as you swipe over the widget's bounding box, it will work.

Note: If you want to control the state with a single touch instead of a swipe, use the `ToggleButton` instead.

Kv Example:

```
BoxLayout:
    Label:
        text: 'power up'
    Switch:
        id: switch
    Label:
        text: 'wooooooooooooooh' if switch.active else ''
```

```
class kivy.uix.switch.Switch(**kwargs)
```

Bases: *kivy.uix.widget.Widget*

Switch class. See module documentation for more information.

active

Indicate whether the switch is active or inactive.

active is a *BooleanProperty* and defaults to False.

active_norm_pos

(internal) Contains the normalized position of the movable element inside the switch, in the 0-1 range.

active_norm_pos is a *NumericProperty* and defaults to 0.

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.

touch_control

(internal) Contains the touch that currently interacts with the switch.

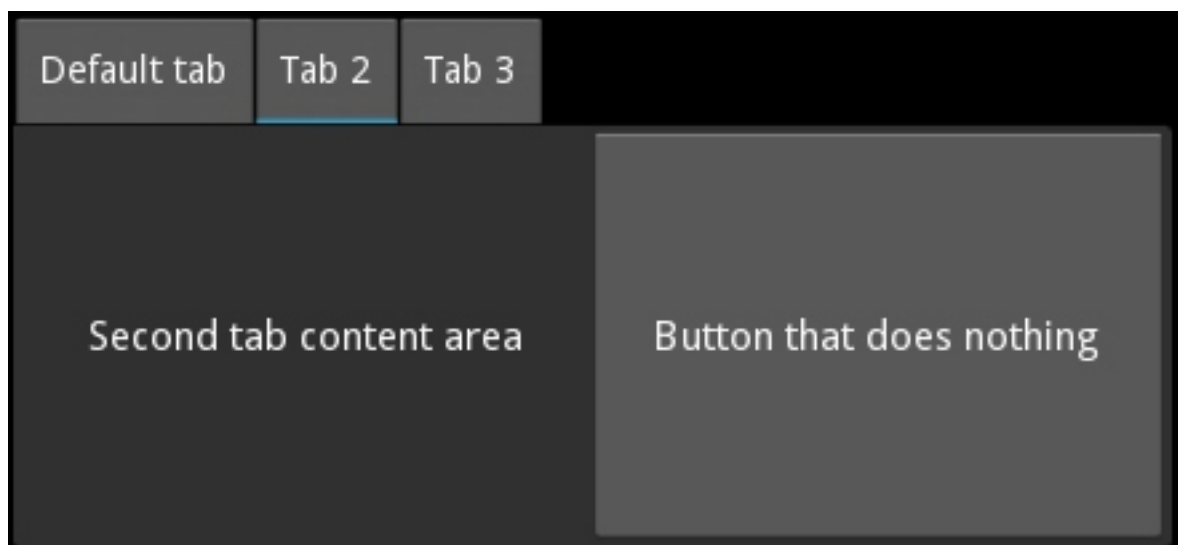
touch_control is an *ObjectProperty* and defaults to None.

touch_distance

(internal) Contains the distance between the initial position of the touch and the current position to determine if the swipe is from the left or right.

touch_distance is a *NumericProperty* and defaults to 0.

33.44 TabbedPanel



New in version 1.3.0.

The *TabbedPanel* widget manages different widgets in tabs, with a header area for the actual tab buttons and a content area for showing the current tab content.

The *TabbedPanel* provides one default tab.

33.44.1 Simple example

```
'''
TabbedPanel
=====

Test of the widget TabbedPanel.
'''

from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.lang import Builder

Builder.load_string("""
<Test>:
    size_hint: .5, .5
    pos_hint: {'center_x': .5, 'center_y': .5}
    do_default_tab: False

    TabbedPanelItem:
        text: 'first tab'
        Label:
            text: 'First tab content area'
    TabbedPanelItem:
        text: 'tab2'
        BoxLayout:
            Label:
                text: 'Second tab content area'
            Button:
                text: 'Button that does nothing'
    TabbedPanelItem:
        text: 'tab3'
        RstDocument:
            text:
                '\\n'.join(("Hello world", "-----",
                "You are in the third tab."))

""")

class Test(TabbedPanel):
    pass

class TabbedPanelApp(App):
    def build(self):
        return Test()

if __name__ == '__main__':
    TabbedPanelApp().run()
```

Note: A new class *TabbedPanelItem* has been introduced in 1.5.0 for convenience. So now one can simply add a *TabbedPanelItem* to a *TabbedPanel* and *content* to the *TabbedPanelItem* as in the

example provided above.

33.44.2 Customize the Tabbed Panel

You can choose the position in which the tabs are displayed:

```
tab_pos = 'top_mid'
```

An individual tab is called a `TabbedPanelHeader`. It is a special button containing a *content* property. You add the `TabbedPanelHeader` first, and set its *content* property separately:

```
tp = TabbedPanel()
th = TabbedPanelHeader(text='Tab2')
tp.add_widget(th)
```

An individual tab, represented by a `TabbedPanelHeader`, needs its content set. This content can be any widget. It could be a layout with a deep hierarchy of widgets, or it could be an individual widget, such as a label or a button:

```
th.content = your_content_instance
```

There is one “shared” main content area active at any given time, for all the tabs. Your app is responsible for adding the content of individual tabs and for managing them, but it’s not responsible for content switching. The tabbed panel handles switching of the main content object as per user action.

There is a default tab added when the tabbed panel is instantiated. Tabs that you add individually as above, are added in addition to the default tab. Thus, depending on your needs and design, you will want to customize the default tab:

```
tp.default_tab_text = 'Something Specific To Your Use'
```

The default tab machinery requires special consideration and management. Accordingly, an *on_default_tab* event is provided for associating a callback:

```
tp.bind(default_tab = my_default_tab_callback)
```

It’s important to note that by default, `default_tab_cls` is of type *TabbedPanelHeader* and thus has the same properties as other tabs.

Since 1.5.0, it is now possible to disable the creation of the `default_tab` by setting `do_default_tab` to `False`.

Tabs and content can be removed in several ways:

```
tp.remove_widget(widget/tabbed_panel_header)
or
tp.clear_widgets() # to clear all the widgets in the content area
or
tp.clear_tabs() # to remove the TabbedPanelHeaders
```

To access the children of the tabbed panel, use `content.children`:

```
tp.content.children
```

To access the list of tabs:

```
tp.tab_list
```

To change the appearance of the main tabbed panel content:

```
background_color = (1, 0, 0, .5) #50% translucent red
border = [0, 0, 0, 0]
background_image = 'path/to/background/image'
```

To change the background of a individual tab, use these two properties:

```
tab_header_instance.background_normal = 'path/to/tab_head/img'
tab_header_instance.background_down = 'path/to/tab_head/img_pressed'
```

A `TabbedPanelStrip` contains the individual tab headers. To change the appearance of this tab strip, override the canvas of `TabbedPanelStrip`. For example, in the kv language:

```
<TabbedPanelStrip>
    canvas:
        Color:
            rgba: (0, 1, 0, 1) # green
        Rectangle:
            size: self.size
            pos: self.pos
```

By default the tabbed panel strip takes its background image and color from the tabbed panel's `background_image` and `background_color`.

class `kivy.uix.tabbedpanel.StripLayout(**kwargs)`

Bases: `kivy.uix.gridlayout.GridLayout`

The main layout that is used to house the entire tabbedpanel strip including the blank areas in case the tabs don't cover the entire width/height.

New in version 1.8.0.

background_image

Background image to be used for the Strip layout of the `TabbedPanel`.

`background_image` is a *StringProperty* and defaults to a transparent image.

border

Border property for the `background_image`.

`border` is a *ListProperty* and defaults to [4, 4, 4, 4]

class `kivy.uix.tabbedpanel.TabbedPanel(**kwargs)`

Bases: `kivy.uix.gridlayout.GridLayout`

The `TabbedPanel` class. See module documentation for more information.

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)
```

background_color

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

background_color is a *ListProperty* and defaults to [1, 1, 1, 1].

background_disabled_image

Background image of the main shared content object when disabled.

New in version 1.8.0.

background_disabled_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/tab'.

background_image

Background image of the main shared content object.

background_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/tab'.

border

Border used for *BorderImage* graphics instruction, used itself for *background_image*. Can be changed for a custom background.

It must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instructions for more information.

border is a *ListProperty* and defaults to (16, 16, 16, 16)

clear_widgets(kwargs)**

Remove all (or the specified) children of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

content

This is the object holding (current_tab's content is added to this) the content of the current tab. To Listen to the changes in the content of the current tab, you should bind to current_tabs *content* property.

content is an *ObjectProperty* and defaults to 'None'.

current_tab

Links to the currently selected or active tab.

New in version 1.4.0.

current_tab is an *AliasProperty*, read-only.

default_tab

Holds the default tab.

Note: For convenience, the automatically provided default tab is deleted when you change *default_tab* to something else. As of 1.5.0, this behaviour has been extended to every *default_tab* for consistency and not just the automatically provided one.

default_tab is an *AliasProperty*.

default_tab_cls

Specifies the class to use for the styling of the default tab.

New in version 1.4.0.

Warning: *default_tab_cls* should be subclassed from *TabbedPanelHeader*

default_tab_cls is an *ObjectProperty* and defaults to *TabbedPanelHeader*. If you set a string, the *Factory* will be used to resolve the class.

Changed in version 1.8.0: The *Factory* will resolve the class if a string is set.

default_tab_content

Holds the default tab content.

default_tab_content is an *AliasProperty*.

default_tab_text

Specifies the text displayed on the default tab header.

default_tab_text is a *StringProperty* and defaults to 'default tab'.

do_default_tab

Specifies whether a default_tab head is provided.

New in version 1.5.0.

do_default_tab is a *BooleanProperty* and defaults to 'True'.

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)
```

strip_border

Border to be used on *strip_image*.

New in version 1.8.0.

strip_border is a *ListProperty* and defaults to [4, 4, 4, 4].

strip_image

Background image of the tabbed strip.

New in version 1.8.0.

strip_image is a *StringProperty* and defaults to a empty image.

switch_to(header, do_scroll=False)

Switch to a specific panel header.

Changed in version 1.10.0.

If used with *do_scroll=True*, it scrolls to the header's tab too.

tab_height

Specifies the height of the tab header.

tab_height is a *NumericProperty* and defaults to 40.

tab_list

List of all the tab headers.

tab_list is an *AliasProperty* and is read-only.

tab_pos

Specifies the position of the tabs relative to the content. Can be one of: *left_top*, *left_mid*, *left_bottom*, *top_left*, *top_mid*, *top_right*, *right_top*, *right_mid*, *right_bottom*, *bottom_left*, *bottom_mid*, *bottom_right*.

tab_pos is an *OptionProperty* and defaults to 'top_left'.

tab_width

Specifies the width of the tab header.

tab_width is a *NumericProperty* and defaults to 100.

class `kivy.uix.tabbedpanel.TabbedPanelContent`(**kwargs)

Bases: *kivy.uix.floatlayout.FloatLayout*

The TabbedPanelContent class.

class `kivy.uix.tabbedpanel.TabbedPanelHeader`(**kwargs)

Bases: *kivy.uix.togglebutton.ToggleButton*

A Base for implementing a Tabbed Panel Head. A button intended to be used as a Heading/Tab for a TabbedPanel widget.

You can use this TabbedPanelHeader widget to add a new tab to a TabbedPanel.

content

Content to be loaded when this tab header is selected.

content is an *ObjectProperty* and defaults to None.

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 `kivy.uix.tabbedpanel.TabbedPanelItem`(**kwargs)

Bases: *kivy.uix.tabbedpanel.TabbedPanelHeader*

This is a convenience class that provides a header of type TabbedPanelHeader and links it with the content automatically. Thus facilitating you to simply do the following in kv language:

```
<TabbedPanel>:
    # ...other settings
    TabbedPanelItem:
        BoxLayout:
            Label:
                text: 'Second tab content area'
            Button:
                text: 'Button that does nothing'
```

New in version 1.5.0.

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)
```

class kivy.uix.tabbedpanel.TabbedPanelStrip(**kwargs)

Bases: *kivy.uix.gridlayout.GridLayout*

A strip intended to be used as background for Heading/Tab. This does not cover the blank areas in case the tabs don't cover the entire width/height of the TabbedPanel(use *StripLayout* for that).

tabbed_panel

Link to the panel that the tab strip is a part of.

tabbed_panel is an *ObjectProperty* and defaults to None .

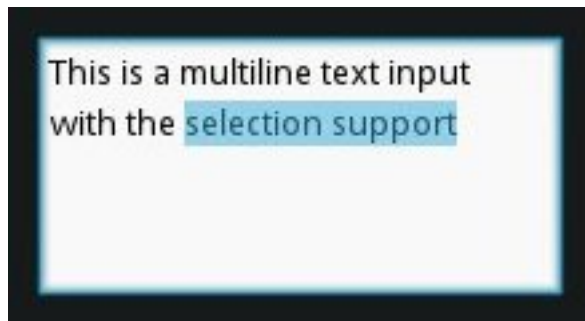
exception kivy.uix.tabbedpanel.TabbedPanelException

Bases: Exception

The TabbedPanelException class.

33.45 Text Input

New in version 1.0.4.



The *TextInput* widget provides a box for editable plain text.

Unicode, multiline, cursor navigation, selection and clipboard features are supported.

The *TextInput* uses two different coordinate systems:

- (x, y) - coordinates in pixels, mostly used for rendering on screen.
- (row, col) - cursor index in characters / lines, used for selection and cursor movement.

33.45.1 Usage example

To create a multiline *TextInput* (the 'enter' key adds a new line):

```
from kivy.uix.textinput import TextInput
textinput = TextInput(text='Hello world')
```

To create a singleline *TextInput*, set the *TextInput.multiline* property to False (the 'enter' key will defocus the TextInput and emit an *TextInput.on_text_validate()* event):

```
def on_enter(instance, value):
    print('User pressed enter in', instance)

textinput = TextInput(text='Hello world', multiline=False)
textinput.bind(on_text_validate=on_enter)
```

The textinput's text is stored in its *TextInput.text* property. To run a callback when the text changes:

```
def on_text(instance, value):
    print('The widget', instance, 'have:', value)

textinput = TextInput()
textinput.bind(text=on_text)
```

You can set the *focus* to a TextInput, meaning that the input box will be highlighted and keyboard focus will be requested:

```
textinput = TextInput(focus=True)
```

The textinput is defocused if the 'escape' key is pressed, or if another widget requests the keyboard. You can bind a callback to the focus property to get notified of focus changes:

```
def on_focus(instance, value):
    if value:
        print('User focused', instance)
    else:
        print('User defocused', instance)

textinput = TextInput()
textinput.bind(focus=on_focus)
```

See *FocusBehavior*, from which the *TextInput* inherits, for more details.

33.45.2 Selection

The selection is automatically updated when the cursor position changes. You can get the currently selected text from the *TextInput.selection_text* property.

33.45.3 Filtering

You can control which text can be added to the *TextInput* by overwriting *TextInput.insert_text()*. Every string that is typed, pasted or inserted by any other means into the *TextInput* is passed through this function. By overwriting it you can reject or change unwanted characters.

For example, to write only in capitalized characters:

```
class CapitalInput(TextInput):

    def insert_text(self, substring, from_undo=False):
        s = substring.upper()
        return super(CapitalInput, self).insert_text(s, from_undo=from_undo)
```

Or to only allow floats (0 - 9 and a single period):

```
class FloatInput(TextInput):

    pat = re.compile('[^0-9]')
    def insert_text(self, substring, from_undo=False):
        pat = self.pat
        if '.' in self.text:
            s = re.sub(pat, '', substring)
        else:
            s = '.'.join([re.sub(pat, '', s) for s in substring.split('.', 1)])
        return super(FloatInput, self).insert_text(s, from_undo=from_undo)
```

33.45.4 Default shortcuts

Shortcuts	Description
Left	Move cursor to left
Right	Move cursor to right
Up	Move cursor to up
Down	Move cursor to down
Home	Move cursor at the beginning of the line
End	Move cursor at the end of the line
PageUp	Move cursor to 3 lines before
PageDown	Move cursor to 3 lines after
Backspace	Delete the selection or character before the cursor
Del	Delete the selection of character after the cursor
Shift + <dir>	Start a text selection. Dir can be Up, Down, Left or Right
Control + c	Copy selection
Control + x	Cut selection
Control + v	Paste clipboard content
Control + a	Select all the content
Control + z	undo
Control + r	redo

Note: To enable Emacs-style keyboard shortcuts, you can use *EmacsBehavior*.

class `kivy.uix.textinput.TextInput(**kwargs)`

Bases: *kivy.uix.behaviors.focus.FocusBehavior*, *kivy.uix.widget.Widget*

TextInput class. See module documentation for more information.

Events

on_text_validate Fired only in multiline=False mode when the user hits 'enter'. This will also unfocus the textinput.

on_double_tap Fired when a double tap happens in the text input. The default behavior selects the text around the cursor position. More info at *on_double_tap()*.

on_triple_tap Fired when a triple tap happens in the text input. The default behavior selects the line around the cursor position. More info at *on_triple_tap()*.

on_quad_touch Fired when four fingers are touching the text input. The default behavior selects the whole text. More info at *on_quad_touch()*.

Warning: When changing a *TextInput* property that requires re-drawing, e.g. modifying the *text*, the updates occur on the next clock cycle and not instantly. This might cause any changes to the *TextInput* that occur between the modification and the next cycle to be ignored, or to use previous values. For example, after a update to the *text*, changing the cursor in the same clock frame will move it using the previous text and will likely end up in an incorrect position. The solution is to schedule any updates to occur on the next clock cycle using *schedule_once()*.

Note: Selection is cancelled when TextInput is focused. If you need to show selection when TextInput is focused, you should delay (use *Clock.schedule*) the call to the functions for selecting

text (select_all, select_text).

Changed in version 1.10.0: *background_disabled_active* has been removed.

Changed in version 1.9.0: *TextInput* now inherits from *FocusBehavior*. *keyboard_mode*, *show_keyboard()*, *hide_keyboard()*, *focus()*, and *input_type* have been removed since they are now inherited from *FocusBehavior*.

Changed in version 1.7.0: *on_double_tap*, *on_triple_tap* and *on_quad_touch* events added.

allow_copy

Decides whether to allow copying the text.

New in version 1.8.0.

allow_copy is a *BooleanProperty* and defaults to True.

auto_indent

Automatically indent multiline text.

New in version 1.7.0.

auto_indent is a *BooleanProperty* and defaults to False.

background_active

Background image of the TextInput when it's in focus.

New in version 1.4.1.

background_active is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/textinput_active'.

background_color

Current color of the background, in (r, g, b, a) format.

New in version 1.2.0.

background_color is a *ListProperty* and defaults to [1, 1, 1, 1] (white).

background_disabled_normal

Background image of the TextInput when disabled.

New in version 1.8.0.

background_disabled_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/textinput_disabled'.

background_normal

Background image of the TextInput when it's not in focus.

New in version 1.4.1.

background_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/textinput'.

base_direction

Base direction of text, this impacts horizontal alignment when *halign* is *auto* (the default). Available options are: None, "ltr" (left to right), "rtl" (right to left) plus "weak_ltr" and "weak_rtl".

Note: This feature requires the Pango text provider.

Note: Weak modes are currently not implemented in Kivy text layout, and have the same effect as setting strong mode.

New in version 1.10.1.

base_direction is an *OptionProperty* and defaults to None (autodetect RTL if possible, otherwise LTR).

border

Border used for *BorderImage* graphics instruction. Used with *background_normal* and *background_active*. Can be used for a custom background.

New in version 1.4.1.

It must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instruction for more information about how to use it.

border is a *ListProperty* and defaults to (4, 4, 4, 4).

cancel_selection()

Cancel current selection (if any).

copy(data="")

Copy the value provided in argument *data* into current clipboard. If data is not of type string it will be converted to string. If no data is provided then current selection if present is copied.

New in version 1.8.0.

cursor

Tuple of (row, col) values indicating the current cursor position. You can set a new (row, col) if you want to move the cursor. The scrolling area will be automatically updated to ensure that the cursor is visible inside the viewport.

cursor is an *AliasProperty*.

cursor_blink

This property is used to set whether the graphic cursor should blink or not.

Changed in version 1.10.1: *cursor_blink* has been refactored to enable switching the blinking on/off and the previous behavior has been moved to a private *_cursor_blink* property. The previous default value *False* has been changed to *True*.

cursor_blink is a *BooleanProperty* and defaults to *True*.

cursor_col

Current column of the cursor.

cursor_col is an *AliasProperty* to *cursor[0]*, read-only.

cursor_color

Current color of the cursor, in (r, g, b, a) format.

New in version 1.9.0.

cursor_color is a *ListProperty* and defaults to [1, 0, 0, 1].

cursor_index(cursor=None)

Return the cursor index in the text/value.

cursor_offset()

Get the cursor x offset on the current line.

cursor_pos

Current position of the cursor, in (x, y).

cursor_pos is an *AliasProperty*, read-only.

cursor_row

Current row of the cursor.

cursor_row is an *AliasProperty* to `cursor[1]`, read-only.

cursor_width

Current width of the cursor.

New in version 1.10.0.

cursor_width is a *NumericProperty* and defaults to '1sp'.

cut()

Copy current selection to clipboard then delete it from `TextInput`.

New in version 1.8.0.

delete_selection(*from_undo=False*)

Delete the current text selection (if any).

disabled_foreground_color

Current color of the foreground when disabled, in (r, g, b, a) format.

New in version 1.8.0.

disabled_foreground_color is a *ListProperty* and defaults to [0, 0, 0, 5] (50% transparent black).

do_backspace(*from_undo=False, mode='bkspc'*)

Do backspace operation from the current cursor position. This action might do several things:

- removing the current selection if available.
- removing the previous char and move the cursor back.
- do nothing, if we are at the start.

do_cursor_movement(*action, control=False, alt=False*)

Move the cursor relative to its current position. Action can be one of :

- `cursor_left`: move the cursor to the left
- `cursor_right`: move the cursor to the right
- `cursor_up`: move the cursor on the previous line
- `cursor_down`: move the cursor on the next line
- `cursor_home`: move the cursor at the start of the current line
- `cursor_end`: move the cursor at the end of current line
- `cursor_pgup`: move one "page" before
- `cursor_pgdown`: move one "page" after

In addition, the behavior of certain actions can be modified:

- `control + cursor_left`: move the cursor one word to the left
- `control + cursor_right`: move the cursor one word to the right
- `control + cursor_up`: scroll up one line
- `control + cursor_down`: scroll down one line
- `control + cursor_home`: go to beginning of text
- `control + cursor_end`: go to end of text

- alt + cursor_up: shift line(s) up
- alt + cursor_down: shift line(s) down

Changed in version 1.9.1.

do_redo()

Do redo operation.

New in version 1.3.0.

This action re-does any command that has been un-done by do_undo/ctrl+z. This function is automatically called when *ctrl+r* keys are pressed.

do_undo()

Do undo operation.

New in version 1.3.0.

This action un-does any edits that have been made since the last call to reset_undo(). This function is automatically called when *ctrl+z* keys are pressed.

font_context

Font context. *None* means the font is used in isolation, so you are guaranteed to be drawing with the TTF file resolved by *font_name*. Specifying a value here will load the font file into a named context, enabling fallback between all fonts in the same context. If a font context is set, you are not guaranteed that rendering will actually use the specified TTF file for all glyphs (Pango will pick the one it thinks is best).

If Kivy is linked against a system-wide installation of FontConfig, you can load the system fonts by specifying a font context starting with the special string *system://*. This will load the system fontconfig configuration, and add your application-specific fonts on top of it (this imposes a significant risk of family name collision, Pango may not use your custom font file, but pick one from the system)

Note: This feature requires the Pango text provider.

New in version 1.10.1.

font_context is a *StringProperty* and defaults to *None*.

font_family

Font family, this is only applicable when using *font_context* option. The specified font family will be requested, but note that it may not be available, or there could be multiple fonts registered with the same family. The value can be a family name (string) available in the font context (for example a system font in a *system://* context, or a custom font file added using *kivy.core.text.FontContextManager*). If set to *None*, font selection is controlled by the *font_name* setting.

Note: If using *font_name* to reference a custom font file, you should leave this as *None*. The family name is managed automatically in this case.

Note: This feature requires the Pango text provider.

New in version 1.10.1.

font_family is a *StringProperty* and defaults to *None*.

font_name

Filename of the font to use. The path can be absolute or relative. Relative paths are resolved by the *resource_find()* function.

Warning: Depending on your text provider, the font file may be ignored. However, you can mostly use this without problems.

If the font used lacks the glyphs for the particular language/symbols you are using, you will see '[]' blank box characters instead of the actual glyphs. The solution is to use a font that has the glyphs you need to display. For example, to display क, use a font like freesans.ttf that has the glyph.

font_name is a *StringProperty* and defaults to 'Roboto'. This value is taken from *Config*.

font_size

Font size of the text in pixels.

font_size is a *NumericProperty* and defaults to 15sp.

foreground_color

Current color of the foreground, in (r, g, b, a) format.

New in version 1.2.0.

foreground_color is a *ListProperty* and defaults to [0, 0, 0, 1] (black).

get_cursor_from_index(index)

Return the (row, col) of the cursor from text index.

get_cursor_from_xy(x, y)

Return the (row, col) of the cursor from an (x, y) position.

halign

Horizontal alignment of the text.

halign is an *OptionProperty* and defaults to 'auto'. Available options are : auto, left, center and right. Auto will attempt to autodetect horizontal alignment for RTL text (Pango only), otherwise it behaves like *left*.

New in version 1.10.1.

handle_image_left

Image used to display the Left handle on the TextInput for selection.

New in version 1.8.0.

handle_image_left is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/selector_left'.

handle_image_middle

Image used to display the middle handle on the TextInput for cursor positioning.

New in version 1.8.0.

handle_image_middle is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/selector_middle'.

handle_image_right

Image used to display the Right handle on the TextInput for selection.

New in version 1.8.0.

`handle_image_right` is a `StringProperty` and defaults to `'atlas://data/images/defaulttheme/selector_right'`.

hint_text

Hint text of the widget, shown if text is `''`.

New in version 1.6.0.

Changed in version 1.10.0: The property is now an `AliasProperty` and byte values are decoded to strings. The hint text will stay visible when the widget is focused.

`hint_text` a `AliasProperty` and defaults to `''`.

hint_text_color

Current color of the `hint_text` text, in (r, g, b, a) format.

New in version 1.6.0.

`hint_text_color` is a `ListProperty` and defaults to `[0.5, 0.5, 0.5, 1.0]` (grey).

input_filter

Filters the input according to the specified mode, if not `None`. If `None`, no filtering is applied.

New in version 1.9.0.

`input_filter` is an `ObjectProperty` and defaults to `None`. Can be one of `None`, `'int'` (string), or `'float'` (string), or a callable. If it is `'int'`, it will only accept numbers. If it is `'float'` it will also accept a single period. Finally, if it is a callable it will be called with two parameters; the string to be added and a bool indicating whether the string is a result of undo (`True`). The callable should return a new substring that will be used instead.

insert_text (*substring, from_undo=False*)

Insert new text at the current cursor position. Override this function in order to pre-process text for input validation.

keyboard_on_key_down (*window, keycode, text, modifiers*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input press. The parameters are the same as `kivy.core.window.WindowBase.on_key_down()`.

When overwriting the method in the derived widget, super should be called to enable tab cycling. If the derived widget wishes to use tab for its own purposes, it can call super after it has processed the character (if it does not wish to consume the tab).

Similar to other keyboard functions, it should return `True` if the key was consumed.

keyboard_on_key_up (*window, keycode*)

The method bound to the keyboard when the instance has focus.

When the instance becomes focused, this method is bound to the keyboard and will be called for every input release. The parameters are the same as `kivy.core.window.WindowBase.on_key_up()`.

When overwriting the method in the derived widget, super should be called to enable defocusing on escape. If the derived widget wishes to use escape for its own purposes, it can call super after it has processed the character (if it does not wish to consume the escape).

See `keyboard_on_key_down()`

keyboard_suggestions

If `True` provides auto suggestions on top of keyboard. This will only work if `input_type` is set to `text`.

New in version 1.8.0.

keyboard_suggestions is a *BooleanProperty* and defaults to True.

line_height

Height of a line. This property is automatically computed from the *font_name*, *font_size*. Changing the *line_height* will have no impact.

Note: *line_height* is the height of a single line of text. Use *minimum_height*, which also includes padding, to get the height required to display the text properly.

line_height is a *NumericProperty*, read-only.

line_spacing

Space taken up between the lines.

New in version 1.8.0.

line_spacing is a *NumericProperty* and defaults to 0.

minimum_height

Minimum height of the content inside the TextInput.

New in version 1.8.0.

minimum_height is a readonly *AliasProperty*.

Warning: *minimum_width* is calculated based on *width* therefore code like this will lead to an infinite loop:

```
<FancyTextInput>:
    height: self.minimum_height
    width: self.height
```

multiline

If True, the widget will be able show multiple lines of text. If False, the “enter” keypress will defocus the textinput instead of adding a new line.

multiline is a *BooleanProperty* and defaults to True.

on_double_tap()

This event is dispatched when a double tap happens inside TextInput. The default behavior is to select the word around the current cursor position. Override this to provide different behavior. Alternatively, you can bind to this event to provide additional functionality.

on_quad_touch()

This event is dispatched when four fingers are touching inside TextInput. The default behavior is to select all text. Override this to provide different behavior. Alternatively, you can bind to this event to provide additional functionality.

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_triple_tap()

This event is dispatched when a triple tap happens inside `TextInput`. The default behavior is to select the line around current cursor position. Override this to provide different behavior. Alternatively, you can bind to this event to provide additional functionality.

padding

Padding of the text: [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 1.7.0: Replaced `AliasProperty` with `VariableListProperty`.

[`padding`](#) is a [`VariableListProperty`](#) and defaults to [6, 6, 6, 6].

padding_x

Horizontal padding of the text: [padding_left, padding_right].

padding_x also accepts a one argument form [padding_horizontal].

[`padding_x`](#) is a [`VariableListProperty`](#) and defaults to [0, 0]. This might be changed by the current theme.

Deprecated since version 1.7.0: Use [`padding`](#) instead.

padding_y

Vertical padding of the text: [padding_top, padding_bottom].

padding_y also accepts a one argument form [padding_vertical].

[`padding_y`](#) is a [`VariableListProperty`](#) and defaults to [0, 0]. This might be changed by the current theme.

Deprecated since version 1.7.0: Use [`padding`](#) instead.

password

If True, the widget will display its characters as the character set in [`password_mask`](#).

New in version 1.2.0.

[`password`](#) is a [`BooleanProperty`](#) and defaults to False.

password_mask

Sets the character used to mask the text when [`password`](#) is True.

New in version 1.10.0.

[`password_mask`](#) is a [`StringProperty`](#) and defaults to '*'.

paste()

Insert text from system Clipboard into the [`TextInput`](#) at current cursor position.

New in version 1.8.0.

readonly

If True, the user will not be able to change the content of a textinput.

New in version 1.3.0.

[`readonly`](#) is a [`BooleanProperty`](#) and defaults to False.

replace_crlf

Automatically replace CRLF with LF.

New in version 1.9.1.

replace_crlf is a *BooleanProperty* and defaults to True.

reset_undo()

Reset undo and redo lists from memory.

New in version 1.3.0.

scroll_x

X scrolling value of the viewport. The scrolling is automatically updated when the cursor is moved or text changed. If there is no user input, the *scroll_x* and *scroll_y* properties may be changed.

scroll_x is a *NumericProperty* and defaults to 0.

scroll_y

Y scrolling value of the viewport. See *scroll_x* for more information.

scroll_y is a *NumericProperty* and defaults to 0.

select_all()

Select all of the text displayed in this TextInput.

New in version 1.4.0.

select_text(start, end)

Select a portion of text displayed in this TextInput.

New in version 1.4.0.

Parameters

start Index of `textinput.text` from where to start selection

end Index of `textinput.text` till which the selection should be displayed

selection_color

Current color of the selection, in (r, g, b, a) format.

Warning: The color should always have an “alpha” component less than 1 since the selection is drawn after the text.

selection_color is a *ListProperty* and defaults to [0.1843, 0.6549, 0.8313, .5].

selection_from

If a selection is in progress or complete, this property will represent the cursor index where the selection started.

Changed in version 1.4.0: *selection_from* is an *AliasProperty* and defaults to None, readonly.

selection_text

Current content selection.

selection_text is a *StringProperty* and defaults to “”, readonly.

selection_to

If a selection is in progress or complete, this property will represent the cursor index where the selection started.

Changed in version 1.4.0: *selection_to* is an *AliasProperty* and defaults to None, readonly.

suggestion_text

Shows a suggestion text at the end of the current line. The feature is useful for text autocompletion, and it does not implement validation (accepting the suggested text on enter etc.). This can also be used by the IME to setup the current word being edited.

New in version 1.9.0.

suggestion_text is a *StringProperty* and defaults to "".

tab_width

By default, each tab will be replaced by four spaces on the text input widget. You can set a lower or higher value.

tab_width is a *NumericProperty* and defaults to 4.

text

Text of the widget.

Creation of a simple hello world:

```
widget = TextInput(text='Hello world')
```

If you want to create the widget with an unicode string, use:

```
widget = TextInput(text=u'My unicode string')
```

text is an *AliasProperty*.

text_language

Language of the text, if None Pango will determine it from locale. This is an RFC-3066 format language tag (as a string), for example "en_US", "zh_CN", "fr" or "ja". This can impact font selection, metrics and rendering. For example, the same bytes of text can look different for *ur* and *ar* languages, though both use Arabic script.

Note: This feature requires the Pango text provider.

New in version 1.10.1.

text_language is a *StringProperty* and defaults to None.

text_validate_unfocus

If True, the `TextInput.on_text_validate()` event will unfocus the widget, therefore make it stop listening to the keyboard. When disabled, the `TextInput.on_text_validate()` event can be fired multiple times as the result of `TextInput` keeping the focus enabled.

New in version 1.10.1.

text_validate_unfocus is a *BooleanProperty* and defaults to True.

use_bubble

Indicates whether the cut/copy/paste bubble is used.

New in version 1.7.0.

use_bubble is a *BooleanProperty* and defaults to True on mobile OS's, False on desktop OS's.

use_handles

Indicates whether the selection handles are displayed.

New in version 1.8.0.

use_handles is a *BooleanProperty* and defaults to True on mobile OS's, False on desktop OS's.

write_tab

Whether the tab key should move focus to the next widget or if it should enter a tab in the *TextInput*. If *True* a tab will be written, otherwise, focus will move to the next widget.

New in version 1.9.0.

write_tab is a *BooleanProperty* and defaults to *True*.

33.46 Toggle button



The *ToggleButton* widget acts like a checkbox. When you touch or click it, the state toggles between 'normal' and 'down' (as opposed to a *Button* that is only 'down' as long as it is pressed).

Toggle buttons can also be grouped to make radio buttons - only one button in a group can be in a 'down' state. The group name can be a string or any other hashable Python object:

```
btn1 = ToggleButton(text='Male', group='sex',)
btn2 = ToggleButton(text='Female', group='sex', state='down')
btn3 = ToggleButton(text='Mixed', group='sex')
```

Only one of the buttons can be 'down'/checked at the same time.

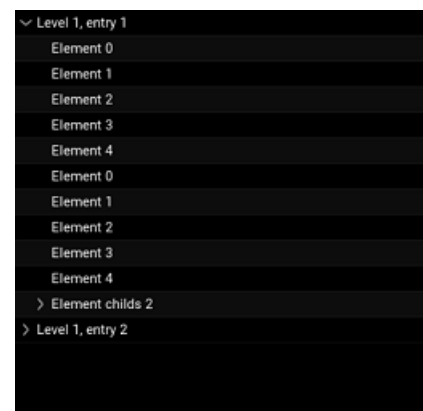
To configure the *ToggleButton*, you can use the same properties that you can use for a *Button* class.

class *kivy.uix.togglebutton.ToggleButton*(**kwargs)

Bases: *kivy.uix.behaviors.togglebutton.ToggleButtonBehavior*, *kivy.uix.button.Button*

Toggle button class, see module documentation for more information.

33.47 Tree View



New in version 1.0.4.

TreeView is a widget used to represent a tree structure. It is currently very basic, supporting a minimal feature set.

33.47.1 Introduction

A *TreeView* is populated with *TreeNode* instances, but you cannot use a *TreeNode* directly. You must combine it with another widget, such as *Label*, *Button* or even your own widget. The *TreeView* always creates a default root node, based on *TreeViewLabel*.

TreeNode is a class object containing needed properties for serving as a tree node. Extend *TreeNode* to create custom node types for use with a *TreeView*.

For constructing your own subclass, follow the pattern of *TreeViewLabel* which combines a *Label* and a *TreeNode*, producing a *TreeViewLabel* for direct use in a *TreeView* instance.

To use the *TreeViewLabel* class, you could create two nodes directly attached to root:

```
tv = TreeView()
tv.add_node(TreeViewLabel(text='My first item'))
tv.add_node(TreeViewLabel(text='My second item'))
```

Or, create two nodes attached to a first:

```
tv = TreeView()
n1 = tv.add_node(TreeViewLabel(text='Item 1'))
tv.add_node(TreeViewLabel(text='SubItem 1'), n1)
tv.add_node(TreeViewLabel(text='SubItem 2'), n1)
```

If you have a large tree structure, perhaps you would need a utility function to populate the tree view:

```
def populate_tree_view(tree_view, parent, node):
    if parent is None:
        tree_node = tree_view.add_node(TreeViewLabel(text=node['node_id'],
                                                    is_open=True))
    else:
        tree_node = tree_view.add_node(TreeViewLabel(text=node['node_id'],
                                                    is_open=True), parent)

    for child_node in node['children']:
        populate_tree_view(tree_view, tree_node, child_node)

tree = {'node_id': '1',
        'children': [{'node_id': '1.1',
                        'children': [{'node_id': '1.1.1',
                                      'children': [{'node_id': '1.1.1.1',
                                                      'children': []}]},
                                      {'node_id': '1.1.2',
                                      'children': []},
                                      {'node_id': '1.1.3',
                                      'children': []}]},
                        {'node_id': '1.2',
                        'children': []}]}

class TreeWidget(FloatLayout):
    def __init__(self, **kwargs):
        super(TreeWidget, self).__init__(**kwargs)

        tv = TreeView(root_options=dict(text='Tree One'),
                        hide_root=False,
                        indent_level=4)
```

(continues on next page)

```

    populate_tree_view(tv, None, tree)

    self.add_widget(tv)

```

The root widget in the tree view is opened by default and has text set as 'Root'. If you want to change that, you can use the `TreeView.root_options` property. This will pass options to the root widget:

```
tv = TreeView(root_options=dict(text='My root label'))
```

33.47.2 Creating Your Own Node Widget

For a button node type, combine a `Button` and a `TreeNode` as follows:

```

class TreeViewButton(Button, TreeNode):
    pass

```

You must know that, for a given node, only the `size_hint_x` will be honored. The allocated width for the node will depend of the current width of the `TreeView` and the level of the node. For example, if a node is at level 4, the width allocated will be:

```
treeview.width - treeview.indent_start - treeview.indent_level * node.level
```

You might have some trouble with that. It is the developer's responsibility to correctly handle adapting the graphical representation nodes, if needed.

class `kivy.uix.treeview.TreeView`(***kwargs*)

Bases: `kivy.uix.widget.Widget`

`TreeView` class. See module documentation for more information.

Events

on_node_expand: (node,) Fired when a node is being expanded

on_node_collapse: (node,) Fired when a node is being collapsed

add_node(node, parent=None)

Add a new node to the tree.

Parameters

node: instance of a `TreeNode` Node to add into the tree

parent: instance of a `TreeNode`, defaults to `None` Parent node to attach the new node. If `None`, it is added to the `root` node.

Returns the node `node`.

deselect_node(**args*)

Deselect any selected node.

New in version 1.10.0.

get_node_at_pos(pos)

Get the node at the position (x, y).

hide_root

Use this property to show /hide the initial root node. If `True`, the root node will be appear as a closed node.

`hide_root` is a `BooleanProperty` and defaults to `False`.

indent_level

Width used for the indentation of each level except the first level.

Computation of indent for each level of the tree is:

```
indent = indent_start + level * indent_level
```

indent_level is a *NumericProperty* and defaults to 16.

indent_start

Indentation width of the level 0 / root node. This is mostly the initial size to accommodate a tree icon (collapsed / expanded). See *indent_level* for more information about the computation of level indentation.

indent_start is a *NumericProperty* and defaults to 24.

iterate_all_nodes (*node=None*)

Generator to iterate over all nodes from *node* and down whether expanded or not. If *node* is *None*, the generator start with *root*.

iterate_open_nodes (*node=None*)

Generator to iterate over all the expended nodes starting from *node* and down. If *node* is *None*, the generator start with *root*.

To get all the open nodes:

```
treeview = TreeView()  
# ... add nodes ...  
for node in treeview.iterate_open_nodes():  
    print(node)
```

load_func

Callback to use for asynchronous loading. If set, asynchronous loading will be automatically done. The callback must act as a Python generator function, using yield to send data back to the treeview.

The callback should be in the format:

```
def callback(treeview, node):  
    for name in ('Item 1', 'Item 2'):  
        yield TreeViewLabel(text=name)
```

load_func is a *ObjectProperty* and defaults to None.

minimum_height

Minimum height needed to contain all children.

New in version 1.0.9.

minimum_height is a *NumericProperty* and defaults to 0.

minimum_size

Minimum size needed to contain all children.

New in version 1.0.9.

minimum_size is a *ReferenceListProperty* of (*minimum_width*, *minimum_height*) properties.

minimum_width

Minimum width needed to contain all children.

New in version 1.0.9.

minimum_width is a *NumericProperty* and defaults to 0.

on_touch_down(*touch*)

Receive a touch down event.

Parameters

touch: **MotionEvent** class Touch received. The touch is in parent coordinates. See *relative layout* 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.

remove_node(*node*)

Removes a node from the tree.

New in version 1.0.7.

Parameters

node: instance of a **TreeNode** Node to remove from the tree. If *node* is *root*, it is not removed.

root

Root node.

By default, the root node widget is a **TreeViewLabel** with text 'Root'. If you want to change the default options passed to the widget creation, use the *root_options* property:

```
treeview = TreeView(root_options={
    'text': 'Root directory',
    'font_size': 15})
```

root_options will change the properties of the **TreeViewLabel** instance. However, you cannot change the class used for root node yet.

root is an **AliasProperty** and defaults to None. It is read-only. However, the content of the widget can be changed.

root_options

Default root options to pass for root widget. See *root* property for more information about the usage of *root_options*.

root_options is an **ObjectProperty** and defaults to {}.

select_node(*node*)

Select a node in the tree.

selected_node

Node selected by **TreeView.select_node()** or by touch.

selected_node is a **AliasProperty** and defaults to None. It is read-only.

toggle_node(*node*)

Toggle the state of the node (open/collapsed).

exception kivy.uix.treeview.**TreeViewException**

Bases: Exception

Exception for errors in the **TreeView**.

class kivy.uix.treeview.**TreeViewLabel**(**kwargs)

Bases: **kivy.uix.label.Label**, **kivy.uix.treeview.TreeNode**

Combines a **Label** and a **TreeNode** to create a **TreeViewLabel** that can be used as a text node in the tree.

See module documentation for more information.

class kivy.uix.treeview.**TreeNode**(**kwargs)

Bases: `builtins.object`

TreeNode class, used to build a node class for a TreeView object.

color_selected

Background color of the node when the node is selected.

color_selected is a *ListProperty* and defaults to `[.1, .1, .1, 1]`.

even_color

Background color of even nodes when the node is not selected.

bg_color is a *ListProperty* and defaults to `[.5, .5, .5, .1]`.

is_leaf

Boolean to indicate whether this node is a leaf or not. Used to adjust the graphical representation.

is_leaf is a *BooleanProperty* and defaults to `True`. It is automatically set to `False` when child is added.

is_loaded

Boolean to indicate whether this node is already loaded or not. This property is used only if the *TreeView* uses asynchronous loading.

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

is_open

Boolean to indicate whether this node is opened or not, in case there are child nodes. This is used to adjust the graphical representation.

Warning: This property is automatically set by the *TreeView*. You can read but not write it.

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

is_selected

Boolean to indicate whether this node is selected or not. This is used adjust the graphical representation.

Warning: This property is automatically set by the *TreeView*. You can read but not write it.

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

level

Level of the node.

level is a *NumericProperty* and defaults to `-1`.

no_selection

Boolean used to indicate whether selection of the node is allowed or not.

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

nodes

List of nodes. The nodes list is different than the children list. A node in the nodes list represents a node on the tree. An item in the children list represents the widget associated with the node.

Warning: This property is automatically set by the *TreeView*. You can read but not write it.

nodes is a *ListProperty* and defaults to [].

odd

This property is set by the TreeView widget automatically and is read-only.

odd is a *BooleanProperty* and defaults to False.

odd_color

Background color of odd nodes when the node is not selected.

odd_color is a *ListProperty* and defaults to [1., 1., 1., 0.].

parent_node

Parent node. This attribute is needed because the *parent* can be None when the node is not displayed.

New in version 1.0.7.

parent_node is an *ObjectProperty* and defaults to None.

33.48 Video

The *Video* widget is used to display video files and streams. Depending on your Video core provider, platform, and plugins, you will be able to play different formats. For example, the pygame video provider only supports MPEG1 on Linux and OSX. GStreamer is more versatile, and can read many video containers and codecs such as MKV, OGV, AVI, MOV, FLV (if the correct gstreamer plugins are installed). Our *VideoBase* implementation is used under the hood.

Video loading is asynchronous - many properties are not available until the video is loaded (when the texture is created):

```
def on_position_change(instance, value):
    print('The position in the video is', value)
def on_duration_change(instance, value):
    print('The duration of the video is', value)
video = Video(source='PandaSneezes.avi')
video.bind(position=on_position_change,
           duration=on_duration_change)
```

class kivy.uix.video.**Video**(**kwargs)

Bases: *kivy.uix.image.Image*

Video class. See module documentation for more information.

duration

Duration of the video. The duration defaults to -1, and is set to a real duration when the video is loaded.

duration is a *NumericProperty* and defaults to -1.

eos

Boolean, indicates whether the video has finished playing or not (reached the end of the stream).

eos is a *BooleanProperty* and defaults to False.

loaded

Boolean, indicates whether the video is loaded and ready for playback or not.

New in version 1.6.0.

loaded is a *BooleanProperty* and defaults to False.

options

Options to pass at Video core object creation.

New in version 1.0.4.

options is an *kivy.properties.ObjectProperty* and defaults to {}.

play

Deprecated since version 1.4.0: Use *state* instead.

Boolean, indicates whether the video is playing or not. You can start/stop the video by setting this property:

```
# start playing the video at creation
video = Video(source='movie.mkv', play=True)

# create the video, and start later
video = Video(source='movie.mkv')
# and later
video.play = True
```

play is a *BooleanProperty* and defaults to False.

Deprecated since version 1.4.0: Use *state* instead.

position

Position of the video between 0 and *duration*. The position defaults to -1 and is set to a real position when the video is loaded.

position is a *NumericProperty* and defaults to -1.

seek(percent, precise=True)

Change the position to a percentage of duration.

Parameters

percent: float or int Position to seek, must be between 0-1.

precise: bool, defaults to True Precise seeking is slower, but seeks to exact requested percent.

Warning: Calling seek() before the video is loaded has no effect.

New in version 1.2.0.

Changed in version 1.10.1: The *precise* keyword argument has been added.

state

String, indicates whether to play, pause, or stop the video:

```
# start playing the video at creation
video = Video(source='movie.mkv', state='play')

# create the video, and start later
video = Video(source='movie.mkv')
# and later
video.state = 'play'
```

`state` is an *OptionProperty* and defaults to 'stop'.

unload()

Unload the video. The playback will be stopped.

New in version 1.8.0.

volume

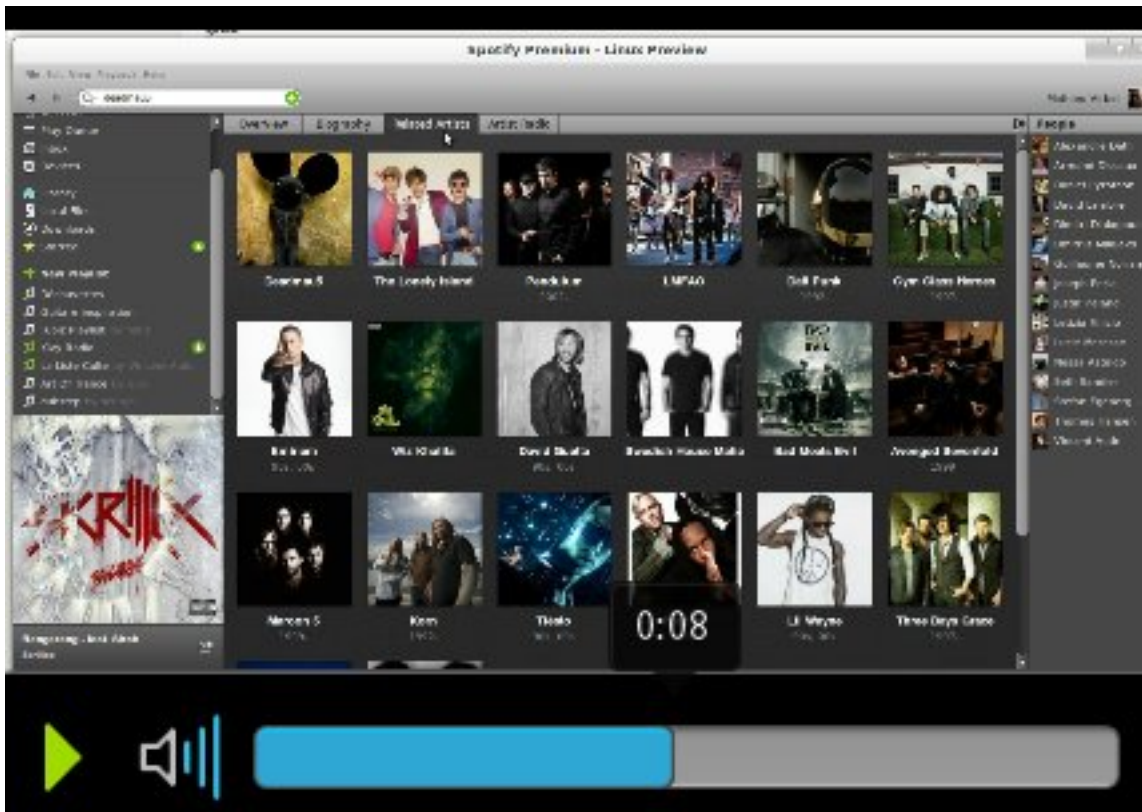
Volume of the video, in the range 0-1. 1 means full volume, 0 means mute.

volume is a *NumericProperty* and defaults to 1.

33.49 Video player

New in version 1.2.0.

The video player widget can be used to play video and let the user control the play/pausing, volume and position. The widget cannot be customized much because of the complex assembly of numerous base widgets.



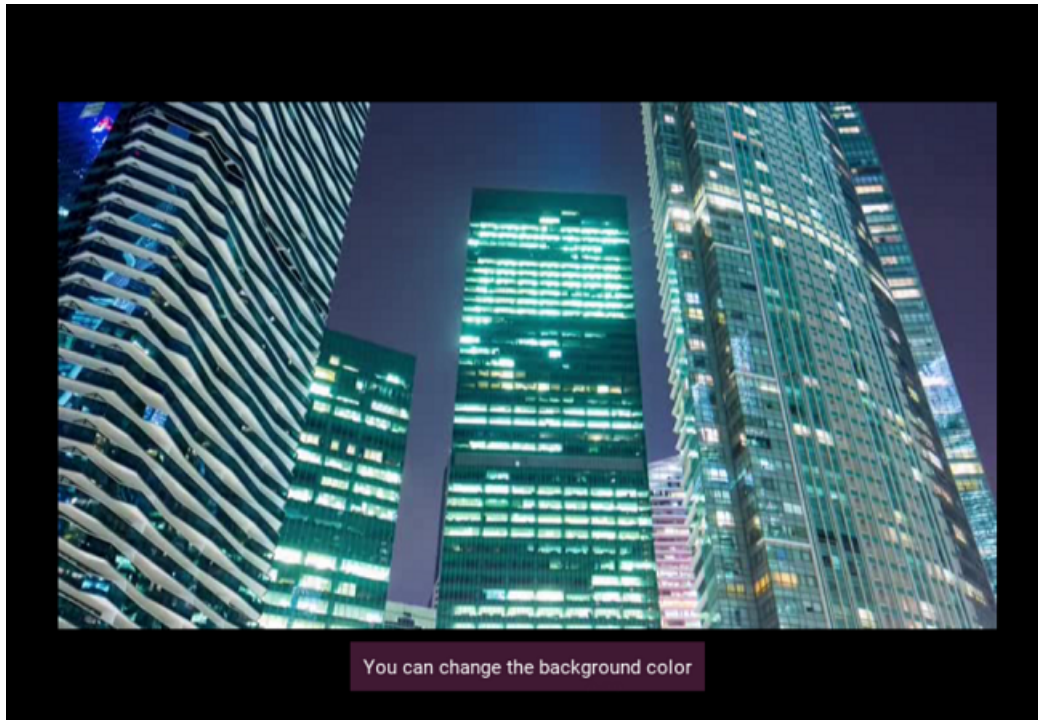
33.49.1 Annotations

If you want to display text at a specific time and for a certain duration, consider annotations. An annotation file has a “.jsa” extension. The player will automatically load the associated annotation file if it exists.

An annotation file is JSON-based, providing a list of label dictionary items. The key and value must match one of the **VideoPlayerAnnotation** items. For example, here is a short version of a jsa file that you can find in *examples/widgets/cityCC0.jsa*:

```
[
    {"start": 0, "duration": 2,
     "text": "This is an example of annotation"},
    {"start": 2, "duration": 2,
     "bgcolor": [0.5, 0.2, 0.4, 0.5],
     "text": "You can change the background color"}
]
```

For our cityCC0.mpg example, the result will be:



If you want to experiment with annotation files, test with:

```
python -m kivy.uix.videoplayer examples/widgets/cityCC0.mpg
```

33.49.2 Fullscreen

The video player can play the video in fullscreen, if *VideoPlayer.allow_fullscreen* is activated by a double-tap on the video. By default, if the video is smaller than the Window, it will be not stretched.

You can allow stretching by passing custom options to a *VideoPlayer* instance:

```
player = VideoPlayer(source='myvideo.avi', state='play',
    options={'allow_stretch': True})
```

33.49.3 End-of-stream behavior

You can specify what happens when the video has finished playing by passing an *eos* (end of stream) directive to the underlying *VideoBase* class. *eos* can be one of 'stop', 'pause' or 'loop' and defaults to 'stop'. For example, in order to loop the video:

```
player = VideoPlayer(source='myvideo.avi', state='play',
    options={'eos': 'loop'})
```

Note: The *eos* property of the VideoBase class is a string specifying the end-of-stream behavior. This property differs from the *eos* properties of the *VideoPlayer* and *Video* classes, whose *eos* property is simply a boolean indicating that the end of the file has been reached.

class `kivy.uix.videoplayer.VideoPlayer(**kwargs)`

Bases: *kivy.uix.gridlayout.GridLayout*

VideoPlayer class. See module documentation for more information.

allow_fullscreen

By default, you can double-tap on the video to make it fullscreen. Set this property to False to prevent this behavior.

allow_fullscreen is a *BooleanProperty* defaults to True.

annotations

If set, it will be used for reading annotations box.

annotations is a *StringProperty* and defaults to "".

duration

Duration of the video. The duration defaults to -1 and is set to the real duration when the video is loaded.

duration is a *NumericProperty* and defaults to -1.

fullscreen

Switch to fullscreen view. This should be used with care. When activated, the widget will remove itself from its parent, remove all children from the window and will add itself to it. When fullscreen is unset, all the previous children are restored and the widget is restored to its previous parent.

Warning: The re-add operation doesn't care about the index position of its children within the parent.

fullscreen is a *BooleanProperty* and defaults to False.

image_loading

Image filename used when the video is loading.

image_loading is a *StringProperty* and defaults to 'data/images/image-loading.gif'.

image_overlay_play

Image filename used to show a "play" overlay when the video has not yet started.

image_overlay_play is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/player-play-overlay'.

image_pause

Image filename used for the "Pause" button.

image_pause is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/media-playback-pause'.

image_play

Image filename used for the "Play" button.

image_play is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/media-playback-start'.

image_stop

Image filename used for the “Stop” button.

image_stop is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/media-playback-stop’.

image_volumehigh

Image filename used for the volume icon when the volume is high.

image_volumehigh is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/audio-volume-high’.

image_volumelow

Image filename used for the volume icon when the volume is low.

image_volumelow is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/audio-volume-low’.

image_volumemedium

Image filename used for the volume icon when the volume is medium.

image_volumemedium is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/audio-volume-medium’.

image_volumemuted

Image filename used for the volume icon when the volume is muted.

image_volumemuted is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/audio-volume-muted’.

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.

options

Optional parameters can be passed to a *Video* instance with this property.

options a *DictProperty* and defaults to {}.

play

Deprecated since version 1.4.0: Use *state* instead.

Boolean, indicates whether the video is playing or not. You can start/stop the video by setting this property:

```
# start playing the video at creation
video = VideoPlayer(source='movie.mkv', play=True)

# create the video, and start later
video = VideoPlayer(source='movie.mkv')
# and later
video.play = True
```

play is a *BooleanProperty* and defaults to False.

position

Position of the video between 0 and *duration*. The position defaults to -1 and is set to the real position when the video is loaded.

position is a *NumericProperty* and defaults to -1.

seek(*percent*, *precise*=True)

Change the position to a percentage of duration.

Parameters

percent: **float or int** Position to seek, must be between 0-1.

precise: **bool, defaults to True** Precise seeking is slower, but seeks to exact requested percent.

Warning: Calling seek() before the video is loaded has no effect.

New in version 1.2.0.

Changed in version 1.10.1: The *precise* keyword argument has been added.

source

Source of the video to read.

source is a *StringProperty* and defaults to "".

Changed in version 1.4.0.

state

String, indicates whether to play, pause, or stop the video:

```
# start playing the video at creation
video = VideoPlayer(source='movie.mkv', state='play')

# create the video, and start later
video = VideoPlayer(source='movie.mkv')
# and later
video.state = 'play'
```

state is an *OptionProperty* and defaults to 'stop'.

thumbnail

Thumbnail of the video to show. If None, VideoPlayer will try to find the thumbnail from the *source* + '.png'.

thumbnail a *StringProperty* and defaults to "".

Changed in version 1.4.0.

volume

Volume of the video in the range 0-1. 1 means full volume and 0 means mute.

volume is a *NumericProperty* and defaults to 1.

class kivy.uix.videoplayer.**VideoPlayerAnnotation**(**kwargs)

Bases: *kivy.uix.label.Label*

Annotation class used for creating annotation labels.

Additional keys are available:

- **bgcolor**: [r, g, b, a] - background color of the text box
- **bgsource**: 'filename' - background image used for the background text box
- **border**: (n, e, s, w) - border used for the background image

duration

Duration of the annotation.

duration is a *NumericProperty* and defaults to 1.

start

Start time of the annotation.

start is a *NumericProperty* and defaults to 0.

33.50 VKeyboard



New in version 1.0.8.

VKeyboard is an onscreen keyboard for Kivy. Its operation is intended to be transparent to the user. Using the widget directly is NOT recommended. Read the section *Request keyboard* first.

33.50.1 Modes

This virtual keyboard has a docked and free mode:

- docked mode (*VKeyboard.docked* = True) Generally used when only one person is using the computer, like a tablet or personal computer etc.
- free mode: (*VKeyboard.docked* = False) Mostly for multitouch surfaces. This mode allows multiple virtual keyboards to be used on the screen.

If the docked mode changes, you need to manually call *VKeyboard.setup_mode()* otherwise the change will have no impact. During that call, the VKeyboard, implemented on top of a *Scatter*, will change the behavior of the scatter and position the keyboard near the target (if target and docked mode is set).

33.50.2 Layouts

The virtual keyboard is able to load a custom layout. If you create a new layout and put the JSON in `<kivy_data_dir>/keyboards/<layoutid>.json`, you can load it by setting *VKeyboard.layout* to your layoutid.

The JSON must be structured like this:

```
{
    "title": "Title of your layout",
    "description": "Description of your layout",
    "cols": 15,
    "rows": 5,
    ...
}
```

Then, you need to describe the keys in each row, for either a “normal”, “shift” or a “special” (added in version 1.9.0) mode. Keys for this row data must be named *normal_<row>*, *shift_<row>* and *special_<row>*. Replace *row* with the row number. Inside each row, you will describe the key. A key is a 4 element list in the format:

```
[ <text displayed on the keyboard>, <text to put when the key is pressed>,
  <text that represents the keycode>, <size of cols> ]
```

Here are example keys:

```
# f key
["f", "f", "f", 1]
# capslock
["", " ", "tab", 1.5]
```

Finally, complete the JSON:

```
{
    ...
    "normal_1": [
        ["`", "`", "`", 1],      ["1", "1", "1", 1],      ["2", "2", "2", 1],
        ["3", "3", "3", 1],      ["4", "4", "4", 1],      ["5", "5", "5", 1],
        ["6", "6", "6", 1],      ["7", "7", "7", 1],      ["8", "8", "8", 1],
        ["9", "9", "9", 1],      ["0", "0", "0", 1],      ["+", "+", "+", 1],
        ["=", "=", "=", 1],      ["", null, "backspace", 2]
    ],
    "shift_1": [ ... ],
    "normal_2": [ ... ],
    "special_2": [ ... ],
    ...
}
```

33.50.3 Request Keyboard

The instantiation of the virtual keyboard is controlled by the configuration. Check *keyboard_mode* and *keyboard_layout* in the [Configuration object](#).

If you intend to create a widget that requires a keyboard, do not use the virtual keyboard directly, but prefer to use the best method available on the platform. Check the [request_keyboard\(\)](#) method in the [Window](#).

If you want a specific layout when you request the keyboard, you should write something like this (from 1.8.0, numeric.json can be in the same directory as your main.py):

```
keyboard = Window.request_keyboard(
    self._keyboard_close, self)
if keyboard.widget:
```

(continues on next page)


```
vkeyboard = self._keyboard.widget
vkeyboard.layout = 'numeric.json'
```

class `kivy.uix.vkeyboard.VKeyboard`(**kwargs)

Bases: `kivy.uix.scatter.Scatter`

VKeyboard is an onscreen keyboard with multitouch support. Its layout is entirely customizable and you can switch between available layouts using a button in the bottom right of the widget.

Events

on_key_down: keycode, internal, modifiers Fired when the keyboard received a key down event (key press).

on_key_up: keycode, internal, modifiers Fired when the keyboard received a key up event (key release).

available_layouts

Dictionary of all available layouts. Keys are the layout ID, and the value is the JSON (translated into a Python object).

available_layouts is a *DictProperty* and defaults to {}.

background

Filename of the background image.

background is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard_background`.

background_border

Background image border. Used for controlling the *border* property of the background.

background_border is a *ListProperty* and defaults to [16, 16, 16, 16]

background_color

Background color, in the format (r, g, b, a). If a background is set, the color will be combined with the background texture.

background_color is a *ListProperty* and defaults to [1, 1, 1, 1].

background_disabled

Filename of the background image when the vkeyboard is disabled.

New in version 1.8.0.

background_disabled is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard__disabled_background`.

callback

Callback can be set to a function that will be called if the VKeyboard is closed by the user.

target is an *ObjectProperty* instance and defaults to None.

collide_margin(x, y)

Do a collision test, and return True if the (x, y) is inside the vkeyboard margin.

docked

Indicate whether the VKeyboard is docked on the screen or not. If you change it, you must manually call *setup_mode()* otherwise it will have no impact. If the VKeyboard is created by the Window, the docked mode will be automatically set by the configuration, using the *keyboard_mode* token in [kivy] section.

docked is a *BooleanProperty* and defaults to False.

font_size

`font_size`, specifies the size of the text on the virtual keyboard keys. It should be kept within limits to ensure the text does not extend beyond the bounds of the key or become too small to read.

New in version 1.10.0.

`font_size` is a *NumericProperty* and defaults to 20.

key_background_color

Key background color, in the format (r, g, b, a). If a key background is set, the color will be combined with the key background texture.

`key_background_color` is a *ListProperty* and defaults to [1, 1, 1, 1].

key_background_down

Filename of the key background image for use when a touch is active on the widget.

`key_background_down` is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard_key_down`.

key_background_normal

Filename of the key background image for use when no touches are active on the widget.

`key_background_normal` is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard_key_normal`.

key_border

Key image border. Used for controlling the *border* property of the key.

`key_border` is a *ListProperty* and defaults to [16, 16, 16, 16]

key_disabled_background_normal

Filename of the key background image for use when no touches are active on the widget and vkeyboard is disabled.

New in version 1.8.0.

`key_disabled_background_normal` is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard_disabled_key_normal`.

key_margin

Key margin, used to create space between keys. The margin is composed of four values, in pixels:

```
key_margin = [top, right, bottom, left]
```

`key_margin` is a *ListProperty* and defaults to [2, 2, 2, 2]

layout

Layout to use for the VKeyboard. By default, it will be the layout set in the configuration, according to the `keyboard_layout` in [kivy] section.

Changed in version 1.8.0: If layout is a .json filename, it will loaded and added to the available_layouts.

`layout` is a *StringProperty* and defaults to None.

layout_path

Path from which layouts are read.

`layout` is a *StringProperty* and defaults to `<kivy_data_dir>/keyboards/`

margin_hint

Margin hint, used as spacing between keyboard background and keys content. The margin is composed of four values, between 0 and 1:

```
margin_hint = [top, right, bottom, left]
```

The margin hints will be multiplied by width and height, according to their position.

margin_hint is a *ListProperty* and defaults to [.05, .06, .05, .06]

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.

refresh(*force=False*)

(internal) Recreate the entire widget and graphics according to the selected layout.

setup_mode(**largs*)

Call this method when you want to readjust the keyboard according to options: *docked* or not, with attached *target* or not:

- If *docked* is True, it will call *setup_mode_dock()*
- If *docked* is False, it will call *setup_mode_free()*

Feel free to overload these methods to create new positioning behavior.

setup_mode_dock(**largs*)

Setup the keyboard in docked mode.

Dock mode will reset the rotation, disable translation, rotation and scale. Scale and position will be automatically adjusted to attach the keyboard to the bottom of the screen.

Note: Don't call this method directly, use *setup_mode()* instead.

setup_mode_free()

Setup the keyboard in free mode.

Free mode is designed to let the user control the position and orientation of the keyboard. The only real usage is for a multiuser environment, but you might found other ways to use it. If a *target* is set, it will place the vkeyboard under the target.

Note: Don't call this method directly, use *setup_mode()* instead.

target

Target widget associated with the VKeyboard. If set, it will be used to send keyboard events. If the VKeyboard mode is "free", it will also be used to set the initial position.

target is an *ObjectProperty* instance and defaults to None.

33.51 Widget class

The *Widget* class is the base class required for creating Widgets. This widget class was designed with a couple of principles in mind:

- *Event Driven*

Widget interaction is built on top of events that occur. If a property changes, the widget can respond to the change in the 'on_<propname>' callback. If nothing changes, nothing will be done. That's the main goal of the *Property* class.

- *Separation Of Concerns (the widget and its graphical representation)*

Widgets don't have a *draw()* method. This is done on purpose: The idea is to allow you to create your own graphical representation outside the widget class. Obviously you can still use all the available properties to do that, so that your representation properly reflects the widget's current state. Every widget has its own *Canvas* that you can use to draw. This separation allows Kivy to run your application in a very efficient manner.

- *Bounding Box / Collision*

Often you want to know if a certain point is within the bounds of your widget. An example would be a button widget where you only want to trigger an action when the button itself is actually touched. For this, you can use the *collide_point()* method, which will return True if the point you pass to it is inside the axis-aligned bounding box defined by the widget's position and size. If a simple AABB is not sufficient, you can override the method to perform the collision checks with more complex shapes, e.g. a polygon. You can also check if a widget collides with another widget with *collide_widget()*.

We also have some default values and behaviors that you should be aware of:

- A *Widget* is not a *Layout*: it will not change the position or the size of its children. If you want control over positioning or sizing, use a *Layout*.
- The default size of a widget is (100, 100). This is only changed if the parent is a *Layout*. For example, if you add a *Label* inside a *Button*, the label will not inherit the button's size or position because the button is not a *Layout*: it's just another *Widget*.
- The default size_hint is (1, 1). If the parent is a *Layout*, then the widget size will be the parent layout's size.
- *on_touch_down()*, *on_touch_move()*, *on_touch_up()* don't do any sort of collisions. If you want to know if the touch is inside your widget, use *collide_point()*.

33.51.1 Using Properties

When you read the documentation, all properties are described in the format:

```
<name> is a <property class> and defaults to <default value>.
```

e.g.

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

If you want to be notified when the pos attribute changes, i.e. when the widget moves, you can bind your own callback function like this:

```
def callback_pos(instance, value):  
    print('The widget', instance, 'moved to', value)
```

(continues on next page)

```
wid = Widget()
wid.bind(pos=callback_pos)
```

Read more about *Properties*.

33.51.2 Basic drawing

Widgets support a range of drawing instructions that you can use to customize the look of your widgets and layouts. For example, to draw a background image for your widget, you can do the following:

```
def redraw(self, args):
    self.bg_rect.size = self.size
    self.bg_rect.pos = self.pos

widget = Widget()
with widget.canvas:
    widget.bg_rect = Rectangle(source="cover.jpg", pos=self.pos, size=self.size)
widget.bind(pos=redraw, size=redraw)
```

To draw a background in kv:

```
Widget:
    canvas:
        Rectangle:
            source: "cover.jpg"
            size: self.size
            pos: self.pos
```

These examples only scratch the surface. Please see the *kivy.graphics* documentation for more information.

33.51.3 Widget touch event bubbling

When you catch touch events between multiple widgets, you often need to be aware of the order in which these events are propagated. In Kivy, events bubble up from the first child upwards through the other children. If a widget has children, the event is passed through its children before being passed on to the widget after it.

As the *on_touch_up()* method inserts widgets at index 0 by default, this means the event goes from the most recently added widget back to the first one added. Consider the following:

```
box = BoxLayout()
box.add_widget(Label(text="a"))
box.add_widget(Label(text="b"))
box.add_widget(Label(text="c"))
```

The label with text “c” gets the event first, “b” second and “a” last. You can reverse this order by manually specifying the index:

```
box = BoxLayout()
box.add_widget(Label(text="a"), index=0)
box.add_widget(Label(text="b"), index=1)
box.add_widget(Label(text="c"), index=2)
```

Now the order would be “a”, “b” then “c”. One thing to keep in mind when using kv is that declaring a widget uses the *add_widget()* method for insertion. Hence, using

```

BoxLayout:
    MyLabel:
        text: "a"
    MyLabel:
        text: "b"
    MyLabel:
        text: "c"

```

would result in the event order “c”, “b” then “a” as “c” was actually the last added widget. It thus has index 0, “b” index 1 and “a” index 2. Effectively, the child order is the reverse of its listed order.

This ordering is the same for the `on_touch_move()` and `on_touch_up()` events.

In order to stop this event bubbling, a method can return `True`. This tells Kivy the event has been handled and the event propagation stops. For example:

```

class MyWidget(Widget):
    def on_touch_down(self, touch):
        If <some_condition>:
            # Do stuff here and kill the event
            return True
        else:
            return super(MyWidget, self).on_touch_down(touch)

```

This approach gives you good control over exactly how events are dispatched and managed. Sometimes, however, you may wish to let the event be completely propagated before taking action. You can use the `Clock` to help you here:

```

class MyWidget(Label):
    def on_touch_down(self, touch, after=False):
        if after:
            print "Fired after the event has been dispatched!"
        else:
            Clock.schedule_once(lambda dt: self.on_touch_down(touch, True))
            return super(MyWidget, self).on_touch_down(touch)

```

33.51.4 Usage of `Widget.center`, `Widget.right`, and `Widget.top`

A common mistake when using one of the computed properties such as `Widget.right` is to use it to make a widget follow its parent with a KV rule such as `right: self.parent.right`. Consider, for example:

```

FloatLayout:
    id: layout
    width: 100
    Widget:
        id: wid
        right: layout.right

```

The (mistaken) expectation is that this rule ensures that `wid`’s `right` will always be whatever `layout`’s `right` is - that is `wid.right` and `layout.right` will always be identical. In actual fact, this rule only says that “whenever `layout`’s `right` changes, `wid`’s `right` will be set to that value”. The difference being that as long as `layout.right` doesn’t change, `wid.right` could be anything, even a value that will make them different.

Specifically, for the KV code above, consider the following example:

```

>>> print(layout.right, wid.right)
(100, 100)

```

(continues on next page)

```
>>> wid.x = 200
>>> print(layout.right, wid.right)
(100, 300)
```

As can be seen, initially they are in sync, however, when we change *wid.x* they go out of sync because *layout.right* is not changed and the rule is not triggered.

The proper way to make the widget follow its parent's right is to use *Widget.pos_hint*. If instead of *right: layout.right* we did *pos_hint: {'right': 1}*, then the widget's right will always be set to be at the parent's right at each layout update.

class kivy.uix.widget.Widget(**kwargs)

Bases: kivy.uix.widget.WidgetBase

Widget class. See module documentation for more information.

Events

on_touch_down: (touch,) Fired when a new touch event occurs. *touch* is the touch object.

on_touch_move: (touch,) Fired when an existing touch moves. *touch* is the touch object.

on_touch_up: (touch,) Fired when an existing touch disappears. *touch* is the touch object.

on_kv_post: (base_widget,) Fired after all the kv rules associated with the widget and all other widgets that are in any of those rules have had all their kv rules applied. *base_widget* is the base-most widget whose instantiation triggered the kv rules (i.e. the widget instantiated from Python, e.g. *MyWidget()*).

Changed in version 1.11.0.

Warning: Adding a *__del__* method to a class derived from *Widget* with Python prior to 3.4 will disable automatic garbage collection for instances of that class. This is because the *Widget* class creates reference cycles, thereby *preventing garbage collection*.

Changed in version 1.0.9: Everything related to event properties has been moved to the *EventDispatcher*. Event properties can now be used when constructing a simple class without subclassing *Widget*.

Changed in version 1.5.0: The constructor now accepts *on_** arguments to automatically bind callbacks to properties or events, as in the Kv language.

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)
```

apply_class_lang_rules (*root=None, ignored_consts=set(), rule_children=None*)

Method that is called by kivy to apply the kv rules of this widget's class.

Parameters

root: *Widget* The root widget that instantiated this widget in kv, if the widget was instantiated in kv, otherwise None.

ignored_consts: set (internal) See [apply\(\)](#).

rule_children: list (internal) See [apply\(\)](#).

This is useful to be able to execute code before/after the class kv rules are applied to the widget. E.g. if the kv code requires some properties to be initialized before it is used in a binding rule. If overwriting remember to call `super`, otherwise the kv rules will not be applied.

In the following example,

```
class MyWidget(Widget):
    pass

class OtherWidget(MyWidget):
    pass
```

<MyWidget>: my_prop: some_value

<OtherWidget>: other_prop: some_value

When `OtherWidget` is instantiated with `OtherWidget()`, the widget's [apply_class_lang_rules\(\)](#) is called and it applies the kv rules of this class - <MyWidget> and <OtherWidget>.

Similarly, when the widget is instantiated from kv, e.g.

```
<MyBox@BoxLayout>:
    height: 55
    OtherWidget:
        width: 124
```

`OtherWidget`'s [apply_class_lang_rules\(\)](#) is called and it applies the kv rules of this class - <MyWidget> and <OtherWidget>.

Note: It applies only the class rules not the instance rules. I.e. in the above kv example in the `MyBox` rule when `OtherWidget` is instantiated, its [apply_class_lang_rules\(\)](#) applies the <MyWidget> and <OtherWidget> rules to it - it does not apply the `width: 124` rule.

The width: 124 rule is part of the MyBox rule and is applied by the MyBox's instance's `apply_class_lang_rules()`.

Changed in version 1.11.0.

canvas = None

Canvas of the widget.

The canvas is a graphics object that contains all the drawing instructions for the graphical representation of the widget.

There are no general properties for the Widget class, such as background color, to keep the design simple and lean. Some derived classes, such as Button, do add such convenience properties but generally the developer is responsible for implementing the graphics representation for a custom widget from the ground up. See the derived widget classes for patterns to follow and extend.

See *Canvas* for more information about the usage.

center

Center position of the widget.

center is a *ReferenceListProperty* of (*center_x*, *center_y*) properties.

center_x

X center position of the widget.

center_x is an *AliasProperty* of (*x* + *width* / 2.).

center_y

Y center position of the widget.

center_y is an *AliasProperty* of (*y* + *height* / 2.).

children

List of children of this widget.

children is a *ListProperty* and defaults to an empty list.

Use *add_widget()* and *remove_widget()* for manipulating the children list. Don't manipulate the children list directly unless you know what you are doing.

clear_widgets (children=None)

Remove all (or the specified) *children* of this widget. If the 'children' argument is specified, it should be a list (or filtered list) of children of the current widget.

Changed in version 1.8.0: The *children* argument can be used to specify the children you want to remove.

cls

Class of the widget, used for styling.

collide_point(x, y)

Check if a point (x, y) is inside the widget's axis aligned bounding box.

Parameters

x: numeric x position of the point (in parent coordinates)

y: numeric y position of the point (in parent coordinates)

Returns A bool. True if the point is inside the bounding box, False otherwise.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

collide_widget(wid)

Check if another widget collides with this widget. This function performs an axis-aligned bounding box intersection test by default.

Parameters

wid: **Widget** class Widget to test collision with.

Returns bool. True if the other widget collides with this widget, False otherwise.

```
>>> wid = Widget(size=(50, 50))
>>> wid2 = Widget(size=(50, 50), pos=(25, 25))
>>> wid.collide_widget(wid2)
True
>>> wid2.pos = (55, 55)
>>> wid.collide_widget(wid2)
False
```

disabled

Indicates whether this widget can interact with input or not.

disabled is an *AliasProperty* and defaults to False.

Note:

1. Child Widgets, when added to a disabled widget, will be disabled automatically.
 2. Disabling/enabling a parent disables/enables all of its children.
-

New in version 1.8.0.

Changed in version 1.10.1: *disabled* was changed from a *BooleanProperty* to an *AliasProperty* to allow access to its previous state when a parent's disabled state is changed.

export_as_image(*args, **kwargs)

Return an core *Image* of the actual widget.

New in version 1.11.0.

export_to_png(filename, *args, **kwargs)

Saves an image of the widget and its children in png format at the specified filename. Works by removing the widget canvas from its parent, rendering to an *Fbo*, and calling *save()*.

Note: The image includes only this widget and its children. If you want to include widgets elsewhere in the tree, you must call *export_to_png()* from their common parent, or use *screenshot()* to capture the whole window.

Note: The image will be saved in png format, you should include the extension in your filename.

New in version 1.9.0.

Parameters

filename: **str** The filename with which to save the png.

scale: **float** The amount by which to scale the saved image, defaults to 1.

New in version 1.11.0.

get_parent_window()

Return the parent window.

Returns Instance of the parent window. Can be a *WindowBase* or *Widget*.

get_root_window()

Return the root window.

Returns Instance of the root window. Can be a *WindowBase* or *Widget*.

get_window_matrix(x=0, y=0)

Calculate the transformation matrix to convert between window and widget coordinates.

Parameters

x: float, defaults to 0 Translates the matrix on the x axis.

y: float, defaults to 0 Translates the matrix on the y axis.

height

Height of the widget.

height is a *NumericProperty* and defaults to 100.

Warning: Keep in mind that the *height* property is subject to layout logic and that this has not yet happened at the time of the widget's `__init__` method.

id

Identifier of the widget in the tree.

id is a *StringProperty* and defaults to None.

Note: The *id* is not the same as `id` in the kv language. For the latter, see *ids* and Kivy Language: *ids*.

Warning: The *id* property has been deprecated and will be removed completely in future versions.

ids

This is a dictionary of *ids* defined in your kv language. This will only be populated if you use *ids* in your kv language code.

New in version 1.7.0.

ids is a *DictProperty* and defaults to an empty dict {}.

The *ids* are populated for each root level widget definition. For example:

```
# in kv
<MyWidget@Widget>:
    id: my_widget
    Label:
        id: label_widget
    Widget:
        id: inner_widget
        Label:
            id: inner_label
    TextInput:
```

(continues on next page)

(continued from previous page)

```
        id: text_input
    OtherWidget:
        id: other_widget

<OtherWidget@Widget>
    id: other_widget
    Label:
        id: other_label
    TextInput:
        id: other_textinput
```

Then, in python:

```
>>> widget = MyWidget()
>>> print(widget.ids)
{'other_widget': <weakproxy at 041CFED0 to OtherWidget at 041BEC38>,
 'inner_widget': <weakproxy at 04137EA0 to Widget at 04138228>,
 'inner_label': <weakproxy at 04143540 to Label at 04138260>,
 'label_widget': <weakproxy at 04137B70 to Label at 040F97A0>,
 'text_input': <weakproxy at 041BB5D0 to TextInput at 041BEC00>}
>>> print(widget.ids['other_widget'].ids)
{'other_textinput': <weakproxy at 041DBB40 to TextInput at 041BEF48>,
 'other_label': <weakproxy at 041DB570 to Label at 041BEEA0>}
>>> print(widget.ids['label_widget'].ids)
{}
```

on_touch_down(*touch*)

Receive a touch down event.

Parameters

touch: **MotionEvent** class Touch received. The touch is in parent coordinates.
See [relative layout](#) 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.

opacity

Opacity of the widget and all its children.

New in version 1.4.1.

The opacity attribute controls the opacity of the widget and its children. Be careful, it's a cumulative attribute: the value is multiplied by the current global opacity and the result is applied to the current context color.

For example, if the parent has an opacity of 0.5 and a child has an opacity of 0.2, the real opacity of the child will be $0.5 * 0.2 = 0.1$.

Then, the opacity is applied by the shader as:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

opacity is a *NumericProperty* and defaults to 1.0.

parent

Parent of this widget. The parent of a widget is set when the widget is added to another widget and unset when the widget is removed from its parent.

parent is an *ObjectProperty* and defaults to None.

pos

Position of the widget.

pos is a *ReferenceListProperty* of (*x*, *y*) properties.

pos_hint

Position hint. This property allows you to set the position of the widget inside its parent layout (similar to *size_hint*).

For example, if you want to set the top of the widget to be at 90% height of its parent layout, you can write:

```
widget = Widget(pos_hint={'top': 0.9})
```

The keys 'x', 'right' and 'center_x' will use the parent width. The keys 'y', 'top' and 'center_y' will use the parent height.

See *Float Layout* for further reference.

Note: *pos_hint* is not used by all layouts. Check the documentation of the layout in question to see if it supports *pos_hint*.

pos_hint is an *ObjectProperty* containing a dict.

proxy_ref

Return a proxy reference to the widget, i.e. without creating a reference to the widget. See *weakref.proxy* for more information.

New in version 1.7.2.

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)
```

right

Right position of the widget.

right is an *AliasProperty* of (*x* + *width*).

size

Size of the widget.

size is a *ReferenceListProperty* of (*width*, *height*) properties.

size_hint

Size hint.

size_hint is a *ReferenceListProperty* of (*size_hint_x*, *size_hint_y*) properties.

See *size_hint_x* for more information.

size_hint_max

Maximum size when using *size_hint*.

size_hint_max is a *ReferenceListProperty* of (*size_hint_max_x*, *size_hint_max_y*) properties.

New in version 1.10.0.

size_hint_max_x

When not None, the x-direction maximum size (in pixels, like *width*) when *size_hint_x* is also not None.

Similar to *size_hint_min_x*, except that it sets the maximum width.

size_hint_max_x is a *NumericProperty* and defaults to None.

New in version 1.10.0.

size_hint_max_y

When not None, the y-direction maximum size (in pixels, like *height*) when *size_hint_y* is also not None.

Similar to *size_hint_min_y*, except that it sets the maximum height.

size_hint_max_y is a *NumericProperty* and defaults to None.

New in version 1.10.0.

size_hint_min

Minimum size when using *size_hint*.

size_hint_min is a *ReferenceListProperty* of (*size_hint_min_x*, *size_hint_min_y*) properties.

New in version 1.10.0.

size_hint_min_x

When not None, the x-direction minimum size (in pixels, like *width*) when *size_hint_x* is also not None.

When *size_hint_x* is not None, it is the minimum width that the widget will be set due to the *size_hint_x*. I.e. when a smaller size would be set, *size_hint_min_x* is the value used instead for the widget width. When None, or when *size_hint_x* is None, *size_hint_min_x* doesn't do anything.

Only the *Layout* and *Window* classes make use of the hint.

size_hint_min_x is a *NumericProperty* and defaults to None.

New in version 1.10.0.

size_hint_min_y

When not None, the y-direction minimum size (in pixels, like *height*) when *size_hint_y* is also not None.

When *size_hint_y* is not None, it is the minimum height that the widget will be set due to the *size_hint_y*. I.e. when a smaller size would be set, *size_hint_min_y* is the value used instead for the widget height. When None, or when *size_hint_y* is None, *size_hint_min_y* doesn't do anything.

Only the *Layout* and *Window* classes make use of the hint.

`size_hint_min_y` is a *NumericProperty* and defaults to None.

New in version 1.10.0.

size_hint_x

x size hint. Represents how much space the widget should use in the direction of the x axis relative to its parent's width. Only the *Layout* and *Window* classes make use of the hint.

The `size_hint` is used by layouts for two purposes:

- When the layout considers widgets on their own rather than in relation to its other children, the `size_hint_x` is a direct proportion of the parent width, normally between 0.0 and 1.0. For instance, a widget with `size_hint_x=0.5` in a vertical *BoxLayout* will take up half the *BoxLayout*'s width, or a widget in a *FloatLayout* with `size_hint_x=0.2` will take up 20% of the *FloatLayout* width. If the `size_hint` is greater than 1, the widget will be wider than the parent.
- When multiple widgets can share a row of a layout, such as in a horizontal *BoxLayout*, their widths will be their `size_hint_x` as a fraction of the sum of widget `size_hints`. For instance, if the `size_hint_xs` are (0.5, 1.0, 0.5), the first widget will have a width of 25% of the parent width.

`size_hint_x` is a *NumericProperty* and defaults to 1.

size_hint_y

y size hint.

`size_hint_y` is a *NumericProperty* and defaults to 1.

See `size_hint_x` for more information, but with widths and heights swapped.

to_local(*x, y, relative=False*)

Transform parent coordinates to local (current widget) coordinates.

See *relativelayout* for details on the coordinate systems.

Parameters

relative: bool, defaults to False Change to True if you want to translate coordinates to relative widget coordinates.

to_parent(*x, y, relative=False*)

Transform local (current widget) coordinates to parent coordinates.

See *relativelayout* for details on the coordinate systems.

Parameters

relative: bool, defaults to False Change to True if you want to translate relative positions from a widget to its parent coordinates.

to_widget(*x, y, relative=False*)

Convert the coordinate from window to local (current widget) coordinates.

See *relativelayout* for details on the coordinate systems.

to_window(*x, y, initial=True, relative=False*)

If `initial` is True, the default, it transforms **parent** coordinates to window coordinates. Otherwise, it transforms **local** (current widget) coordinates to window coordinates.

See *relativelayout* for details on the coordinate systems.

top

Top position of the widget.

`top` is an *AliasProperty* of (`y + height`).

walk(*restrict=False, loopback=False*)

Iterator that walks the widget tree starting with this widget and goes forward returning widgets in the order in which layouts display them.

Parameters

restrict: bool, defaults to False If True, it will only iterate through the widget and its children (or children of its children etc.). Defaults to False.

loopback: bool, defaults to False If True, when the last widget in the tree is reached, it'll loop back to the uppermost root and start walking until we hit this widget again. Naturally, it can only loop back when *restrict* is False. Defaults to False.

Returns A generator that walks the tree, returning widgets in the forward layout order.

For example, given a tree with the following structure:

```
GridLayout:
  Button
  BoxLayout:
    id: box
    Widget
    Button
  Widget
```

walking this tree:

```
>>> # Call walk on box with loopback True, and restrict False
>>> [type(widget) for widget in box.walk(loopback=True)]
[<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>,
 <class 'Widget'>, <class 'GridLayout'>, <class 'Button'>]
>>> # Now with loopback False, and restrict False
>>> [type(widget) for widget in box.walk()]
[<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>,
 <class 'Widget'>]
>>> # Now with restrict True
>>> [type(widget) for widget in box.walk(restrict=True)]
[<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>]
```

New in version 1.9.0.

walk_reverse(*loopback=False*)

Iterator that walks the widget tree backwards starting with the widget before this, and going backwards returning widgets in the reverse order in which layouts display them.

This walks in the opposite direction of *walk()*, so a list of the tree generated with *walk()* will be in reverse order compared to the list generated with this, provided *loopback* is True.

Parameters

loopback: bool, defaults to False If True, when the uppermost root in the tree is reached, it'll loop back to the last widget and start walking back until after we hit widget again. Defaults to False.

Returns A generator that walks the tree, returning widgets in the reverse layout order.

For example, given a tree with the following structure:

```
GridLayout:
  Button
```

(continues on next page)


```
BoxLayout:
    id: box
    Widget
    Button
    Widget
```

walking this tree:

```
>>> # Call walk on box with loopback True
>>> [type(widget) for widget in box.walk_reverse(loopback=True)]
[<class 'Button'>, <class 'GridLayout'>, <class 'Widget'>,
 <class 'Button'>, <class 'Widget'>, <class 'BoxLayout'>]
>>> # Now with loopback False
>>> [type(widget) for widget in box.walk_reverse()]
[<class 'Button'>, <class 'GridLayout'>]
>>> forward = [w for w in box.walk(loopback=True)]
>>> backward = [w for w in box.walk_reverse(loopback=True)]
>>> forward == backward[::-1]
True
```

New in version 1.9.0.

width

Width of the widget.

width is a *NumericProperty* and defaults to 100.

Warning: Keep in mind that the *width* property is subject to layout logic and that this has not yet happened at the time of the widget's `__init__` method.

x

X position of the widget.

x is a *NumericProperty* and defaults to 0.

y

Y position of the widget.

y is a *NumericProperty* and defaults to 0.

exception kivy.uix.widget.WidgetException

Bases: Exception

Fired when the widget gets an exception.

Part V

APPENDIX

The appendix contains licensing information and an enumeration of all the different modules, classes, functions and variables available in Kivy.

License

Kivy is released and distributed under the terms of the MIT license starting version 1.7.2. Older versions are still under the LGPLv3.

You should have received a copy of the MIT license alongside your Kivy distribution. See the LICENSE file in the Kivy root folder. An online version of the license can be found at:

<https://github.com/kivy/kivy/blob/master/LICENSE>

In a nutshell, the license allows you to use Kivy in your own projects regardless of whether they are open source, closed source, commercial or free. Even if the license doesn't require it, we would really appreciate when you make changes to the Kivy sourcecode **itself**, share those changes with us!

For a list of authors, please see the file AUTHORS that accompanies the Kivy source code distribution (next to LICENSE).

Kivy – Copyright 2010-2018, The Kivy Authors.

Python Module Index

k

kivy, 159
kivy.animation, 160
kivy.app, 171
kivy.atlas, 186
kivy.base, 222
kivy.cache, 189
kivy.clock, 191
kivy.compat, 201
kivy.config, 202
kivy.context, 208
kivy.core, 259
kivy.core.audio, 259
kivy.core.camera, 261
kivy.core.clipboard, 262
kivy.core.gl, 262
kivy.core.image, 262
kivy.core.spelling, 267
kivy.core.text, 268
kivy.core.text.markup, 275
kivy.core.text.text_layout, 272
kivy.core.video, 276
kivy.core.window, 278
kivy.deps, 293
kivy.effects, 295
kivy.effects.dampedscroll, 295
kivy.effects.kinetic, 296
kivy.effects.opacityscroll, 297
kivy.effects.scroll, 298
kivy.event, 208
kivy.factory, 216
kivy.garden, 301
kivy.geometry, 217
kivy.gesture, 218
kivy.graphics, 305
kivy.graphics.cgl, 331
kivy.graphics.compiler, 340
kivy.graphics.context, 336
kivy.graphics.context_instructions, 332
kivy.graphics.fbo, 336
kivy.graphics.gl_instructions, 339
kivy.graphics.instructions, 327
kivy.graphics.opengl, 341
kivy.graphics.opengl_utils, 350
kivy.graphics.scissor_instructions, 351
kivy.graphics.shader, 352
kivy.graphics.stencil_instructions, 354
kivy.graphics.svg, 356
kivy.graphics.tesselator, 357
kivy.graphics.texture, 360
kivy.graphics.transformation, 366
kivy.graphics.vertex_instructions, 370
kivy.input, 381
kivy.input.factory, 401
kivy.input.motionEvent, 397
kivy.input.postproc, 385
kivy.input.postproc.calibration, 385
kivy.input.postproc.dejitter, 386
kivy.input.postproc.doubletap, 387
kivy.input.postproc.ignorelist, 387
kivy.input.postproc.retaintouch, 387
kivy.input.postproc.tripletap, 388
kivy.input.provider, 402
kivy.input.providers, 388
kivy.input.providers.androidjoystick, 388
kivy.input.providers.hidinput, 390
kivy.input.providers.leapfinger, 389
kivy.input.providers.linuxwacom, 392
kivy.input.providers.mactouch, 392
kivy.input.providers.mouse, 389
kivy.input.providers.mtdev, 391
kivy.input.providers.probesysfs, 388
kivy.input.providers.tuio, 393
kivy.input.providers.wm_common, 389
kivy.input.providers.wm_pen, 393

- kivy.input.providers.wm_touch, 393
- kivy.input.recorder, 395
- kivy.input.shape, 402
- kivy.interactive, 219
- kivy.lang, 403
- kivy.lang.builder, 419
- kivy.lang.parser, 423
- kivy.lib, 425
- kivy.lib.ddsfile, 425
- kivy.lib.gstplayer, 425
- kivy.lib.mtdev, 426
- kivy.loader, 184
- kivy.logger, 225
- kivy.metrics, 226
- kivy.modules, 429
- kivy.modules.console, 430
- kivy.modules.inspector, 434
- kivy.modules.joycursor, 435
- kivy.modules.keybinding, 436
- kivy.modules.monitor, 437
- kivy.modules.recorder, 437
- kivy.modules.screen, 438
- kivy.modules.touchring, 438
- kivy.modules.webdebugger, 439
- kivy.multistroke, 228
- kivy.network, 441
- kivy.network.urlrequest, 441
- kivy.parser, 236
- kivy.properties, 237
- kivy.resources, 251
- kivy.storage, 445
- kivy.storage.dictstore, 449
- kivy.storage.jsonstore, 449
- kivy.storage.redisstore, 449
- kivy.support, 251
- kivy.tools, 451
- kivy.tools.packaging, 452
- kivy.uix, 453
- kivy.uix.accordion, 499
- kivy.uix.actionbar, 504
- kivy.uix.anchorlayout, 511
- kivy.uix.behaviors, 453
- kivy.uix.behaviors.button, 465
- kivy.uix.behaviors.codenavigation, 467
- kivy.uix.behaviors.compoundselection, 467
- kivy.uix.behaviors.cover, 473
- kivy.uix.behaviors.drag, 474
- kivy.uix.behaviors.emacs, 475
- kivy.uix.behaviors.focus, 476
- kivy.uix.behaviors.knspac, 480
- kivy.uix.behaviors.togglebutton, 485
- kivy.uix.behaviors.touchripple, 487
- kivy.uix.boxlayout, 513
- kivy.uix.bubble, 515
- kivy.uix.button, 520
- kivy.uix.camera, 521
- kivy.uix.carousel, 522
- kivy.uix.checkbox, 526
- kivy.uix.codeinput, 528
- kivy.uix.colorpicker, 530
- kivy.uix.dropdown, 532
- kivy.uix.effectwidget, 535
- kivy.uix.filechooser, 540
- kivy.uix.floatlayout, 550
- kivy.uix.gesturesurface, 552
- kivy.uix.gridlayout, 555
- kivy.uix.image, 559
- kivy.uix.label, 562
- kivy.uix.layout, 575
- kivy.uix.modalview, 577
- kivy.uix.pagelayout, 580
- kivy.uix.popup, 581
- kivy.uix.progressbar, 584
- kivy.uix.recycleboxlayout, 585
- kivy.uix.recyclegridlayout, 586
- kivy.uix.recyclelayout, 586
- kivy.uix.recycleview, 489
- kivy.uix.recycleview.datamodel, 493
- kivy.uix.recycleview.layout, 494
- kivy.uix.recycleview.views, 497
- kivy.uix.relativelayout, 587
- kivy.uix.rst, 592
- kivy.uix.sandbox, 595
- kivy.uix.scatter, 596
- kivy.uix.scatterlayout, 601
- kivy.uix.screenmanager, 603
- kivy.uix.scrollview, 612
- kivy.uix.settings, 618
- kivy.uix.slider, 628
- kivy.uix.spinner, 632
- kivy.uix.splitter, 634
- kivy.uix.stacklayout, 637
- kivy.uix.stencilview, 639
- kivy.uix.switch, 640
- kivy.uix.tabbedpanel, 641
- kivy.uix.textinput, 648
- kivy.uix.togglebutton, 662
- kivy.uix.treeview, 662
- kivy.uix.video, 668
- kivy.uix.videoplayer, 670
- kivy.uix.vkeyboard, 675
- kivy.uix.widget, 680
- kivy.utils, 252
- kivy.vector, 254
- kivy.weakmethod, 258
- kivy.weakproxy, 258

Index

A

- a (kivy.graphics.ClearColor attribute), 325
- a (kivy.graphics.Color attribute), 311
- a (kivy.graphics.context_instructions.Color attribute), 333
- a (kivy.graphics.gl_instructions.ClearColor attribute), 339
- a (kivy.uix.colorpicker.ColorWheel attribute), 531
- AbstractStore (class in kivy.storage), 447
- accept_stroke() (kivy.uix.gesturesurface.GestureContainer method), 554
- Accordion (class in kivy.uix.accordion), 501
- accordion (kivy.uix.accordion.AccordionItem attribute), 502
- AccordionException, 504
- AccordionItem (class in kivy.uix.accordion), 501
- action_previous (kivy.uix.actionbar.ActionView attribute), 508
- action_view (kivy.uix.actionbar.ActionBar attribute), 510
- ActionBar (class in kivy.uix.actionbar), 510
- ActionBarException, 505
- ActionButton (class in kivy.uix.actionbar), 505
- ActionCheck (class in kivy.uix.actionbar), 506
- ActionDropDown (class in kivy.uix.actionbar), 506
- ActionGroup (class in kivy.uix.actionbar), 506
- ActionItem (class in kivy.uix.actionbar), 505
- ActionOverflow (class in kivy.uix.actionbar), 507
- ActionPrevious (class in kivy.uix.actionbar), 509
- ActionSeparator (class in kivy.uix.actionbar), 506
- ActionToggleButton (class in kivy.uix.actionbar), 506
- ActionView (class in kivy.uix.actionbar), 508
- activate() (kivy.modules.console.ConsoleAddon method), 434
- activated (kivy.modules.console.Console attribute), 433
- active (kivy.uix.checkbox.CheckBox attribute), 527
- active (kivy.uix.switch.Switch attribute), 640
- active_norm_pos (kivy.uix.switch.Switch attribute), 640
- add() (kivy.graphics.Canvas method), 309
- add() (kivy.graphics.InstructionGroup method), 314
- add() (kivy.graphics.instructions.Canvas method), 329
- add() (kivy.graphics.instructions.InstructionGroup method), 327
- add_callback() (kivy.config.ConfigParser method), 206
- add_contour() (kivy.graphics.tesselator.Tesselator method), 359
- add_event_listener() (kivy.base.EventLoopBase method), 222
- add_gesture() (kivy.gesture.GestureDatabase method), 219
- add_gesture() (kivy.multistroke.Recognizer method), 230
- add_handler() (kivy.base.ExceptionManagerBase method), 224
- add_input_provider() (kivy.base.EventLoopBase method), 222
- add_interface() (kivy.uix.settings.Settings method), 621
- add_item() (kivy.uix.settings.MenuSidebar method), 628
- add_json_panel() (kivy.uix.settings.Settings method), 621
- add_kivy_panel() (kivy.uix.settings.Settings method), 621
- add_mipmap() (kivy.core.image.ImageData method), 266
- add_node() (kivy.uix.treeview.TreeView method), 664
- add_panel() (kivy.modules.console.Console method), 433
- add_panel() (kivy.uix.settings.ContentPanel method), 626
- add_panel() (kivy.uix.settings.InterfaceWithSidebar

method), 626

add_point() (kivy.gesture.GestureStroke method), 219

add_point() (kivy.graphics.Point method), 321

add_point() (kivy.graphics.vertex_instructions.Point method), 377

add_point() (kivy.multistroke.UnistrokeTemplate method), 235

add_postproc_module() (kivy.base.EventLoopBase method), 222

add_reload_observer() (kivy.graphics.context.Context method), 336

add_reload_observer() (kivy.graphics.Fbo method), 312

add_reload_observer() (kivy.graphics.fbo.Fbo method), 338

add_reload_observer() (kivy.graphics.texture.Texture method), 363

add_screen() (kivy.uix.screenmanager.ShaderTransition method), 610

add_screen() (kivy.uix.screenmanager.SwapTransition method), 611

add_screen() (kivy.uix.screenmanager.TransitionBase method), 609

add_stroke() (kivy.gesture.Gesture method), 218

add_stroke() (kivy.multistroke.Candidate method), 236

add_stroke() (kivy.multistroke.MultistrokeGesture method), 234

add_stroke() (kivy.uix.gesturesurface.GestureContainer method), 554

add_toolbar_widget() (kivy.modules.console.Console method), 433

add_widget() (kivy.core.window.WindowBase method), 281

add_widget() (kivy.uix.accordion.Accordion method), 501

add_widget() (kivy.uix.accordion.AccordionItem method), 502

add_widget() (kivy.uix.actionbar.ActionBar method), 510

add_widget() (kivy.uix.actionbar.ActionGroup method), 506

add_widget() (kivy.uix.actionbar.ActionOverflow method), 507

add_widget() (kivy.uix.actionbar.ActionView method), 508

add_widget() (kivy.uix.boxlayout.BoxLayout method), 514

add_widget() (kivy.uix.bubble.Bubble method), 517

add_widget() (kivy.uix.carousel.Carousel method), 523

add_widget() (kivy.uix.dropdown.DropDown method), 533

add_widget() (kivy.uix.effectwidget.EffectWidget method), 537

add_widget() (kivy.uix.filechooser.FileChooser method), 545

add_widget() (kivy.uix.floatlayout.FloatLayout method), 551

add_widget() (kivy.uix.layout.Layout method), 575

add_widget() (kivy.uix.popup.Popup method), 583

add_widget() (kivy.uix.recycleview.RecycleView method), 492

add_widget() (kivy.uix.sandbox.Sandbox method), 595

add_widget() (kivy.uix.scatterlayout.ScatterLayout method), 601

add_widget() (kivy.uix.screenmanager.ScreenManager method), 606

add_widget() (kivy.uix.scrollview.ScrollView method), 614

add_widget() (kivy.uix.settings.ContentPanel method), 627

add_widget() (kivy.uix.settings.SettingItem method), 622

add_widget() (kivy.uix.splitter.Splitter method), 635

add_widget() (kivy.uix.tabbedpanel.TabbedPanel method), 644

add_widget() (kivy.uix.tabbedpanel.TabbedPanelItem method), 647

add_widget() (kivy.uix.widget.Widget method), 683

adddefaultsection() (kivy.config.ConfigParser method), 207

addons (kivy.modules.console.Console attribute), 433

AdvancedEffectBase (class in kivy.uix.effectwidget), 539

after (kivy.graphics.Canvas attribute), 309

after (kivy.graphics.instructions.Canvas attribute), 329

AliasProperty (class in kivy.properties), 246

allow_copy (kivy.uix.textinput.TextInput attribute), 652

allow_fullscreen (kivy.uix.videoplayer.VideoPlayer attribute), 672

allow_no_selection (kivy.uix.behaviors.togglebutton.ToggleButton attribute), 486

allow_no_selection (kivy.uix.behaviors.ToggleButtonBehavior attribute), 455

allow_screensaver (kivy.core.window.WindowBase

attribute), 281
 allow_stretch (kivy.uix.image.Image attribute), 560
 always_release (kivy.uix.behaviors.button.ButtonBehavior attribute), 466
 always_release (kivy.uix.behaviors.ButtonBehavior attribute), 455
 always_release (kivy.uix.behaviors.touchripple.TouchRippleButtonBehavior attribute), 488
 always_release (kivy.uix.behaviors.TouchRippleButtonBehavior attribute), 465
 anchor_x (kivy.graphics.svg.Svg attribute), 356
 anchor_x (kivy.uix.anchorlayout.AnchorLayout attribute), 512
 anchor_y (kivy.graphics.svg.Svg attribute), 356
 anchor_y (kivy.uix.anchorlayout.AnchorLayout attribute), 512
 AnchorLayout (class in kivy.uix.anchorlayout), 512
 anchors (kivy.core.text.markup.MarkupLabel attribute), 276
 anchors (kivy.uix.label.Label attribute), 566
 AndroidMotionEventProvider (class in kivy.input.providers.androidjoystick), 388
 angle (kivy.graphics.context_instructions.Rotate attribute), 334
 angle (kivy.graphics.Rotate attribute), 322
 angle() (kivy.vector.Vector method), 255
 angle_end (kivy.graphics.Ellipse attribute), 311
 angle_end (kivy.graphics.vertex_instructions.Ellipse attribute), 372
 angle_start (kivy.graphics.Ellipse attribute), 311
 angle_start (kivy.graphics.vertex_instructions.Ellipse attribute), 372
 anim_available (kivy.core.image.Image attribute), 264
 anim_cancel_duration (kivy.uix.carousel.Carousel attribute), 524
 anim_delay (kivy.core.image.Image attribute), 264
 anim_delay (kivy.uix.image.Image attribute), 560
 anim_duration (kivy.uix.accordion.Accordion attribute), 501
 anim_func (kivy.uix.accordion.Accordion attribute), 501
 anim_index (kivy.core.image.Image attribute), 264
 anim_kwargs (kivy.uix.pagelayout.PageLayout attribute), 580
 anim_loop (kivy.uix.image.Image attribute), 560
 anim_move_duration (kivy.uix.carousel.Carousel attribute), 524
 anim_reset() (kivy.core.image.Image method), 264
 anim_type (kivy.uix.carousel.Carousel attribute), 524
 animated_properties (kivy.animation.Animation attribute), 162
 Animation (class in kivy.animation), 161
 AnimationTransition (class in kivy.animation), 163
 animations (kivy.uix.videoplayer.VideoPlayer attribute), 672
 App (class in kivy.app), 177
 app_icon (kivy.uix.actionbar.ActionPrevious attribute), 509
 app_icon_height (kivy.uix.actionbar.ActionPrevious attribute), 509
 app_icon_width (kivy.uix.actionbar.ActionPrevious attribute), 509
 append() (kivy.cache.Cache static method), 190
 apply() (kivy.lang.builder.BuilderBase method), 420
 apply() (kivy.lang.BuilderBase method), 416
 apply_class_lang_rules() (kivy.uix.widget.Widget method), 684
 apply_property() (kivy.event.EventDispatcher method), 209
 apply_rules() (kivy.lang.builder.BuilderBase method), 420
 apply_rules() (kivy.lang.BuilderBase method), 416
 apply_selection() (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 494
 apply_transform() (kivy.uix.scatter.Scatter method), 598
 apply_transform_2d() (kivy.input.MotionEvent method), 381
 apply_transform_2d() (kivy.input.motionevent.MotionEvent method), 398
 ApplyContextMatrix (class in kivy.graphics), 326
 arrow_image (kivy.uix.bubble.Bubble attribute), 518
 arrow_pos (kivy.uix.bubble.Bubble attribute), 518
 ask_update() (kivy.graphics.Callback method), 308
 ask_update() (kivy.graphics.Canvas method), 309
 ask_update() (kivy.graphics.instructions.Callback method), 331
 ask_update() (kivy.graphics.instructions.Canvas method), 329
 ask_update() (kivy.graphics.texture.Texture method), 363
 ask_update() (kivy.graphics.texture.TextureRegion method), 366
 async_clear() (kivy.storage.AbstractStore method), 447
 async_count() (kivy.storage.AbstractStore method), 447
 async_delete() (kivy.storage.AbstractStore method), 447
 async_exists() (kivy.storage.AbstractStore method), 447

async_find() (kivy.storage.AbstractStore method), 447
 async_get() (kivy.storage.AbstractStore method), 448
 async_idle() (kivy.base.EventLoopBase method), 222
 async_idle() (kivy.clock.ClockBaseBehavior method), 199
 async_idle() (kivy.clock.ClockBaseFreeInterruptOnly method), 201
 async_idle() (kivy.clock.ClockBaseInterruptBehavior method), 200
 async_keys() (kivy.storage.AbstractStore method), 448
 async_put() (kivy.storage.AbstractStore method), 448
 async_run() (kivy.app.App method), 178
 async_runTouchApp() (in module kivy.app), 184
 async_runTouchApp() (in module kivy.base), 224
 async_tick() (kivy.clock.ClockBaseBehavior method), 199
 AsyncImage (class in kivy.uix.image), 561
 Atlas (class in kivy.atlas), 188
 attach_recycleview() (kivy.uix.recycleview.datamodel.RecycleDataModel method), 494
 attach_recycleview() (kivy.uix.recycleview.datamodel.RecycleDataModelBehavior method), 493
 attach_recycleview() (kivy.uix.recycleview.views.RecycleDataAdapter method), 497
 attach_to (kivy.uix.dropdown.DropDown attribute), 534
 attach_to (kivy.uix.modalview.ModalView attribute), 578
 auto_bring_to_front (kivy.uix.scatter.Scatter attribute), 598
 auto_dismiss (kivy.uix.dropdown.DropDown attribute), 534
 auto_dismiss (kivy.uix.modalview.ModalView attribute), 578
 auto_indent (kivy.uix.textinput.TextInput attribute), 652
 auto_scale (kivy.graphics.BorderImage attribute), 308
 auto_scale (kivy.graphics.vertex_instructions.BorderImage attribute), 372
 auto_width (kivy.uix.dropdown.DropDown attribute), 534
 available_layouts (kivy.uix.vkeyboard.VKeyboard attribute), 677
 axis (kivy.graphics.context_instructions.Rotate attribute), 334
 axis (kivy.graphics.Rotate attribute), 322
B
 b (kivy.graphics.ClearColor attribute), 325
 b (kivy.graphics.Color attribute), 311
 b (kivy.graphics.context_instructions.Color attribute), 333
 b (kivy.graphics.gl_instructions.ClearColor attribute), 339
 b (kivy.uix.colorpicker.ColorWheel attribute), 531
 background (kivy.uix.modalview.ModalView attribute), 578
 background (kivy.uix.vkeyboard.VKeyboard attribute), 677
 background_active (kivy.uix.textinput.TextInput attribute), 652
 background_border (kivy.uix.vkeyboard.VKeyboard attribute), 677
 background_checkbox_disabled_down (kivy.uix.checkbox.CheckBox attribute), 527
 background_checkbox_disabled_normal (kivy.uix.checkbox.CheckBox attribute), 527
 background_checkbox_down (kivy.uix.checkbox.CheckBox attribute), 527
 background_checkbox_down (kivy.uix.checkbox.CheckBox attribute), 527
 background_checkbox_down (kivy.uix.checkbox.CheckBox attribute), 527
 background_color (kivy.uix.actionbar.ActionBar attribute), 511
 background_color (kivy.uix.actionbar.ActionView attribute), 508
 background_color (kivy.uix.bubble.Bubble attribute), 518
 background_color (kivy.uix.button.Button attribute), 520
 background_color (kivy.uix.effectwidget.EffectWidget attribute), 538
 background_color (kivy.uix.modalview.ModalView attribute), 578
 background_color (kivy.uix.rst.RstDocument attribute), 593
 background_color (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 background_color (kivy.uix.textinput.TextInput attribute), 652
 background_color (kivy.uix.vkeyboard.VKeyboard attribute), 677
 background_disabled (kivy.uix.vkeyboard.VKeyboard attribute), 677
 background_disabled_down (kivy.uix.button.Button attribute), 521
 background_disabled_horizontal (kivy.uix.slider.Slider attribute), 629
 background_disabled_image (kivy.uix.tabbedpanel.TabbedPanel attribute), 645

attribute), 645
 background_disabled_normal (kivy.uix.accordion.AccordionItem attribute), 502
 background_disabled_normal (kivy.uix.button.Button attribute), 521
 background_disabled_normal (kivy.uix.textinput.TextInput attribute), 652
 background_disabled_selected (kivy.uix.accordion.AccordionItem attribute), 502
 background_disabled_vertical (kivy.uix.slider.Slider attribute), 629
 background_down (kivy.uix.actionbar.ActionItem attribute), 505
 background_down (kivy.uix.button.Button attribute), 521
 background_horizontal (kivy.uix.slider.Slider attribute), 629
 background_image (kivy.uix.actionbar.ActionBar attribute), 511
 background_image (kivy.uix.actionbar.ActionSeparator attribute), 506
 background_image (kivy.uix.actionbar.ActionView attribute), 509
 background_image (kivy.uix.bubble.Bubble attribute), 518
 background_image (kivy.uix.tabbedpanel.StripLayout attribute), 644
 background_image (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 background_normal (kivy.uix.accordion.AccordionItem attribute), 502
 background_normal (kivy.uix.actionbar.ActionItem attribute), 505
 background_normal (kivy.uix.button.Button attribute), 521
 background_normal (kivy.uix.textinput.TextInput attribute), 652
 background_radio_disabled_down (kivy.uix.checkbox.CheckBox attribute), 527
 background_radio_disabled_normal (kivy.uix.checkbox.CheckBox attribute), 528
 background_radio_down (kivy.uix.checkbox.CheckBox attribute), 528
 background_radio_normal (kivy.uix.checkbox.CheckBox attribute), 528
 background_selected (kivy.uix.accordion.AccordionItem attribute), 502
 background_vertical (kivy.uix.slider.Slider attribute), 629
 background_width (kivy.uix.slider.Slider attribute), 629
 bar_color (kivy.uix.scrollview.ScrollView attribute), 614
 bar_inactive_color (kivy.uix.scrollview.ScrollView attribute), 614
 bar_margin (kivy.uix.scrollview.ScrollView attribute), 614
 bar_pos (kivy.uix.scrollview.ScrollView attribute), 615
 bar_pos_x (kivy.uix.scrollview.ScrollView attribute), 615
 bar_pos_y (kivy.uix.scrollview.ScrollView attribute), 615
 bar_width (kivy.uix.scrollview.ScrollView attribute), 615
 base_direction (kivy.uix.label.Label attribute), 567
 base_direction (kivy.uix.textinput.TextInput attribute), 652
 base_font_size (kivy.uix.rst.RstDocument attribute), 593
 bbox (kivy.uix.scatter.Scatter attribute), 598
 before (kivy.graphics.Canvas attribute), 309
 before (kivy.graphics.instructions.Canvas attribute), 329
 best (kivy.multistroke.ProgressTracker attribute), 233
 Bezier (class in kivy.graphics), 306
 Bezier (class in kivy.graphics.vertex_instructions), 379
 bezier (kivy.graphics.Line attribute), 316
 bezier (kivy.graphics.vertex_instructions.Line attribute), 374
 bezier_precision (kivy.graphics.Line attribute), 316
 bezier_precision (kivy.graphics.vertex_instructions.Line attribute), 374
 bind() (kivy.event.EventDispatcher method), 209
 bind() (kivy.event.Observable method), 216
 bind() (kivy.graphics.Fbo method), 312
 bind() (kivy.graphics.fbo.Fbo method), 338
 bind() (kivy.graphics.texture.Texture method), 363
 bind() (kivy.graphics.texture.TextureRegion method), 366
 bind() (kivy.lang.builder.Observable method), 419
 bind() (kivy.lang.Observable method), 415
 bind() (kivy.properties.Property method), 242
 BindTexture (class in kivy.graphics), 307
 BindTexture (class in kivy.graphics.context_instructions), 333
 blit_buffer() (kivy.graphics.texture.Texture method), 363
 blit_data() (kivy.graphics.texture.Texture method),

364
 bold (kivy.uix.label.Label attribute), 567
 BooleanProperty (class in kivy.properties), 245
 border (kivy.graphics.BorderImage attribute), 308
 border (kivy.graphics.vertex_instructions.BorderImage attribute), 372
 border (kivy.uix.actionbar.ActionBar attribute), 511
 border (kivy.uix.bubble.Bubble attribute), 518
 border (kivy.uix.button.Button attribute), 521
 border (kivy.uix.modalview.ModalView attribute), 579
 border (kivy.uix.pagelayout.PageLayout attribute), 580
 border (kivy.uix.splitter.Splitter attribute), 636
 border (kivy.uix.tabbedpanel.StripLayout attribute), 644
 border (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 border (kivy.uix.textinput.TextInput attribute), 653
 border_auto_scale (kivy.uix.bubble.Bubble attribute), 518
 border_horizontal (kivy.uix.slider.Slider attribute), 630
 border_vertical (kivy.uix.slider.Slider attribute), 630
 BorderImage (class in kivy.graphics), 307
 BorderImage (class in kivy.graphics.vertex_instructions), 371
 borderless (kivy.core.window.WindowBase attribute), 281
 boundary() (in module kivy.utils), 253
 BoundedNumericProperty (class in kivy.properties), 245
 bounds (kivy.properties.BoundedNumericProperty attribute), 245
 BoxLayout (class in kivy.uix.boxlayout), 514
 Bubble (class in kivy.uix.bubble), 517
 BubbleButton (class in kivy.uix.bubble), 519
 BubbleContent (class in kivy.uix.bubble), 519
 bufferfmt (kivy.graphics.texture.Texture attribute), 364
 build() (kivy.app.App method), 178
 build_config() (kivy.app.App method), 178
 build_settings() (kivy.app.App method), 178
 Builder (in module kivy.lang.builder), 420
 BuilderBase (class in kivy.lang), 415
 BuilderBase (class in kivy.lang.builder), 420
 BuilderException, 419, 423
 Button (class in kivy.uix.button), 520
 ButtonBehavior (class in kivy.uix.behaviors), 454
 ButtonBehavior (class in kivy.uix.behaviors.button), 466
 buttons_layout (kivy.uix.settings.MenuSidebar attribute), 628

C
 Cache (class in kivy.cache), 190
 Callback (class in kivy.graphics), 308
 Callback (class in kivy.graphics.instructions), 330
 callback (kivy.clock.ClockEvent attribute), 196
 callback (kivy.core.window.Keyboard attribute), 279
 callback (kivy.graphics.Callback attribute), 309
 callback (kivy.graphics.instructions.Callback attribute), 331
 callback (kivy.uix.vkeyboard.VKeyboard attribute), 677
 Camera (class in kivy.uix.camera), 522
 CameraBase (class in kivy.core.camera), 261
 cancel() (kivy.animation.Animation method), 162
 cancel() (kivy.clock.ClockEvent method), 196
 cancel() (kivy.effects.kinetic.KineticEffect method), 296
 cancel() (kivy.network.urlrequest.UrlRequest method), 443
 cancel() (kivy.uix.filechooser.FileChooserController method), 546
 cancel() (kivy.uix.filechooser.FileChooserProgressBase method), 549
 cancel_all() (kivy.animation.Animation static method), 162
 cancel_property() (kivy.animation.Animation method), 162
 cancel_selection() (kivy.uix.textinput.TextInput method), 653
 Candidate (class in kivy.multistroke), 235
 Canvas (class in kivy.graphics), 309
 Canvas (class in kivy.graphics.instructions), 329
 canvas (kivy.uix.widget.Widget attribute), 685
 CanvasBase (class in kivy.graphics), 310
 CanvasBase (class in kivy.graphics.instructions), 329
 cap (kivy.graphics.Line attribute), 316
 cap (kivy.graphics.vertex_instructions.Line attribute), 374
 cap_precision (kivy.graphics.Line attribute), 316
 cap_precision (kivy.graphics.vertex_instructions.Line attribute), 374
 CardTransition (class in kivy.uix.screenmanager), 611
 Carousel (class in kivy.uix.carousel), 523
 center (kivy.core.window.WindowBase attribute), 281
 center (kivy.uix.widget.Widget attribute), 685
 center_stroke() (kivy.gesture.GestureStroke method), 219
 center_x (kivy.uix.widget.Widget attribute), 685
 center_y (kivy.uix.widget.Widget attribute), 685
 cgl_get_backend_name() (in module kivy.graphics.cgl), 332

cgl_get_initialized_backend_name() (in module kivy.graphics.cgl), 332
 cgl_init() (in module kivy.graphics.cgl), 332
 ChangeState (class in kivy.graphics), 326
 ChannelMixEffect (class in kivy.uix.effectwidget), 539
 check() (kivy.core.spelling.SpellingBase method), 267
 CheckBox (class in kivy.uix.checkbox), 527
 children (kivy.core.window.WindowBase attribute), 281
 children (kivy.graphics.InstructionGroup attribute), 314
 children (kivy.graphics.instructions.InstructionGroup attribute), 327
 children (kivy.uix.widget.Widget attribute), 685
 chunk_size (kivy.network.urlrequest.UrlRequest attribute), 443
 cid (kivy.clock.ClockEvent attribute), 196
 circle (kivy.graphics.Line attribute), 316
 circle (kivy.graphics.vertex_instructions.Line attribute), 375
 circumcircle() (in module kivy.geometry), 217
 clear() (kivy.core.window.WindowBase method), 281
 clear() (kivy.graphics.Canvas method), 309
 clear() (kivy.graphics.InstructionGroup method), 314
 clear() (kivy.graphics.instructions.Canvas method), 329
 clear() (kivy.graphics.instructions.InstructionGroup method), 327
 clear() (kivy.storage.AbstractStore method), 448
 clear() (kivy.utils.SafeList method), 253
 clear_buffer() (kivy.graphics.Fbo method), 312
 clear_buffer() (kivy.graphics.fbo.Fbo method), 338
 clear_color (kivy.graphics.ClearBuffers attribute), 326
 clear_color (kivy.graphics.Fbo attribute), 313
 clear_color (kivy.graphics.fbo.Fbo attribute), 338
 clear_color (kivy.graphics.gl_instructions.ClearBuffers attribute), 339
 clear_depth (kivy.graphics.ClearBuffers attribute), 326
 clear_depth (kivy.graphics.gl_instructions.ClearBuffers attribute), 340
 clear_selection() (kivy.uix.behaviors.compoundselection.CompoundSelection method), 469
 clear_selection() (kivy.uix.behaviors.CompoundSelection method), 460
 clear_stencil (kivy.graphics.ClearBuffers attribute), 326
 clear_stencil (kivy.graphics.gl_instructions.ClearBuffers attribute), 340
 clear_widgets() (kivy.uix.actionbar.ActionGroup method), 507
 clear_widgets() (kivy.uix.bubble.Bubble method), 518
 clear_widgets() (kivy.uix.carousel.Carousel method), 524
 clear_widgets() (kivy.uix.dropdown.DropDown method), 534
 clear_widgets() (kivy.uix.effectwidget.EffectWidget method), 538
 clear_widgets() (kivy.uix.sandbox.Sandbox method), 595
 clear_widgets() (kivy.uix.scatterlayout.ScatterLayout method), 602
 clear_widgets() (kivy.uix.screenmanager.ScreenManager method), 607
 clear_widgets() (kivy.uix.splitter.Splitter method), 636
 clear_widgets() (kivy.uix.tabbedpanel.TabbedPanel method), 645
 clear_widgets() (kivy.uix.widget.Widget method), 685
 ClearBuffers (class in kivy.graphics), 325
 ClearBuffers (class in kivy.graphics.gl_instructions), 339
 ClearColor (class in kivy.graphics), 325
 ClearColor (class in kivy.graphics.gl_instructions), 339
 clearcolor (kivy.core.window.WindowBase attribute), 281
 clearcolor (kivy.uix.screenmanager.ShaderTransition attribute), 610
 Clock (in module kivy.clock), 196
 clock (kivy.clock.ClockEvent attribute), 196
 clock() (in module kivy.compat), 201
 clock_resolution (kivy.clock.CyClockBase attribute), 197
 ClockBase (class in kivy.clock), 201
 ClockBaseBehavior (class in kivy.clock), 199
 ClockBaseFreeInterruptAll (class in kivy.clock), 201
 ClockBaseFreeInterruptOnly (class in kivy.clock), 201
 ClockBaseInterrupt (class in kivy.clock), 201
 ClockBaseInterruptBehavior (class in kivy.clock), 200
 ClockBaseInterruptFreeBehavior (class in kivy.clock), 200
 ClockEvent (class in kivy.clock), 196
 close (kivy.graphics.Line attribute), 317
 close (kivy.graphics.vertex_instructions.Line attribute), 375
 close() (kivy.base.EventLoopBase method), 222
 close() (kivy.core.window.WindowBase method), 281
 close_button (kivy.uix.settings.MenuSidebar attribute), 507

tribute), 628
 close_settings() (kivy.app.App method), 178
 cls (kivy.uix.widget.Widget attribute), 685
 cm() (in module kivy.metrics), 227
 CodeInput (class in kivy.uix.codeinput), 529
 CodeNavigationBehavior (class in kivy.uix.behaviors), 463
 CodeNavigationBehavior (class in kivy.uix.behaviors.codenavigation), 467
 col_default_width (kivy.uix.gridlayout.GridLayout attribute), 557
 col_force_default (kivy.uix.gridlayout.GridLayout attribute), 557
 collapse (kivy.uix.accordion.AccordionItem attribute), 503
 collapse_alpha (kivy.uix.accordion.AccordionItem attribute), 503
 collide_margin() (kivy.uix.vkeyboard.VKeyboard method), 677
 collide_point() (kivy.uix.scatter.Scatter method), 598
 collide_point() (kivy.uix.scatter.ScatterPlane method), 601
 collide_point() (kivy.uix.scatterlayout.ScatterPlaneLayout method), 602
 collide_point() (kivy.uix.widget.Widget method), 685
 collide_widget() (kivy.uix.widget.Widget method), 685
 Color (class in kivy.graphics), 310
 Color (class in kivy.graphics.context_instructions), 332
 color (kivy.graphics.svg.Svg attribute), 356
 color (kivy.uix.actionbar.ActionPrevious attribute), 509
 color (kivy.uix.checkbox.CheckBox attribute), 528
 color (kivy.uix.colorpicker.ColorPicker attribute), 530
 color (kivy.uix.colorpicker.ColorWheel attribute), 531
 color (kivy.uix.image.Image attribute), 560
 color (kivy.uix.label.Label attribute), 567
 color_selected (kivy.uix.treeview.TreeViewNode attribute), 667
 colorfmt (kivy.graphics.texture.Texture attribute), 364
 ColorPicker (class in kivy.uix.colorpicker), 530
 colors (kivy.uix.rst.RstDocument attribute), 593
 ColorWheel (class in kivy.uix.colorpicker), 531
 cols (kivy.uix.gridlayout.GridLayout attribute), 557
 cols_minimum (kivy.uix.gridlayout.GridLayout attribute), 557
 complete_stroke() (kivy.uix.gesturesurface.GestureSurface method), 554
 CompoundSelectionBehavior (class in kivy.uix.behaviors), 460
 CompoundSelectionBehavior (class in kivy.uix.behaviors.compoundselection), 469
 compute_visible_views() (kivy.uix.recycleboxlayout.RecycleBoxLayout method), 585
 compute_visible_views() (kivy.uix.recyclegridlayout.RecycleGridLayout method), 586
 compute_visible_views() (kivy.uix.recycleview.layout.RecycleLayoutManagerBehavior method), 496
 Config (in module kivy.config), 206
 config (kivy.app.App attribute), 178
 config (kivy.uix.settings.SettingsPanel attribute), 622
 ConfigParser (class in kivy.config), 206
 ConfigParserProperty (class in kivy.properties), 248
 Console (class in kivy.modules.console), 433
 console (kivy.modules.console.ConsoleAddon attribute), 434
 ConsoleAddon (class in kivy.modules.console), 434
 ConsoleButton (class in kivy.modules.console), 434
 ConsoleLabel (class in kivy.modules.console), 434
 ConsoleToggleButton (class in kivy.modules.console), 434
 container (kivy.uix.accordion.AccordionItem attribute), 503
 container (kivy.uix.dropdown.DropDown attribute), 534
 container (kivy.uix.settings.ContentPanel attribute), 627
 container_title (kivy.uix.accordion.AccordionItem attribute), 503
 content (kivy.uix.bubble.Bubble attribute), 519
 content (kivy.uix.popup.Popup attribute), 583
 content (kivy.uix.settings.InterfaceWithSidebar attribute), 626
 content (kivy.uix.settings.SettingItem attribute), 623
 content (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 content (kivy.uix.tabbedpanel.TabbedPanelHeader attribute), 647
 content_height (kivy.core.text.LabelBase attribute), 270
 content_size (kivy.core.text.LabelBase attribute), 270
 content_size (kivy.uix.accordion.AccordionItem attribute), 503

attribute), 503
 content_width (kivy.core.text.LabelBase attribute), 270
 ContentPanel (class in kivy.uix.settings), 626
 Context (class in kivy.context), 208
 Context (class in kivy.graphics.context), 336
 ContextInstruction (class in kivy.graphics), 311
 ContextInstruction (class in kivy.graphics.instructions), 327
 ContextualActionView (class in kivy.uix.actionbar), 509
 convert_distance_to_scroll() (kivy.uix.scrollview.ScrollView method), 615
 copy() (kivy.uix.textinput.TextInput method), 653
 copy_to() (kivy.input.MotionEvent method), 381
 copy_to() (kivy.input.motionEvent.MotionEvent method), 398
 CoreCriticalException, 259
 count() (kivy.storage.AbstractStore method), 448
 counter (kivy.input.recorder.Recorder attribute), 396
 cover_pos (kivy.uix.behaviors.cover.CoverBehavior attribute), 473
 cover_pos (kivy.uix.behaviors.CoverBehavior attribute), 464
 cover_size (kivy.uix.behaviors.cover.CoverBehavior attribute), 473
 cover_size (kivy.uix.behaviors.CoverBehavior attribute), 464
 CoverBehavior (class in kivy.uix.behaviors), 463
 CoverBehavior (class in kivy.uix.behaviors.cover), 473
 create() (kivy.atlas.Atlas static method), 188
 create() (kivy.graphics.texture.Texture static method), 364
 create() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 394
 create_from() (kivy.lang.builder.BuilderBase class method), 421
 create_from() (kivy.lang.BuilderBase class method), 416
 create_from_data() (kivy.graphics.texture.Texture static method), 364
 create_inspector() (in module kivy.modules.inspector), 435
 create_joycursor() (in module kivy.modules.joycursor), 436
 create_json_panel() (kivy.uix.settings.Settings method), 621
 create_property() (kivy.event.EventDispatcher method), 211
 create_settings() (kivy.app.App method), 179
 create_trigger() (kivy.clock.CyClockBase method), 197
 create_trigger() (kivy.clock.CyClockBaseFree method), 198
 create_trigger_free() (kivy.clock.CyClockBaseFree method), 199
 create_view() (kivy.uix.recycleview.views.RecycleDataAdapter method), 497
 create_window() (kivy.core.window.WindowBase method), 281
 current (kivy.uix.screenmanager.ScreenManager attribute), 607
 current_color (kivy.graphics.svg.Svg attribute), 357
 current_panel (kivy.uix.settings.ContentPanel attribute), 627
 current_screen (kivy.uix.screenmanager.ScreenManager attribute), 607
 current_slide (kivy.uix.carousel.Carousel attribute), 524
 current_tab (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 current_uid (kivy.uix.settings.ContentPanel attribute), 627
 cursor (kivy.uix.textinput.TextInput attribute), 653
 cursor_blink (kivy.uix.textinput.TextInput attribute), 653
 cursor_col (kivy.uix.textinput.TextInput attribute), 653
 cursor_color (kivy.uix.textinput.TextInput attribute), 653
 cursor_disabled_image (kivy.uix.slider.Slider attribute), 630
 cursor_height (kivy.uix.slider.Slider attribute), 630
 cursor_image (kivy.uix.slider.Slider attribute), 630
 cursor_index() (kivy.uix.textinput.TextInput method), 653
 cursor_offset() (kivy.uix.textinput.TextInput method), 653
 cursor_pos (kivy.uix.textinput.TextInput attribute), 653
 cursor_row (kivy.uix.textinput.TextInput attribute), 654
 cursor_size (kivy.uix.slider.Slider attribute), 630
 cursor_width (kivy.uix.slider.Slider attribute), 630
 cursor_width (kivy.uix.textinput.TextInput attribute), 654
 cut() (kivy.uix.textinput.TextInput method), 654
 CyClockBase (class in kivy.clock), 197
 CyClockBaseFree (class in kivy.clock), 198

D

DampedScrollEffect (class in kivy.effects.dampedscroll), 295
 dash_length (kivy.graphics.Bezier attribute), 306
 dash_length (kivy.graphics.Line attribute), 317
 dash_length (kivy.graphics.vertex_instructions.Bezier

attribute), 379
 dash_length (kivy.graphics.vertex_instructions.Line attribute), 375
 dash_offset (kivy.graphics.Bezier attribute), 306
 dash_offset (kivy.graphics.Line attribute), 317
 dash_offset (kivy.graphics.vertex_instructions.Bezier attribute), 379
 dash_offset (kivy.graphics.vertex_instructions.Line attribute), 375
 dashes (kivy.graphics.Line attribute), 317
 dashes (kivy.graphics.vertex_instructions.Line attribute), 375
 data (kivy.core.image.ImageData attribute), 266
 data (kivy.uix.recycleview.datamodel.RecycleDataModel attribute), 494
 data (kivy.uix.recycleview.RecycleView attribute), 492
 data_model (kivy.uix.recycleview.RecycleViewBehavior attribute), 491
 DDSException, 426
 deactivate() (kivy.modules.console.ConsoleAddon method), 434
 decode_result() (kivy.network.urlrequest.UrlRequest method), 443
 default_size (kivy.uix.recyclelayout.RecycleLayout attribute), 586
 default_tab (kivy.uix.tabbedpanel.TabbedPanel attribute), 645
 default_tab_cls (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 default_tab_content (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 default_tab_text (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 defaultvalue (kivy.properties.Property attribute), 242
 delete() (kivy.storage.AbstractStore method), 448
 delete_selection() (kivy.uix.textinput.TextInput method), 654
 delete_word_left() (kivy.uix.behaviors.emacs.EmacsBehavior method), 476
 delete_word_left() (kivy.uix.behaviors.EmacsBehavior method), 463
 delete_word_right() (kivy.uix.behaviors.emacs.EmacsBehavior method), 476
 delete_word_right() (kivy.uix.behaviors.EmacsBehavior method), 463
 density() (kivy.metrics.MetricsBase method), 227
 unpack() (kivy.input.MotionEvent method), 382
 unpack() (kivy.input.motionevent.MotionEvent method), 398
 unpack() (kivy.input.providers.hidinput.HIDMotionEvent method), 391
 unpack() (kivy.input.providers.leapfinger.LeapFingerEvent method), 389
 unpack() (kivy.input.providers.linuxwacom.LinuxWacomMotionEvent method), 393
 unpack() (kivy.input.providers.mtdev.MTDMotionEvent method), 392
 unpack() (kivy.input.providers.tuio.Tuio2dCurMotionEvent method), 395
 unpack() (kivy.input.providers.tuio.Tuio2dObjMotionEvent method), 395
 unpack() (kivy.input.providers.wm_pen.WM_Pen method), 393
 unpack() (kivy.input.providers.wm_touch.WM_MotionEvent method), 393
 deprecated() (in module kivy.utils), 253
 deselect (kivy.uix.settings.SettingItem attribute), 623
 deselect_node() (kivy.uix.behaviors.compoundselection.CompoundSelection method), 469
 deselect_node() (kivy.uix.behaviors.CompoundSelectionBehavior method), 460
 deselect_node() (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 495
 deselect_node() (kivy.uix.treeview.TreeView method), 664
 destroy_settings() (kivy.app.App method), 179
 detach_recycleview() (kivy.uix.recycleview.datamodel.RecycleDataModel method), 494
 detach_recycleview() (kivy.uix.recycleview.datamodel.RecycleDataModelBehavior method), 493
 detach_recycleview() (kivy.uix.recycleview.views.RecycleDataAdapter method), 497
 device (kivy.input.MotionEvent attribute), 382
 device (kivy.input.motionevent.MotionEvent attribute), 398
 DictProperty (class in kivy.properties), 248
 DictStore (class in kivy.storage.dictstore), 449
 difference() (in module kivy.utils), 252
 direction (kivy.uix.carousel.Carousel attribute), 524
 direction (kivy.uix.screenmanager.SlideTransition attribute), 610
 directory (kivy.app.App attribute), 179
 disabled (kivy.uix.filechooser.FileChooserController attribute), 546
 deselect (kivy.uix.settings.SettingPath attribute), 624
 disabled (kivy.uix.settings.SettingItem attribute), 623
 disabled (kivy.uix.widget.Widget attribute), 686
 disabled_color (kivy.uix.label.Label attribute), 567
 disabled_foreground_color (kivy.uix.textinput.TextInput attribute), 654
 disabled_outline_color (kivy.uix.label.Label attribute), 567

attribute), 567
 dismiss() (kivy.uix.dropdown.DropDown method), 534
 dismiss() (kivy.uix.modalview.ModalView method), 579
 dismiss_on_select (kivy.uix.dropdown.DropDown attribute), 534
 dispatch() (kivy.event.EventDispatcher method), 211
 dispatch() (kivy.properties.AliasProperty method), 247
 dispatch() (kivy.properties.Property method), 242
 dispatch_children() (kivy.event.EventDispatcher method), 212
 dispatch_generic() (kivy.event.EventDispatcher method), 212
 dispatch_input() (kivy.base.EventLoopBase method), 222
 displacement (kivy.effects.scroll.ScrollEffect attribute), 298
 display_border (kivy.graphics.BorderImage attribute), 308
 display_border (kivy.graphics.vertex_instructions.BorderImage attribute), 372
 display_settings() (kivy.app.App method), 179
 distance() (kivy.input.MotionEvent method), 382
 distance() (kivy.input.motionevent.MotionEvent method), 399
 distance() (kivy.vector.Vector method), 256
 distance2() (kivy.vector.Vector method), 256
 do_backspace() (kivy.uix.textinput.TextInput method), 654
 do_collide_after_children (kivy.uix.scatter.Scatter attribute), 598
 do_cursor_movement() (kivy.uix.textinput.TextInput method), 654
 do_default_tab (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 do_layout() (kivy.uix.anchorlayout.AnchorLayout method), 512
 do_layout() (kivy.uix.boxlayout.BoxLayout method), 514
 do_layout() (kivy.uix.floatlayout.FloatLayout method), 551
 do_layout() (kivy.uix.gridlayout.GridLayout method), 558
 do_layout() (kivy.uix.layout.Layout method), 576
 do_layout() (kivy.uix.pagelayout.PageLayout method), 580
 do_layout() (kivy.uix.recyclelayout.RecycleLayout method), 586
 do_layout() (kivy.uix.relativelayout.RelativeLayout method), 591
 do_layout() (kivy.uix.stacklayout.StackLayout method), 638
 do_redo() (kivy.uix.textinput.TextInput method), 655
 do_rotation (kivy.uix.scatter.Scatter attribute), 598
 do_scale (kivy.uix.scatter.Scatter attribute), 599
 do_scroll (kivy.uix.scrollview.ScrollView attribute), 615
 do_scroll_x (kivy.uix.scrollview.ScrollView attribute), 615
 do_scroll_y (kivy.uix.scrollview.ScrollView attribute), 615
 do_translation (kivy.uix.scatter.Scatter attribute), 599
 do_translation_x (kivy.uix.scatter.Scatter attribute), 599
 do_translation_y (kivy.uix.scatter.Scatter attribute), 599
 do_undo() (kivy.uix.textinput.TextInput method), 655
 docked (kivy.uix.vkeyboard.VKeyboard attribute), 677
 document_root (kivy.uix.rst.RstDocument attribute), 593
 dot() (kivy.vector.Vector method), 256
 dot_product() (kivy.gesture.Gesture method), 218
 double_tap_time (kivy.input.MotionEvent attribute), 382
 double_tap_time (kivy.input.motionevent.MotionEvent attribute), 399
 dp() (in module kivy.metrics), 227
 dpi() (kivy.core.window.WindowBase method), 282
 dpi() (kivy.metrics.MetricsBase method), 227
 dpi_rounded() (kivy.metrics.MetricsBase method), 227
 dpos (kivy.input.MotionEvent attribute), 382
 dpos (kivy.input.motionevent.MotionEvent attribute), 399
 drag_distance (kivy.uix.behaviors.drag.DragBehavior attribute), 474
 drag_distance (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_rect_height (kivy.uix.behaviors.drag.DragBehavior attribute), 475
 drag_rect_height (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_rect_width (kivy.uix.behaviors.drag.DragBehavior attribute), 475
 drag_rect_width (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_rect_x (kivy.uix.behaviors.drag.DragBehavior attribute), 475
 drag_rect_x (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_rect_y (kivy.uix.behaviors.drag.DragBehavior

attribute), 475
 drag_rect_y (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_rectangle (kivy.uix.behaviors.drag.DragBehavior attribute), 475
 drag_rectangle (kivy.uix.behaviors.DragBehavior attribute), 456
 drag_threshold (kivy.effects.scroll.ScrollEffect attribute), 298
 drag_timeout (kivy.uix.behaviors.drag.DragBehavior attribute), 475
 drag_timeout (kivy.uix.behaviors.DragBehavior attribute), 457
 DragBehavior (class in kivy.uix.behaviors), 456
 DragBehavior (class in kivy.uix.behaviors.drag), 474
 draw() (kivy.graphics.Canvas method), 309
 draw() (kivy.graphics.instructions.Canvas method), 329
 DropDown (class in kivy.uix.dropdown), 533
 dropdown_cls (kivy.uix.spinner.Spinner attribute), 634
 dropdown_width (kivy.uix.actionbar.ActionGroup attribute), 507
 dsx (kivy.input.MotionEvent attribute), 382
 dsx (kivy.input.motionevent.MotionEvent attribute), 399
 dsy (kivy.input.MotionEvent attribute), 382
 dsy (kivy.input.motionevent.MotionEvent attribute), 399
 dsz (kivy.input.MotionEvent attribute), 382
 dsz (kivy.input.motionevent.MotionEvent attribute), 399
 duration (kivy.animation.Animation attribute), 162
 duration (kivy.core.video.VideoBase attribute), 277
 duration (kivy.uix.screenmanager.FallOutTransition attribute), 611
 duration (kivy.uix.screenmanager.RiseInTransition attribute), 611
 duration (kivy.uix.screenmanager.TransitionBase attribute), 609
 duration (kivy.uix.video.Video attribute), 668
 duration (kivy.uix.videoplayer.VideoPlayer attribute), 672
 duration (kivy.uix.videoplayer.VideoPlayerAnnotation attribute), 674
 dx (kivy.input.MotionEvent attribute), 382
 dx (kivy.input.motionevent.MotionEvent attribute), 399
 dy (kivy.input.MotionEvent attribute), 382
 dy (kivy.input.motionevent.MotionEvent attribute), 399
 dz (kivy.input.MotionEvent attribute), 382
 dz (kivy.input.motionevent.MotionEvent attribute), 399
E
 edge_damping (kivy.effects.dampedscroll.DampedScrollEffect attribute), 295
 effect_cls (kivy.uix.scrollview.ScrollView attribute), 615
 effect_x (kivy.uix.scrollview.ScrollView attribute), 615
 effect_y (kivy.uix.scrollview.ScrollView attribute), 615
 EffectBase (class in kivy.uix.effectwidget), 538
 effects (kivy.uix.effectwidget.EffectWidget attribute), 538
 EffectWidget (class in kivy.uix.effectwidget), 537
 element_count (kivy.graphics.tessellator.Tessellator attribute), 359
 Ellipse (class in kivy.graphics), 311
 Ellipse (class in kivy.graphics.vertex_instructions), 372
 ellipse (kivy.graphics.Line attribute), 317
 ellipse (kivy.graphics.vertex_instructions.Line attribute), 375
 ellipsis_options (kivy.uix.label.Label attribute), 568
 EmacsBehavior (class in kivy.uix.behaviors), 463
 EmacsBehavior (class in kivy.uix.behaviors.emacs), 476
 emit() (kivy.logger.LoggerHistory method), 225
 ensure_window() (kivy.base.EventLoopBase method), 223
 entry_released() (kivy.uix.filechooser.FileChooserController method), 546
 entry_touched() (kivy.uix.filechooser.FileChooserController method), 546
 eos (kivy.uix.video.Video attribute), 668
 error (kivy.network.urlrequest.UrlRequest attribute), 443
 error_image (kivy.loader.LoaderBase attribute), 185
 escape_markup() (in module kivy.utils), 253
 even_color (kivy.uix.treeview.TreeViewNode attribute), 667
 EventDispatcher (class in kivy.event), 208
 EventLoop (in module kivy.base), 222
 EventLoopBase (class in kivy.base), 222
 events() (kivy.event.EventDispatcher method), 212
 ExceptionHandler (class in kivy.base), 223
 ExceptionManager (in module kivy.base), 224
 ExceptionManagerBase (class in kivy.base), 224
 exists() (kivy.storage.AbstractStore method), 448
 exit() (kivy.base.EventLoopBase method), 223
 export_as_image() (kivy.uix.widget.Widget method), 686

export_gesture() (kivy.multistroke.Recognizer method), 230
 export_to_png() (kivy.uix.widget.Widget method), 686
F
 Factory (in module kivy.factory), 217
 FactoryException, 217
 FadeTransition (class in kivy.uix.screenmanager), 611
 FallOutTransition (class in kivy.uix.screenmanager), 611
 fbind() (kivy.event.EventDispatcher method), 212
 fbind() (kivy.event.Observable method), 216
 fbind() (kivy.lang.builder.Observable method), 419
 fbind() (kivy.lang.Observable method), 415
 Fbo (class in kivy.graphics), 312
 Fbo (class in kivy.graphics.fbo), 337
 fbo (kivy.uix.effectwidget.EffectBase attribute), 538
 fbo_list (kivy.uix.effectwidget.EffectWidget attribute), 538
 file_encodings (kivy.uix.filechooser.FileChooserController attribute), 546
 file_system (kivy.uix.filechooser.FileChooserController attribute), 547
 FileChooser (class in kivy.uix.filechooser), 545
 FileChooserController (class in kivy.uix.filechooser), 546
 FileChooserIconLayout (class in kivy.uix.filechooser), 545
 FileChooserIconView (class in kivy.uix.filechooser), 544
 FileChooserListLayout (class in kivy.uix.filechooser), 544
 FileChooserListView (class in kivy.uix.filechooser), 544
 FileChooserProgressBase (class in kivy.uix.filechooser), 549
 filename (kivy.atlas.Atlas attribute), 189
 filename (kivy.core.audio.Sound attribute), 260
 filename (kivy.core.image.Image attribute), 264
 filename (kivy.core.video.VideoBase attribute), 277
 filename (kivy.input.recorder.Recorder attribute), 397
 files (kivy.uix.filechooser.FileChooserController attribute), 547
 FileSystemAbstract (class in kivy.uix.filechooser), 549
 FileSystemLocal (class in kivy.uix.filechooser), 550
 filter() (kivy.multistroke.Recognizer method), 230
 filter_dirs (kivy.uix.filechooser.FileChooserController attribute), 547
 filters (kivy.uix.filechooser.FileChooserController attribute), 547
 find() (kivy.gesture.GestureDatabase method), 219
 find() (kivy.storage.AbstractStore method), 448
 find_base_direction() (kivy.core.text.LabelBase static method), 270
 find_colliding_gesture() (kivy.uix.gesturesurface.GestureSurface method), 553
 find_double_tap() (kivy.input.postproc.doubletap.InputPostprocDoubleTap method), 387
 find_triple_tap() (kivy.input.postproc.tripletap.InputPostprocTripleTap method), 388
 flag_update() (kivy.graphics.Instruction method), 313
 flag_update() (kivy.graphics.instructions.Instruction method), 327
 flag_update_canvas() (kivy.graphics.context.Context method), 336
 flip() (kivy.core.window.WindowBase method), 282
 flip_horizontal() (kivy.graphics.texture.Texture method), 365
 flip_vertical (kivy.core.image.ImageData attribute), 266
 flip_vertical() (kivy.graphics.texture.Texture method), 365
 FloatLayout (class in kivy.uix.floatlayout), 551
 flush() (kivy.graphics.context.Context method), 336
 flush() (kivy.logger.LoggerHistory method), 226
 fmt (kivy.core.image.ImageData attribute), 266
 focus (kivy.core.window.WindowBase attribute), 282
 focus (kivy.uix.behaviors.focus.FocusBehavior attribute), 477
 focus (kivy.uix.behaviors.FocusBehavior attribute), 457
 focus_next (kivy.uix.behaviors.focus.FocusBehavior attribute), 478
 focus_next (kivy.uix.behaviors.FocusBehavior attribute), 457
 focus_previous (kivy.uix.behaviors.focus.FocusBehavior attribute), 478
 focus_previous (kivy.uix.behaviors.FocusBehavior attribute), 457
 FocusBehavior (class in kivy.uix.behaviors), 457
 FocusBehavior (class in kivy.uix.behaviors.focus), 477
 focused (kivy.uix.behaviors.focus.FocusBehavior attribute), 478
 focused (kivy.uix.behaviors.FocusBehavior attribute), 457
 font_blended (kivy.uix.label.Label attribute), 568

font_context (kivy.uix.label.Label attribute), 568
font_context (kivy.uix.textinput.TextInput attribute), 655
font_family (kivy.uix.label.Label attribute), 568
font_family (kivy.uix.textinput.TextInput attribute), 655
font_features (kivy.uix.label.Label attribute), 569
font_hinting (kivy.uix.label.Label attribute), 569
font_kerning (kivy.uix.label.Label attribute), 569
font_name (kivy.uix.colorpicker.ColorPicker attribute), 530
font_name (kivy.uix.label.Label attribute), 569
font_name (kivy.uix.textinput.TextInput attribute), 655
font_size (kivy.uix.label.Label attribute), 569
font_size (kivy.uix.textinput.TextInput attribute), 656
font_size (kivy.uix.vkeyboard.VKeyboard attribute), 677
fontid (kivy.core.text.LabelBase attribute), 271
fontscale() (kivy.metrics.MetricsBase method), 227
foreground_color (kivy.uix.textinput.TextInput attribute), 656
fork() (kivy.uix.behaviors.knspace.KNSpace method), 484
frames (kivy.clock.ClockBaseBehavior attribute), 199
frames_displayed (kivy.clock.ClockBaseBehavior attribute), 199
frametime (kivy.clock.ClockBaseBehavior attribute), 199
free (kivy.clock.FreeClockEvent attribute), 197
FreeClockEvent (class in kivy.clock), 197
friction (kivy.effects.kinetic.KineticEffect attribute), 296
fs (kivy.graphics.shader.Shader attribute), 354
fs (kivy.uix.screenmanager.ShaderTransition attribute), 610
fullscreen (kivy.core.window.WindowBase attribute), 282
fullscreen (kivy.uix.videoplayer.VideoPlayer attribute), 672
funbind() (kivy.event.EventDispatcher method), 213
funbind() (kivy.event.Observable method), 216
funbind() (kivy.lang.builder.Observable method), 419
funbind() (kivy.lang.Observable method), 415
func_op (kivy.graphics.stencil_instructions.StencilUse attribute), 356
func_op (kivy.graphics.StencilUse attribute), 323
FXAAEffect (class in kivy.uix.effectwidget), 540

G
g (kivy.graphics.ClearColor attribute), 325
g (kivy.graphics.Color attribute), 311
g (kivy.graphics.context_instructions.Color attribute), 333
g (kivy.graphics.gl_instructions.ClearColor attribute), 339
g (kivy.uix.colorpicker.ColorWheel attribute), 531
garden_system_dir (in module kivy.garden), 303
Gesture (class in kivy.gesture), 218
gesture_to_str() (kivy.gesture.GestureDatabase method), 219
GestureContainer (class in kivy.uix.gesturesurface), 554
GestureDatabase (class in kivy.gesture), 219
GestureStroke (class in kivy.gesture), 219
GestureSurface (class in kivy.uix.gesturesurface), 552
get() (kivy.cache.Cache static method), 190
get() (kivy.config.ConfigParser method), 207
get() (kivy.graphics.transformation.Matrix method), 367
get() (kivy.input.factory.MotionEventFactory static method), 402
get() (kivy.input.MotionEventFactory static method), 385
get() (kivy.properties.AliasProperty method), 247
get() (kivy.properties.Property method), 242
get() (kivy.properties.ReferenceListProperty method), 246
get() (kivy.storage.AbstractStore method), 448
get_angle_similarity() (kivy.multistroke.Candidate method), 236
get_application_config() (kivy.app.App method), 179
get_application_icon() (kivy.app.App method), 180
get_application_name() (kivy.app.App method), 180
get_boottime() (kivy.clock.ClockBaseBehavior method), 199
get_cached_extents() (kivy.core.text.LabelBase method), 271
get_callback() (kivy.clock.ClockEvent method), 196
get_color_from_hex() (in module kivy.utils), 252
get_configparser() (kivy.config.ConfigParser static method), 207
get_connection_for_scheme() (kivy.network.urlrequest.UrlRequest method), 443
get_current_context() (in module kivy.context), 208
get_cursor_from_index() (kivy.uix.textinput.TextInput method), 656

`get_cursor_from_xy()` (kivy.uix.textinput.TextInput method), 664
`get_distance()` (kivy.multistroke.MultistrokeGesture method), 234
`get_events()` (kivy.clock.CyClockBase method), 197
`get_extents()` (kivy.core.text.LabelBase method), 271
`get_focus_next()` (kivy.uix.behaviors.focus.FocusBehavior method), 478
`get_focus_next()` (kivy.uix.behaviors.FocusBehavior method), 457
`get_focus_previous()` (kivy.uix.behaviors.focus.FocusBehavior method), 478
`get_focus_previous()` (kivy.uix.behaviors.FocusBehavior method), 458
`get_format()` (kivy.properties.NumericProperty method), 243
`get_fps()` (kivy.clock.ClockBaseBehavior method), 199
`get_gesture()` (kivy.uix.gesturesurface.GestureSurface method), 553
`get_gl_backend_name()` (kivy.core.window.WindowBase method), 282
`get_group()` (kivy.graphics.InstructionGroup method), 314
`get_group()` (kivy.graphics.instructions.InstructionGroup method), 327
`get_hex_from_color()` (in module kivy.utils), 252
`get_index_of_node()` (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 469
`get_index_of_node()` (kivy.uix.behaviors.CompoundSelectionBehavior method), 460
`get_index_of_node()` (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 495
`get_index_of_node()` (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 495
`get_lastaccess()` (kivy.cache.Cache static method), 190
`get_max()` (kivy.properties.BoundedNumericProperty method), 245
`get_min()` (kivy.properties.BoundedNumericProperty method), 245
`get_min_free_timeout()` (kivy.clock.CyClockBaseFree method), 199
`get_min_timeout()` (kivy.clock.CyClockBase method), 197
`get_mipmap()` (kivy.core.image.ImageData method), 266
`get_nice_size()` (kivy.uix.filechooser.FileChooserController method), 547
`get_node_at_pos()` (kivy.uix.treeview.TreeView method), 664
`get_parent_window()` (kivy.uix.widget.Widget method), 686
`get_pixel_color()` (kivy.graphics.Fbo method), 313
`get_pixel_color()` (kivy.graphics.fbo.Fbo method), 338
`get_pos()` (kivy.core.audio.Sound method), 260
`get_property_observers()` (kivy.event.EventDispatcher method), 213
`get_protractor_vector()` (kivy.multistroke.Candidate method), 236
`get_random_color()` (in module kivy.utils), 252
`get_region()` (kivy.graphics.texture.Texture method), 365
`get_resolution()` (kivy.clock.CyClockBase method), 197
`get_rfps()` (kivy.clock.ClockBaseBehavior method), 199
`get_rigid_rotation()` (kivy.gesture.Gesture method), 218
`get_root_window()` (kivy.uix.widget.Widget method), 687
`get_running_app()` (kivy.app.App static method), 180
`get_score()` (kivy.gesture.Gesture method), 219
`get_screen()` (kivy.uix.screenmanager.ScreenManager method), 607
`get_selectable_nodes()` (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 469
`get_selectable_nodes()` (kivy.uix.behaviors.CompoundSelectionBehavior method), 460
`get_selectable_nodes()` (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 495
`get_start_unit_vector()` (kivy.multistroke.Candidate method), 236
`get_system_fonts_dir()` (kivy.core.text.LabelBase static method), 271
`get_time()` (kivy.clock.ClockBaseBehavior method), 200
`get_timestamp()` (kivy.cache.Cache static method), 190
`get_value()` (kivy.uix.settings.SettingsPanel method), 622
`get_vectors()` (kivy.uix.gesturesurface.GestureContainer method), 554
`get_view()` (kivy.uix.recycleview.views.RecycleDataAdapter method), 497
`get_view_index_at()` (kivy.uix.recycleboxlayout.RecycleBoxLayout method), 585

get_view_index_at() (kivy.uix.recyclegridlayout.RecycleGridLayout method), 586	glGetInteger() (in module kivy.graphics.opengl_utils), 350
get_view_index_at() (kivy.uix.recycleview.layout.RecycleViewLayoutManager method), 496	glGetTextureManager() (in module kivy.graphics.opengl), 341
get_visible_view() (kivy.uix.recycleview.views.RecycleView.views.RecycleView method), 498	glDetachShader() (in module kivy.graphics.opengl), 341
get_widgets() (kivy.uix.behaviors.togglebutton.ToggleButtonBehavior static method), 486	glBindAttribLocation() (in module kivy.graphics.opengl), 341
get_widgets() (kivy.uix.behaviors.ToggleButtonBehavior static method), 455	glBindBuffer() (in module kivy.graphics.opengl), 341
get_window_matrix() (kivy.uix.widget.Widget method), 687	glBindFramebuffer() (in module kivy.graphics.opengl), 341
getdefault() (kivy.config.ConfigParser method), 207	glBindRenderbuffer() (in module kivy.graphics.opengl), 341
getdefaulttint() (kivy.config.ConfigParser method), 207	glBindTexture() (in module kivy.graphics.opengl), 341
getsize() (kivy.uix.filechooser.FileSystemAbstract method), 549	glBlendColor() (in module kivy.graphics.opengl), 341
getsize() (kivy.uix.filechooser.FileSystemLocal method), 550	glBlendEquation() (in module kivy.graphics.opengl), 341
getter() (kivy.event.EventDispatcher method), 214	glBlendEquationSeparate() (in module kivy.graphics.opengl), 341
gl_backends_allowed (kivy.core.window.WindowBase attribute), 282	glBlendFunc() (in module kivy.graphics.opengl), 341
gl_backends_ignored (kivy.core.window.WindowBase attribute), 282	glBlendFuncSeparate() (in module kivy.graphics.opengl), 341
gl_dealloc() (kivy.graphics.context.Context method), 336	glBufferData() (in module kivy.graphics.opengl), 341
gl_get_extensions() (in module kivy.graphics.opengl_utils), 350	glBufferSubData() (in module kivy.graphics.opengl), 341
gl_get_texture_formats() (in module kivy.graphics.opengl_utils), 351	glCheckFramebufferStatus() (in module kivy.graphics.opengl), 341
gl_get_version() (in module kivy.graphics.opengl_utils), 351	glClear() (in module kivy.graphics.opengl), 342
gl_get_version_major() (in module kivy.graphics.opengl_utils), 351	glClearColor() (in module kivy.graphics.opengl), 342
gl_get_version_minor() (in module kivy.graphics.opengl_utils), 351	glClearStencil() (in module kivy.graphics.opengl), 342
gl_has_capability() (in module kivy.graphics.opengl_utils), 350	glColorMask() (in module kivy.graphics.opengl), 342
gl_has_extension() (in module kivy.graphics.opengl_utils), 350	glCompileShader() (in module kivy.graphics.opengl), 342
gl_has_texture_conversion() (in module kivy.graphics.opengl_utils), 351	glCompressedTexImage2D() (in module kivy.graphics.opengl), 342
gl_has_texture_format() (in module kivy.graphics.opengl_utils), 351	glCompressedTexSubImage2D() (in module kivy.graphics.opengl), 342
gl_has_texture_native_format() (in module kivy.graphics.opengl_utils), 351	glCopyTexImage2D() (in module kivy.graphics.opengl), 342
gl_init_resources() (in module kivy.graphics), 326	glCopyTexSubImage2D() (in module kivy.graphics.opengl), 342
gl_init_resources() (in module kivy.graphics.context_instructions), 335	glCreateProgram() (in module kivy.graphics.opengl), 342
gl_init_symbols() (in module kivy.graphics.opengl), 350	glCreateShader() (in module kivy.graphics.opengl), 342
	glCullFace() (in module kivy.graphics.opengl), 342
	glDeleteBuffers() (in module kivy.graphics.opengl), 342

kivy.graphics.opengl), 342			
glDeleteFramebuffers()	(in module		
kivy.graphics.opengl), 342			
glDeleteProgram()	(in module		
kivy.graphics.opengl), 342			
glDeleteRenderbuffers()	(in module		
kivy.graphics.opengl), 342			
glDeleteShader()	(in module		
kivy.graphics.opengl), 342			
glDeleteTextures()	(in module		
kivy.graphics.opengl), 343			
glDepthFunc()	(in module kivy.graphics.opengl),		
343			
glDepthMask()	(in module kivy.graphics.opengl),		
343			
glDetachShader()	(in module		
kivy.graphics.opengl), 343			
glDisable()	(in module kivy.graphics.opengl), 343		
glDisableVertexAttribArray()	(in module		
kivy.graphics.opengl), 343			
glDrawArrays()	(in module kivy.graphics.opengl),		
343			
glDrawElements()	(in module		
kivy.graphics.opengl), 343			
glEnable()	(in module kivy.graphics.opengl), 343		
glEnableVertexAttribArray()	(in module		
kivy.graphics.opengl), 343			
glFinish()	(in module kivy.graphics.opengl), 343		
glFlush()	(in module kivy.graphics.opengl), 343		
glFramebufferRenderbuffer()	(in module		
kivy.graphics.opengl), 343			
glFramebufferTexture2D()	(in module		
kivy.graphics.opengl), 343			
glFrontFace()	(in module kivy.graphics.opengl),		
343			
glGenBuffers()	(in module kivy.graphics.opengl),		
343			
glGenerateMipmap()	(in module		
kivy.graphics.opengl), 344			
glGenFramebuffers()	(in module		
kivy.graphics.opengl), 343			
glGenRenderbuffers()	(in module		
kivy.graphics.opengl), 343			
glGenTextures()	(in module kivy.graphics.opengl),		
343			
glGetActiveAttrib()	(in module		
kivy.graphics.opengl), 344			
glGetActiveUniform()	(in module		
kivy.graphics.opengl), 344			
glGetAttachedShaders()	(in module		
kivy.graphics.opengl), 344			
glGetAttribLocation()	(in module		
kivy.graphics.opengl), 344			
glGetBooleanv()	(in module		
kivy.graphics.opengl), 344			
glGetBufferParameteriv()	(in module		
kivy.graphics.opengl), 344			
glGetError()	(in module kivy.graphics.opengl), 344		
glGetFloatv()	(in module kivy.graphics.opengl),		
344			
glGetFramebufferAttachmentParameteriv()	(in module kivy.graphics.opengl), 344		
glGetIntegerv()	(in module kivy.graphics.opengl),		
344			
glGetProgramInfoLog()	(in module		
kivy.graphics.opengl), 344			
glGetProgramiv()	(in module		
kivy.graphics.opengl), 344			
glGetRenderbufferParameteriv()	(in module		
kivy.graphics.opengl), 345			
glGetShaderInfoLog()	(in module		
kivy.graphics.opengl), 345			
glGetShaderiv()	(in module kivy.graphics.opengl),		
345			
glGetShaderPrecisionFormat()	(in module		
kivy.graphics.opengl), 345			
glGetShaderSource()	(in module		
kivy.graphics.opengl), 345			
glGetString()	(in module kivy.graphics.opengl),		
345			
glGetTexParameterfv()	(in module		
kivy.graphics.opengl), 345			
glGetTexParameteriv()	(in module		
kivy.graphics.opengl), 345			
glGetUniformfv()	(in module		
kivy.graphics.opengl), 345			
glGetUniformiv()	(in module		
kivy.graphics.opengl), 345			
glGetUniformLocation()	(in module		
kivy.graphics.opengl), 345			
glGetVertexAttribfv()	(in module		
kivy.graphics.opengl), 345			
glGetVertexAttribiv()	(in module		
kivy.graphics.opengl), 345			
glGetVertexAttribPointerv()	(in module		
kivy.graphics.opengl), 345			
glHint()	(in module kivy.graphics.opengl), 345		
glIsBuffer()	(in module kivy.graphics.opengl), 345		
glIsEnabled()	(in module kivy.graphics.opengl),		
346			
glIsFramebuffer()	(in module		
kivy.graphics.opengl), 346			
glIsProgram()	(in module kivy.graphics.opengl),		
346			
glIsRenderbuffer()	(in module		
kivy.graphics.opengl), 346			
glIsShader()	(in module kivy.graphics.opengl), 346		
glIsTexture()	(in module kivy.graphics.opengl),		
346			
glLineWidth()	(in module kivy.graphics.opengl),		

[346](#)
[glLinkProgram\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glPixelStorei\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glPolygonOffset\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glReadPixels\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glReleaseShaderCompiler\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glRenderbufferStorage\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glSampleCoverage\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glScissor\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glShaderBinary\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glShaderSource\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glsl](#) ([kivy.uix.effectwidget.EffectBase](#) attribute), [538](#)
[glStencilFunc\(\)](#) (in module [kivy.graphics.opengl](#)), [346](#)
[glStencilFuncSeparate\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glStencilMask\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glStencilMaskSeparate\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glStencilOp\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glStencilOpSeparate\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexImage2D\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexParameterf\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexParameterfv\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexParameteri\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexParameteriv\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glTexSubImage2D\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glUniform1f\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glUniform1fv\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glUniform1i\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glUniform1iv\(\)](#) (in module [kivy.graphics.opengl](#)), [347](#)
[glUniform2f\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform2fv\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform2i\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform2iv\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform3f\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform3fv\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform3i\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform3iv\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform4f\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform4fv\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform4i\(\)](#) (in module [kivy.graphics.opengl](#)), [348](#)
[glUniform4iv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glUniformMatrix2fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glUniformMatrix3fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glUniformMatrix4fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glUseProgram\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glValidateProgram\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib1f\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib1fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib2f\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib2fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib3f\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib3fv\(\)](#) (in module [kivy.graphics.opengl](#)), [349](#)
[glVertexAttrib4f\(\)](#) (in module [kivy.graphics.opengl](#)), [350](#)
[glVertexAttrib4fv\(\)](#) (in module [kivy.graphics.opengl](#)), [350](#)
[glVertexAttribPointer\(\)](#) (in module [kivy.graphics.opengl](#)), [350](#)
[glViewport\(\)](#) (in module [kivy.graphics.opengl](#)), [350](#)
[goto\(\)](#) ([kivy.uix.rst.RstDocument](#) method), [593](#)
[goto_node\(\)](#) ([kivy.uix.behaviors.compoundselection.CompoundSelection](#) method), [593](#)

method), 470
 goto_node() (kivy.uix.behaviors.CompoundSelectionBehavior method), 607
 method), 460
 goto_node() (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 495
 method), 162
 goto_view() (kivy.uix.recycleview.layout.RecycleViewBehavior method), 496
 height (kivy.core.image.Image attribute), 264
 height (kivy.core.image.ImageData attribute), 266
 height (kivy.core.window.WindowBase attribute), 283
 grab() (kivy.input.MotionEvent method), 382
 grab() (kivy.input.motionevent.MotionEvent method), 399
 grab_current (kivy.input.MotionEvent attribute), 382
 grab_current (kivy.input.motionevent.MotionEvent attribute), 399
 grab_mouse() (kivy.core.window.WindowBase method), 282
 gradients (kivy.graphics.svg.Svg attribute), 357
 GraphicException, 313, 378
 GridLayout (class in kivy.uix.gridlayout), 557
 GridLayoutException, 559
 group (kivy.graphics.Instruction attribute), 313
 group (kivy.graphics.instructions.Instruction attribute), 327
 group (kivy.uix.behaviors.togglebutton.ToggleButton attribute), 487
 group (kivy.uix.behaviors.ToggleButtonBehavior attribute), 456
H
 h (kivy.core.text.text_layout.LayoutLine attribute), 274
 h (kivy.graphics.Color attribute), 311
 h (kivy.graphics.context_instructions.Color attribute), 333
 halign (kivy.uix.label.Label attribute), 570
 halign (kivy.uix.textinput.TextInput attribute), 656
 handle_exception() (kivy.base.ExceptionHandler method), 224
 handle_exception() (kivy.base.ExceptionManagerBase method), 224
 handle_image_left (kivy.uix.textinput.TextInput attribute), 656
 handle_image_middle (kivy.uix.textinput.TextInput attribute), 656
 handle_image_right (kivy.uix.textinput.TextInput attribute), 656
 handles() (kivy.uix.gesturesurface.GestureContainer method), 555
 has_after (kivy.graphics.Canvas attribute), 309
 has_after (kivy.graphics.instructions.Canvas attribute), 329
 has_before (kivy.graphics.Canvas attribute), 309
 has_before (kivy.graphics.instructions.Canvas attribute), 329
 has_screen() (kivy.uix.screenmanager.ScreenManager method), 460
 have_properties_to_animate() (kivy.animation.Animation method), 162
 height (kivy.core.image.Image attribute), 264
 height (kivy.core.image.ImageData attribute), 266
 height (kivy.core.window.WindowBase attribute), 283
 height (kivy.graphics.svg.Svg attribute), 357
 height (kivy.graphics.texture.Texture attribute), 365
 height (kivy.input.shape.ShapeRect attribute), 402
 height (kivy.uix.widget.Widget attribute), 687
 hex_color (kivy.uix.colorpicker.ColorPicker attribute), 531
 hide() (kivy.core.window.WindowBase method), 283
 hide_keyboard() (kivy.uix.behaviors.focus.FocusBehavior method), 478
 hide_keyboard() (kivy.uix.behaviors.FocusBehavior method), 458
 hide_root (kivy.uix.treeview.TreeView attribute), 664
 HIDMotionEvent (class in kivy.input.providers.hidinput), 391
 highlight_at() (kivy.modules.console.Console method), 433
 hint_text (kivy.uix.textinput.TextInput attribute), 657
 hint_text_color (kivy.uix.textinput.TextInput attribute), 657
 HorizontalBlurEffect (class in kivy.uix.effectwidget), 540
 hsv (kivy.graphics.Color attribute), 311
 hsv (kivy.graphics.context_instructions.Color attribute), 333
 hsv (kivy.uix.colorpicker.ColorPicker attribute), 531
 icon (kivy.app.App attribute), 180
 icon (kivy.core.window.WindowBase attribute), 283
 icon (kivy.uix.actionbar.ActionButton attribute), 506
 icon (kivy.uix.actionbar.ActionToggleButton attribute), 506
 id (kivy.graphics.texture.Texture attribute), 365
 id (kivy.input.MotionEvent attribute), 382
 id (kivy.input.motionevent.MotionEvent attribute), 399
 id (kivy.uix.widget.Widget attribute), 687
 identity() (kivy.graphics.transformation.Matrix method), 367

idle() (kivy.base.EventLoopBase method), 223
 idle() (kivy.clock.ClockBaseBehavior method), 200
 idle() (kivy.clock.ClockBaseFreeInterruptOnly method), 201
 idle() (kivy.clock.ClockBaseInterruptBehavior method), 200
 ids (kivy.uix.widget.Widget attribute), 687
 ignore_perpendicular_swipes (kivy.uix.carousel.Carousel attribute), 524
 ignored_touch (kivy.uix.behaviors.focus.FocusBehavior attribute), 478
 ignored_touch (kivy.uix.behaviors.FocusBehavior attribute), 458
 Image (class in kivy.core.image), 263
 Image (class in kivy.uix.image), 560
 image (kivy.core.image.Image attribute), 264
 image() (kivy.loader.LoaderBase method), 185
 image_loading (kivy.uix.videoplayer.VideoPlayer attribute), 672
 image_overlay_play (kivy.uix.videoplayer.VideoPlayer attribute), 672
 image_pause (kivy.uix.videoplayer.VideoPlayer attribute), 672
 image_play (kivy.uix.videoplayer.VideoPlayer attribute), 672
 image_ratio (kivy.uix.image.Image attribute), 560
 image_stop (kivy.uix.videoplayer.VideoPlayer attribute), 672
 image_volumehigh (kivy.uix.videoplayer.VideoPlayer attribute), 673
 image_volumelow (kivy.uix.videoplayer.VideoPlayer attribute), 673
 image_volumemedium (kivy.uix.videoplayer.VideoPlayer attribute), 673
 image_volumemuted (kivy.uix.videoplayer.VideoPlayer attribute), 673
 ImageData (class in kivy.core.image), 266
 import_gesture() (kivy.multistroke.Recognizer method), 231
 important (kivy.uix.actionbar.ActionItem attribute), 505
 in_back() (kivy.animation.AnimationTransition static method), 163
 in_bbox() (kivy.vector.Vector static method), 256
 in_bounce() (kivy.animation.AnimationTransition static method), 163
 in_circ() (kivy.animation.AnimationTransition static method), 163
 in_cubic() (kivy.animation.AnimationTransition static method), 164
 in_elastic() (kivy.animation.AnimationTransition static method), 164
 in_expo() (kivy.animation.AnimationTransition static method), 164
 in_out_back() (kivy.animation.AnimationTransition static method), 164
 in_out_bounce() (kivy.animation.AnimationTransition static method), 165
 in_out_circ() (kivy.animation.AnimationTransition static method), 165
 in_out_cubic() (kivy.animation.AnimationTransition static method), 165
 in_out_elastic() (kivy.animation.AnimationTransition static method), 165
 in_out_expo() (kivy.animation.AnimationTransition static method), 166
 in_out_quad() (kivy.animation.AnimationTransition static method), 166
 in_out_quart() (kivy.animation.AnimationTransition static method), 166
 in_out_quint() (kivy.animation.AnimationTransition static method), 166
 in_out_sine() (kivy.animation.AnimationTransition static method), 167
 in_quad() (kivy.animation.AnimationTransition static method), 167
 in_quart() (kivy.animation.AnimationTransition static method), 167
 in_quint() (kivy.animation.AnimationTransition static method), 167
 in_sine() (kivy.animation.AnimationTransition static method), 168
 inch() (in module kivy.metrics), 227
 indent_level (kivy.uix.treeview.TreeView attribute), 664
 indent_start (kivy.uix.treeview.TreeView attribute), 665
 index (kivy.core.camera.CameraBase attribute), 262
 index (kivy.uix.camera.Camera attribute), 522
 index (kivy.uix.carousel.Carousel attribute), 524
 index (kivy.uix.filechooser.FileChooserProgressBase attribute), 549
 indexof() (kivy.graphics.InstructionGroup method), 314
 indexof() (kivy.graphics.instructions.InstructionGroup method), 327
 indices (kivy.graphics.Mesh attribute), 320
 indices (kivy.graphics.vertex_instructions.Mesh attribute), 378
 init() (kivy.modules.console.ConsoleAddon method), 434
 init_async_lib() (kivy.clock.ClockBaseBehavior method), 200
 init_async_lib() (kivy.clock.ClockBaseInterruptBehavior method), 200
 init_camera() (kivy.core.camera.CameraBase

method), 262

init_gesture() (kivy.uix.gesturesurface.GestureSurface method), 553

input_absinfo (class in kivy.lib.mtdev), 427

input_event (class in kivy.lib.mtdev), 427

input_filter (kivy.uix.textinput.TextInput attribute), 657

input_type (kivy.uix.behaviors.focus.FocusBehavior attribute), 479

input_type (kivy.uix.behaviors.FocusBehavior attribute), 458

InputPostprocCalibration (class in kivy.input.postproc.calibration), 386

InputPostprocDejitter (class in kivy.input.postproc.dejitter), 386

InputPostprocDoubleTap (class in kivy.input.postproc.doubletap), 387

InputPostprocIgnoreList (class in kivy.input.postproc.ignorelist), 387

InputPostprocRetainTouch (class in kivy.input.postproc.retaintouch), 387

InputPostprocTripleTap (class in kivy.input.postproc.tripletap), 388

insert() (kivy.graphics.InstructionGroup method), 314

insert() (kivy.graphics.instructions.InstructionGroup method), 327

insert_text() (kivy.uix.textinput.TextInput method), 657

inside_group (kivy.uix.actionbar.ActionItem attribute), 505

inspect_enabled (kivy.modules.console.Console attribute), 433

install_android() (in module kivy.support), 252

install_gobject_iteration() (in module kivy.support), 251

install_twisted_reactor() (in module kivy.support), 251

Instruction (class in kivy.graphics), 313

Instruction (class in kivy.graphics.instructions), 327

InstructionGroup (class in kivy.graphics), 313

InstructionGroup (class in kivy.graphics.instructions), 327

InteractiveLauncher (class in kivy.interactive), 222

interface (kivy.uix.settings.Settings attribute), 621

interface_cls (kivy.uix.settings.Settings attribute), 622

InterfaceWithSidebar (class in kivy.uix.settings), 626

interpolate() (in module kivy.utils), 253

intersect() (kivy.graphics.scissor_instructions.Rect method), 352

intersection() (in module kivy.utils), 252

invalidate() (kivy.uix.recycleview.views.RecycleDataAdapter method), 498

inverse() (kivy.graphics.transformation.Matrix method), 367

InvertEffect (class in kivy.uix.effectwidget), 539

is_active (kivy.uix.screenmanager.TransitionBase attribute), 609

is_color_transparent() (in module kivy.utils), 252

is_dead() (kivy.weakmethod.WeakMethod method), 258

is_dir() (kivy.uix.filechooser.FileSystemAbstract method), 549

is_dir() (kivy.uix.filechooser.FileSystemLocal method), 550

is_double_tap (kivy.input.MotionEvent attribute), 383

is_double_tap (kivy.input.motionEvent.MotionEvent attribute), 399

is_event_type() (kivy.event.EventDispatcher method), 214

is_finished (kivy.network.urlrequest.UrlRequest attribute), 443

is_focusable (kivy.uix.behaviors.focus.FocusBehavior attribute), 479

is_focusable (kivy.uix.behaviors.FocusBehavior attribute), 458

is_hidden() (kivy.uix.filechooser.FileSystemAbstract method), 549

is_hidden() (kivy.uix.filechooser.FileSystemLocal method), 550

is_last_line (kivy.core.text.text_layout.LayoutLine attribute), 274

is_leaf (kivy.uix.treeview.TreeViewNode attribute), 667

is_loaded (kivy.uix.treeview.TreeViewNode attribute), 667

is_manual (kivy.effects.kinetic.KineticEffect attribute), 296

is_mouse_scrolling (kivy.input.MotionEvent attribute), 383

is_mouse_scrolling (kivy.input.motionEvent.MotionEvent attribute), 400

is_open (kivy.uix.actionbar.ActionGroup attribute), 507

is_open (kivy.uix.spinner.Spinner attribute), 634

is_open (kivy.uix.treeview.TreeViewNode attribute), 667

is_selected (kivy.uix.treeview.TreeViewNode attribute), 667

is_shortened (kivy.uix.label.Label attribute), 570

is_touch (kivy.input.MotionEvent attribute), 383

is_touch (kivy.input.motionEvent.MotionEvent attribute), 400

is_triggered (kivy.clock.ClockEvent attribute), 196

is_triple_tap (kivy.input.MotionEvent attribute), 383

is_triple_tap (kivy.input.motionevent.MotionEvent attribute), 400
 isclose() (in module kivy.compat), 202
 italic (kivy.uix.label.Label attribute), 570
 iterate_all_nodes() (kivy.uix.treeview.TreeView method), 665
 iterate_mipmaps() (kivy.core.image.ImageData method), 266
 iterate_open_nodes() (kivy.uix.treeview.TreeView method), 665

J
 joint (kivy.graphics.Line attribute), 318
 joint (kivy.graphics.vertex_instructions.Line attribute), 376
 joint_precision (kivy.graphics.Line attribute), 318
 joint_precision (kivy.graphics.vertex_instructions.Line attribute), 376
 JsonStore (class in kivy.storage.jsonstore), 449

K
 keep_data (kivy.uix.image.Image attribute), 560
 keep_ratio (kivy.uix.image.Image attribute), 560
 keep_ref (kivy.uix.behaviors.knspace.KNSpace attribute), 484
 keep_within_parent (kivy.uix.splitter.Splitter attribute), 636
 key (kivy.uix.settings.SettingItem attribute), 623
 key_background_color (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_background_down (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_background_normal (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_bindings (kivy.uix.behaviors.emacs.EmacsBehavior attribute), 476
 key_bindings (kivy.uix.behaviors.EmacsBehavior attribute), 463
 key_border (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_disabled_background_normal (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_margin (kivy.uix.vkeyboard.VKeyboard attribute), 678
 key_selection (kivy.uix.recycleview.layout.LayoutSelectionBehavior attribute), 496
 key_viewclass (kivy.uix.recycleview.layout.RecycleViewLayoutManagerBehavior attribute), 496
 key_viewclass (kivy.uix.recycleview.RecycleView attribute), 492
 Keyboard (class in kivy.core.window), 278
 keyboard (kivy.uix.behaviors.focus.FocusBehavior attribute), 479
 keyboard (kivy.uix.behaviors.FocusBehavior attribute), 458
 keyboard_anim_args (kivy.core.window.WindowBase attribute), 283
 keyboard_height (kivy.core.window.WindowBase attribute), 283
 keyboard_mode (kivy.uix.behaviors.focus.FocusBehavior attribute), 479
 keyboard_mode (kivy.uix.behaviors.FocusBehavior attribute), 459
 keyboard_on_key_down() (kivy.uix.behaviors.focus.FocusBehavior method), 480
 keyboard_on_key_down() (kivy.uix.behaviors.FocusBehavior method), 459
 keyboard_on_key_down() (kivy.uix.textinput.TextInput method), 657
 keyboard_on_key_up() (kivy.uix.behaviors.focus.FocusBehavior method), 480
 keyboard_on_key_up() (kivy.uix.behaviors.FocusBehavior method), 459
 keyboard_on_key_up() (kivy.uix.textinput.TextInput method), 657
 keyboard_padding (kivy.core.window.WindowBase attribute), 283
 keyboard_select (kivy.uix.behaviors.compoundselection.CompoundSelection attribute), 470
 keyboard_select (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461
 keyboard_suggestions (kivy.uix.textinput.TextInput attribute), 657
 keycode_to_string() (kivy.core.window.Keyboard method), 279
 keys() (kivy.storage.AbstractStore method), 448
 KineticEffect (class in kivy.effects.kinetic), 296
 kivy (module), 159
 kivy.animation (module), 160
 kivy.app (module), 171
 kivy.atlas (module), 186
 kivy.base (module), 222
 kivy.cache (module), 189
 kivy.clock (module), 191
 kivy.compat (module), 201
 kivy.config (module), 202
 kivy.context (module), 208
 kivy.core (module), 259

kivy.core.audio (module), 259
 kivy.core.camera (module), 261
 kivy.core.clipboard (module), 262
 kivy.core.gl (module), 262
 kivy.core.image (module), 262
 kivy.core.spelling (module), 267
 kivy.core.text (module), 268
 kivy.core.text.markup (module), 275
 kivy.core.text.text_layout (module), 272
 kivy.core.video (module), 276
 kivy.core.window (module), 278
 kivy.deps (module), 293
 kivy.effects (module), 295
 kivy.effects.dampedscroll (module), 295
 kivy.effects.kinetic (module), 296
 kivy.effects.opacityscroll (module), 297
 kivy.effects.scroll (module), 298
 kivy.event (module), 208
 kivy.factory (module), 216
 kivy.garden (module), 301
 kivy.geometry (module), 217
 kivy.gesture (module), 218
 kivy.graphics (module), 305
 kivy.graphics.cgl (module), 331
 kivy.graphics.compiler (module), 340
 kivy.graphics.context (module), 336
 kivy.graphics.context_instructions (module), 332
 kivy.graphics.fbo (module), 336
 kivy.graphics.gl_instructions (module), 339
 kivy.graphics.instructions (module), 327
 kivy.graphics.opengl (module), 341
 kivy.graphics.opengl_utils (module), 350
 kivy.graphics.scissor_instructions (module), 351
 kivy.graphics.shader (module), 352
 kivy.graphics.stencil_instructions (module), 354
 kivy.graphics.svg (module), 356
 kivy.graphics.tessellator (module), 357
 kivy.graphics.texture (module), 360
 kivy.graphics.transformation (module), 366
 kivy.graphics.vertex_instructions (module), 370
 kivy.input (module), 381
 kivy.input.factory (module), 401
 kivy.input.motionEvent (module), 397
 kivy.input.postproc (module), 385
 kivy.input.postproc.calibration (module), 385
 kivy.input.postproc.dejitter (module), 386
 kivy.input.postproc.doubletap (module), 387
 kivy.input.postproc.ignorelist (module), 387
 kivy.input.postproc.retain_touch (module), 387
 kivy.input.postproc.tripletap (module), 388
 kivy.input.provider (module), 402
 kivy.input.providers (module), 388
 kivy.input.providers.androidjoystick (module), 388
 kivy.input.providers.hidinput (module), 390
 kivy.input.providers.leapfinger (module), 389
 kivy.input.providers.linuxwacom (module), 392
 kivy.input.providers.mactouch (module), 392
 kivy.input.providers.mouse (module), 389
 kivy.input.providers.mtdev (module), 391
 kivy.input.providers.probesysfs (module), 388
 kivy.input.providers.tuio (module), 393
 kivy.input.providers.wm_common (module), 389
 kivy.input.providers.wm_pen (module), 393
 kivy.input.providers.wm_touch (module), 393
 kivy.input.recorder (module), 395
 kivy.input.shape (module), 402
 kivy.interactive (module), 219
 kivy.lang (module), 403
 kivy.lang.builder (module), 419
 kivy.lang.parser (module), 423
 kivy.lib (module), 425
 kivy.lib.ddsfile (module), 425
 kivy.lib.gstplayer (module), 425
 kivy.lib.mtdev (module), 426
 kivy.loader (module), 184
 kivy.logger (module), 225
 kivy.metrics (module), 226
 kivy.modules (module), 429
 kivy.modules.console (module), 430
 kivy.modules.inspector (module), 434
 kivy.modules.joystick (module), 435
 kivy.modules.keybinding (module), 436
 kivy.modules.monitor (module), 437
 kivy.modules.recorder (module), 437
 kivy.modules.screen (module), 438
 kivy.modules.touchring (module), 438
 kivy.modules.webdebugger (module), 439
 kivy.multistroke (module), 228
 kivy.network (module), 441
 kivy.network.urlrequest (module), 441
 kivy.parser (module), 236
 kivy.properties (module), 237
 kivy.resources (module), 251
 kivy.storage (module), 445
 kivy.storage.dictstore (module), 449
 kivy.storage.jsonstore (module), 449
 kivy.storage.redisstore (module), 449
 kivy.support (module), 251
 kivy.tools (module), 451
 kivy.tools.packaging (module), 452
 kivy.uix (module), 453
 kivy.uix.accordion (module), 499
 kivy.uix.actionbar (module), 504
 kivy.uix.anchorlayout (module), 511
 kivy.uix.behaviors (module), 453
 kivy.uix.behaviors.button (module), 465
 kivy.uix.behaviors.codenavigation (module), 467
 kivy.uix.behaviors.compoundselection (module), 467

kivy.uix.behaviors.cover (module), 473
 kivy.uix.behaviors.drag (module), 474
 kivy.uix.behaviors.emacs (module), 475
 kivy.uix.behaviors.focus (module), 476
 kivy.uix.behaviors.knspace (module), 480
 kivy.uix.behaviors.togglebutton (module), 485
 kivy.uix.behaviors.touchripple (module), 487
 kivy.uix.boxlayout (module), 513
 kivy.uix.bubble (module), 515
 kivy.uix.button (module), 520
 kivy.uix.camera (module), 521
 kivy.uix.carousel (module), 522
 kivy.uix.checkbox (module), 526
 kivy.uix.codeinput (module), 528
 kivy.uix.colorpicker (module), 530
 kivy.uix.dropdown (module), 532
 kivy.uix.effectwidget (module), 535
 kivy.uix.filechooser (module), 540
 kivy.uix.floatlayout (module), 550
 kivy.uix.gesturesurface (module), 552
 kivy.uix.gridlayout (module), 555
 kivy.uix.image (module), 559
 kivy.uix.label (module), 562
 kivy.uix.layout (module), 575
 kivy.uix.modalview (module), 577
 kivy.uix.pagelayout (module), 580
 kivy.uix.popup (module), 581
 kivy.uix.progressbar (module), 584
 kivy.uix.recycleboxlayout (module), 585
 kivy.uix.recyclegridlayout (module), 586
 kivy.uix.recyclelayout (module), 586
 kivy.uix.recycleview (module), 489
 kivy.uix.recycleview.datamodel (module), 493
 kivy.uix.recycleview.layout (module), 494
 kivy.uix.recycleview.views (module), 497
 kivy.uix.relativelayout (module), 587
 kivy.uix.rst (module), 592
 kivy.uix.sandbox (module), 595
 kivy.uix.scatter (module), 596
 kivy.uix.scatterlayout (module), 601
 kivy.uix.screenmanager (module), 603
 kivy.uix.scrollview (module), 612
 kivy.uix.settings (module), 618
 kivy.uix.slider (module), 628
 kivy.uix.spinner (module), 632
 kivy.uix.splitter (module), 634
 kivy.uix.stacklayout (module), 637
 kivy.uix.stencilview (module), 639
 kivy.uix.switch (module), 640
 kivy.uix.tabbedpanel (module), 641
 kivy.uix.textinput (module), 648
 kivy.uix.togglebutton (module), 662
 kivy.uix.treeview (module), 662
 kivy.uix.video (module), 668
 kivy.uix.videoplayer (module), 670
 kivy.uix.vkeyboard (module), 675
 kivy.uix.widget (module), 680
 kivy.utils (module), 252
 kivy.vector (module), 254
 kivy.weakmethod (module), 258
 kivy.weakproxy (module), 258
 kivy_base_dir (in module kivy), 160
 kivy_config_fn (in module kivy), 160
 kivy_configure() (in module kivy), 160
 kivy_data_dir (in module kivy), 160
 kivy_home_dir (in module kivy), 160
 kivy_icons_dir (in module kivy), 160
 kivy_options (in module kivy), 160
 kivy_register_post_configuration() (in module kivy), 160
 kivy_shader_dir (in module kivy), 160
 kivy_usermodules_dir (in module kivy), 160
 knsname (kivy.uix.behaviors.knspace.KNSpaceBehavior attribute), 485
 KNSpace (class in kivy.uix.behaviors.knspace), 484
 knspace (in module kivy.uix.behaviors.knspace), 485
 knspace (kivy.uix.behaviors.knspace.KNSpaceBehavior attribute), 485
 knspace_key (kivy.uix.behaviors.knspace.KNSpaceBehavior attribute), 485
 KNSpaceBehavior (class in kivy.uix.behaviors.knspace), 485
 kv_directory (kivy.app.App attribute), 180
 kv_file (kivy.app.App attribute), 180

 L
 Label (class in kivy.uix.label), 566
 label (kivy.core.text.LabelBase attribute), 271
 LabelBase (class in kivy.core.text), 269
 last_touch (kivy.uix.behaviors.button.ButtonBehavior attribute), 466
 last_touch (kivy.uix.behaviors.ButtonBehavior attribute), 455
 last_touch (kivy.uix.behaviors.touchripple.TouchRippleButtonBehavior attribute), 488
 last_touch (kivy.uix.behaviors.TouchRippleButtonBehavior attribute), 465
 Layout (class in kivy.uix.layout), 575
 layout (kivy.uix.filechooser.FileChooserController attribute), 547
 layout (kivy.uix.vkeyboard.VKeyboard attribute), 678
 layout_hint_with_bounds() (kivy.uix.layout.Layout method), 576
 layout_manager (kivy.uix.recycleview.RecycleViewBehavior attribute), 491
 layout_path (kivy.uix.vkeyboard.VKeyboard attribute), 678

layout_text() (in module kivy.core.text.text_layout), 272
 LayoutChangeException, 494
 LayoutLine (class in kivy.core.text.text_layout), 274
 LayoutSelectionBehavior (class in kivy.uix.recycleview.layout), 494
 LayoutWord (class in kivy.core.text.text_layout), 273
 LeapFingerEvent (class in kivy.input.providers.leapfinger), 389
 LeapFingerEventProvider (class in kivy.input.providers.leapfinger), 389
 left (kivy.core.window.WindowBase attribute), 283
 length (kivy.core.audio.Sound attribute), 260
 length (kivy.properties.VariableListProperty attribute), 248
 length() (kivy.graphics.InstructionGroup method), 314
 length() (kivy.graphics.instructions.InstructionGroup method), 327
 length() (kivy.vector.Vector method), 256
 length2() (kivy.vector.Vector method), 256
 level (kivy.uix.treeview.TreeViewNode attribute), 667
 lexer (kivy.uix.codeinput.CodeInput attribute), 529
 lh (kivy.core.text.text_layout.LayoutWord attribute), 274
 limit_to (kivy.uix.bubble.Bubble attribute), 519
 Line (class in kivy.graphics), 314
 Line (class in kivy.graphics.vertex_instructions), 372
 line_height (kivy.uix.label.Label attribute), 570
 line_height (kivy.uix.textinput.TextInput attribute), 658
 line_intersection() (kivy.vector.Vector static method), 256
 line_spacing (kivy.uix.textinput.TextInput attribute), 658
 line_wrap (kivy.core.text.text_layout.LayoutLine attribute), 274
 linear() (kivy.animation.AnimationTransition static method), 168
 link() (kivy.properties.DictProperty method), 248
 link() (kivy.properties.ListProperty method), 244
 link() (kivy.properties.Property method), 242
 link() (kivy.properties.ReferenceListProperty method), 246
 link() (kivy.properties.VariableListProperty method), 248
 link_deps() (kivy.properties.AliasProperty method), 247
 link_deps() (kivy.properties.ConfigParserProperty method), 250
 link_deps() (kivy.properties.Property method), 243
 link_deps() (kivy.properties.ReferenceListProperty method), 246
 LinuxWacomMotionEvent (class in kivy.input.providers.linuxwacom), 393
 list() (kivy.input.factory.MotionEventFactory static method), 402
 list() (kivy.input.MotionEventFactory static method), 385
 list_languages() (kivy.core.spelling.SpellingBase method), 267
 listdir() (kivy.uix.filechooser.FileSystemAbstract method), 549
 listdir() (kivy.uix.filechooser.FileSystemLocal method), 550
 ListProperty (class in kivy.properties), 243
 load() (kivy.core.audio.Sound method), 260
 load() (kivy.core.audio.SoundLoader static method), 261
 load() (kivy.core.image.Image static method), 264
 load() (kivy.core.video.VideoBase method), 277
 load_config() (kivy.app.App method), 181
 load_file() (kivy.lang.builder.BuilderBase method), 421
 load_file() (kivy.lang.BuilderBase method), 417
 load_func (kivy.uix.treeview.TreeView attribute), 665
 load_kv() (kivy.app.App method), 181
 load_memory() (kivy.core.image.Image method), 264
 load_next() (kivy.uix.carousel.Carousel method), 524
 load_previous() (kivy.uix.carousel.Carousel method), 524
 load_slide() (kivy.uix.carousel.Carousel method), 525
 load_string() (kivy.lang.builder.BuilderBase method), 421
 load_string() (kivy.lang.BuilderBase method), 417
 loaded (kivy.uix.video.Video attribute), 668
 LoaderBase (class in kivy.loader), 184
 LoadIdentity (class in kivy.graphics), 326
 loading_image (kivy.loader.LoaderBase attribute), 185
 Logger (in module kivy.logger), 225
 LoggerHistory (class in kivy.logger), 225
 look_at() (kivy.graphics.transformation.Matrix method), 367
 loop (kivy.clock.ClockEvent attribute), 197
 loop (kivy.core.audio.Sound attribute), 260
 loop (kivy.uix.carousel.Carousel attribute), 525
 lw (kivy.core.text.text_layout.LayoutWord attribute), 274

M

- MacMotionEventProvider (class in kivy.input.providers.mactouch), 392
- mag_filter (kivy.graphics.texture.Texture attribute), 365
- mainloop() (kivy.core.window.WindowBase method), 283
- mainthread() (in module kivy.clock), 201
- make_view_dirty() (kivy.uix.recycleview.views.RecycleViewAdapter method), 498
- make_views_dirty() (kivy.uix.recycleview.views.RecycleViewAdapter method), 498
- manager (kivy.uix.filechooser.FileChooser attribute), 546
- manager (kivy.uix.screenmanager.Screen attribute), 606
- manager (kivy.uix.screenmanager.TransitionBase attribute), 609
- margin_hint (kivy.uix.vkeyboard.VKeyboard attribute), 678
- markup (kivy.core.text.markup.MarkupLabel attribute), 276
- markup (kivy.uix.actionbar.ActionPrevious attribute), 509
- markup (kivy.uix.label.Label attribute), 570
- MarkupLabel (class in kivy.core.text.markup), 276
- match() (kivy.lang.builder.BuilderBase method), 421
- match() (kivy.lang.BuilderBase method), 417
- match_candidate() (kivy.multistroke.MultistrokeGesture method), 234
- match_rule_name() (kivy.lang.builder.BuilderBase method), 421
- match_rule_name() (kivy.lang.BuilderBase method), 417
- Matrix (class in kivy.graphics.transformation), 367
- matrix (kivy.graphics.context_instructions.MatrixInstruction attribute), 335
- matrix (kivy.graphics.MatrixInstruction attribute), 319
- MatrixInstruction (class in kivy.graphics), 319
- MatrixInstruction (class in kivy.graphics.context_instructions), 335
- max (kivy.effects.scroll.ScrollEffect attribute), 298
- max (kivy.uix.progressbar.ProgressBar attribute), 585
- max (kivy.uix.slider.Slider attribute), 630
- max_height (kivy.uix.dropdown.DropDown attribute), 534
- max_history (kivy.effects.kinetic.KineticEffect attribute), 297
- max_iteration (kivy.clock.CyClockBase attribute), 197
- max_lines (kivy.uix.label.Label attribute), 570
- max_size (kivy.uix.splitter.Splitter attribute), 636
- max_upload_per_frame (kivy.loader.LoaderBase attribute), 185
- maximize() (kivy.core.window.WindowBase method), 284
- menu (kivy.uix.settings.InterfaceWithSidebar attribute), 626
- MenuSidebar (class in kivy.uix.settings), 628
- merge_gestures() (kivy.uix.gesturesurface.GestureSurface method), 553
- Mesh (class in kivy.graphics), 319
- Mesh (class in kivy.graphics.vertex_instructions), 377
- meshes (kivy.graphics.tesselator.Tesselator attribute), 359
- Metrics (in module kivy.metrics), 227
- MetricsBase (class in kivy.metrics), 227
- min (kivy.effects.scroll.ScrollEffect attribute), 298
- min (kivy.uix.slider.Slider attribute), 630
- min_distance (kivy.effects.kinetic.KineticEffect attribute), 297
- min_filter (kivy.graphics.texture.Texture attribute), 365
- min_move (kivy.uix.carousel.Carousel attribute), 525
- min_overscroll (kivy.effects.dampedscroll.DampedScrollEffect attribute), 295
- min_size (kivy.uix.splitter.Splitter attribute), 636
- MIN_SLEEP (kivy.clock.ClockBaseBehavior attribute), 199
- min_space (kivy.uix.accordion.Accordion attribute), 501
- min_space (kivy.uix.accordion.AccordionItem attribute), 503
- min_state_time (kivy.uix.behaviors.button.ButtonBehavior attribute), 466
- min_state_time (kivy.uix.behaviors.ButtonBehavior attribute), 455
- min_state_time (kivy.uix.dropdown.DropDown attribute), 534
- min_velocity (kivy.effects.kinetic.KineticEffect attribute), 297
- minimize() (kivy.core.window.WindowBase method), 284
- minimum_bounding_circle() (in module kivy.geometry), 217
- minimum_height (kivy.core.window.WindowBase attribute), 284
- minimum_height (kivy.uix.boxlayout.BoxLayout attribute), 515
- minimum_height (kivy.uix.gridlayout.GridLayout attribute), 558
- minimum_height (kivy.uix.stacklayout.StackLayout attribute), 638
- minimum_height (kivy.uix.textinput.TextInput attribute), 638

tribute), 658
 minimum_height (kivy.uix.treeview.TreeView attribute), 665
 minimum_size (kivy.uix.boxlayout.BoxLayout attribute), 515
 minimum_size (kivy.uix.gridlayout.GridLayout attribute), 558
 minimum_size (kivy.uix.stacklayout.StackLayout attribute), 638
 minimum_size (kivy.uix.treeview.TreeView attribute), 665
 minimum_width (kivy.core.window.WindowBase attribute), 284
 minimum_width (kivy.uix.actionbar.ActionItem attribute), 505
 minimum_width (kivy.uix.boxlayout.BoxLayout attribute), 515
 minimum_width (kivy.uix.gridlayout.GridLayout attribute), 558
 minimum_width (kivy.uix.stacklayout.StackLayout attribute), 638
 minimum_width (kivy.uix.treeview.TreeView attribute), 665
 mipmap (kivy.graphics.texture.Texture attribute), 365
 mipmap (kivy.uix.actionbar.ActionItem attribute), 505
 mipmap (kivy.uix.image.Image attribute), 560
 mipmap (kivy.uix.label.Label attribute), 570
 mipmaps (kivy.core.image.ImageData attribute), 266
 mm() (in module kivy.metrics), 227
 ModalView (class in kivy.uix.modalview), 578
 mode (kivy.graphics.Mesh attribute), 320
 mode (kivy.graphics.vertex_instructions.Mesh attribute), 378
 mode (kivy.modules.console.Console attribute), 433
 mode (kivy.uix.actionbar.ActionGroup attribute), 507
 mode (kivy.uix.screenmanager.CardTransition attribute), 612
 modifiers (kivy.core.window.WindowBase attribute), 284
 MonochromeEffect (class in kivy.uix.effectwidget), 539
 MotionEvent (class in kivy.input), 381
 MotionEvent (class in kivy.input.motionevent), 398
 MotionEventFactory (class in kivy.input), 385
 MotionEventFactory (class in kivy.input.factory), 401
 MotionEventProvider (class in kivy.input), 384
 MotionEventProvider (class in kivy.input.provider), 402
 mouse_pos (kivy.core.window.WindowBase attribute), 284
 MotionEventProvider (class in kivy.input.providers.mouse), 390
 move() (kivy.input.MotionEvent method), 383
 move() (kivy.input.motionevent.MotionEvent method), 400
 mtdev (class in kivy.lib.mtdev), 427
 mtdev_caps (class in kivy.lib.mtdev), 427
 MTDMotionEvent (class in kivy.input.providers.mtdev), 392
 multiline (kivy.uix.textinput.TextInput attribute), 658
 multiply() (kivy.graphics.transformation.Matrix method), 367
 multiselect (kivy.uix.behaviors.compoundselection.CompoundSelection attribute), 470
 multiselect (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461
 multiselect (kivy.uix.filechooser.FileChooserController attribute), 547
 MultistrokeGesture (class in kivy.multistroke), 233
 N
 name (kivy.app.App attribute), 181
 name (kivy.config.ConfigParser attribute), 207
 name (kivy.uix.screenmanager.Screen attribute), 606
 next (kivy.clock.ClockEvent attribute), 197
 next() (kivy.uix.screenmanager.ScreenManager method), 607
 next_slide (kivy.uix.carousel.Carousel attribute), 525
 no_selection (kivy.uix.treeview.TreeViewNode attribute), 667
 nocache (kivy.core.image.Image attribute), 265
 nocache (kivy.uix.image.Image attribute), 561
 nodes (kivy.uix.treeview.TreeViewNode attribute), 667
 nodes_order_reversed (kivy.uix.behaviors.compoundselection.CompoundSelection attribute), 471
 nodes_order_reversed (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461
 NoLanguageSelectedError, 268
 norm_image_size (kivy.uix.image.Image attribute), 561
 normal_matrix() (kivy.graphics.transformation.Matrix method), 368
 normalize() (kivy.gesture.Gesture method), 219
 normalize() (kivy.vector.Vector method), 257
 normalize_stroke() (kivy.gesture.GestureStroke method), 219
 NoSuchLangError, 267

NoTransition (class in kivy.uix.screenmanager), 611

num_workers (kivy.loader.LoaderBase attribute), 185

NumericProperty (class in kivy.properties), 243

O

ObjectProperty (class in kivy.properties), 244

ObjectWithUid (class in kivy.event), 216

Observable (class in kivy.event), 216

Observable (class in kivy.lang), 415

Observable (class in kivy.lang.builder), 419

observable_dict (kivy.uix.recycleview.datamodel.RecycleDataModel attribute), 494

odd (kivy.uix.treeview.TreeViewNode attribute), 668

odd_color (kivy.uix.treeview.TreeViewNode attribute), 668

on_bring_to_front() (kivy.uix.scatter.Scatter method), 599

on_close() (kivy.core.window.WindowBase method), 284

on_config_change() (kivy.app.App method), 181

on_context_created() (kivy.uix.sandbox.Sandbox method), 595

on_current_uid() (kivy.uix.settings.ContentPanel method), 627

on_cursor_enter() (kivy.core.window.WindowBase method), 284

on_cursor_leave() (kivy.core.window.WindowBase method), 284

on_double_tap() (kivy.uix.textinput.TextInput method), 658

on_dropfile() (kivy.core.window.WindowBase method), 285

on_exception() (kivy.uix.sandbox.Sandbox method), 595

on_flip() (kivy.core.window.WindowBase method), 285

on_hide() (kivy.core.window.WindowBase method), 285

on_joy_axis() (kivy.core.window.WindowBase method), 285

on_joy_ball() (kivy.core.window.WindowBase method), 285

on_joy_button_down() (kivy.core.window.WindowBase method), 285

on_joy_button_up() (kivy.core.window.WindowBase method), 285

on_joy_hat() (kivy.core.window.WindowBase method), 285

on_key_down() (kivy.core.window.WindowBase method), 285

on_key_up() (kivy.core.window.WindowBase method), 285

on_keyboard() (kivy.core.window.WindowBase method), 285

on_maximize() (kivy.core.window.WindowBase method), 285

on_memorywarning() (kivy.core.window.WindowBase method), 286

on_minimize() (kivy.core.window.WindowBase method), 286

on_motion() (kivy.core.window.WindowBase method), 286

on_mouse_down() (kivy.core.window.WindowBase method), 286

on_mouse_move() (kivy.core.window.WindowBase method), 286

on_mouse_up() (kivy.core.window.WindowBase method), 286

on_pause() (kivy.app.App method), 181

on_pause() (kivy.base.EventLoopBase method), 223

on_quad_touch() (kivy.uix.textinput.TextInput method), 658

on_request_close() (kivy.core.window.WindowBase method), 286

on_resize() (kivy.core.window.WindowBase method), 286

on_restore() (kivy.core.window.WindowBase method), 286

on_resume() (kivy.app.App method), 182

on_rotate() (kivy.core.window.WindowBase method), 286

on_schedule() (kivy.clock.CyClockBase method), 198

on_selected_uid() (kivy.uix.settings.MenuSidebar method), 628

on_show() (kivy.core.window.WindowBase method), 287

on_start() (kivy.app.App method), 182

on_start() (kivy.base.EventLoopBase method), 223

on_stop() (kivy.app.App method), 182

on_stop() (kivy.base.EventLoopBase method), 223

on_textedit() (kivy.core.window.WindowBase method), 287

on_textinput() (kivy.core.window.WindowBase method), 287

on_texture() (kivy.core.image.Image method), 265

on_touch_down() (kivy.core.window.WindowBase method), 287

on_touch_down() (kivy.modules.console.Console method), 433

on_touch_down() (kivy.uix.accordion.AccordionItem method), 503

on_touch_down() (kivy.uix.carousel.Carousel method), 525

`on_touch_down()` (kivy.uix.colorpicker.ColorWheel method), 531
`on_touch_down()` (kivy.uix.dropdown.DropDown method), 535
`on_touch_down()` (kivy.uix.filechooser.FileChooserController method), 553
`on_touch_down()` (kivy.uix.filechooser.FileChooserProgressBase method), 549
`on_touch_down()` (kivy.uix.gesturesurface.GestureSurface method), 553
`on_touch_down()` (kivy.uix.label.Label method), 571
`on_touch_down()` (kivy.uix.modalview.ModalView method), 579
`on_touch_down()` (kivy.uix.pagelayout.PageLayout method), 580
`on_touch_down()` (kivy.uix.popup.Popup method), 583
`on_touch_down()` (kivy.uix.relativelayout.RelativeLayout method), 591
`on_touch_down()` (kivy.uix.sandbox.Sandbox method), 596
`on_touch_down()` (kivy.uix.scatter.Scatter method), 599
`on_touch_down()` (kivy.uix.screenmanager.ScreenManager method), 607
`on_touch_down()` (kivy.uix.scrollview.ScrollView method), 616
`on_touch_down()` (kivy.uix.settings.SettingItem method), 623
`on_touch_down()` (kivy.uix.settings.Settings method), 622
`on_touch_down()` (kivy.uix.slider.Slider method), 630
`on_touch_down()` (kivy.uix.switch.Switch method), 641
`on_touch_down()` (kivy.uix.tabbedpanel.TabbedPanelHeader method), 647
`on_touch_down()` (kivy.uix.textinput.TextInput method), 658
`on_touch_down()` (kivy.uix.treeview.TreeView method), 665
`on_touch_down()` (kivy.uix.videooplayer.VideoPlayer method), 673
`on_touch_down()` (kivy.uix.vkeyboard.VKeyboard method), 679
`on_touch_down()` (kivy.uix.widget.Widget method), 688
`on_touch_move()` (kivy.core.window.WindowBase method), 287
`on_touch_move()` (kivy.modules.console.Console method), 433
`on_touch_move()` (kivy.uix.carousel.Carousel method), 525
`on_touch_move()` (kivy.uix.colorpicker.ColorWheel method), 531
`on_touch_move()` (kivy.uix.filechooser.FileChooserProgressBase method), 549
`on_touch_move()` (kivy.uix.gesturesurface.GestureSurface method), 554
`on_touch_move()` (kivy.uix.modalview.ModalView method), 579
`on_touch_move()` (kivy.uix.pagelayout.PageLayout method), 581
`on_touch_move()` (kivy.uix.relativelayout.RelativeLayout method), 591
`on_touch_move()` (kivy.uix.sandbox.Sandbox method), 596
`on_touch_move()` (kivy.uix.scatter.Scatter method), 599
`on_touch_move()` (kivy.uix.screenmanager.ScreenManager method), 607
`on_touch_move()` (kivy.uix.scrollview.ScrollView method), 616
`on_touch_move()` (kivy.uix.slider.Slider method), 630
`on_touch_move()` (kivy.uix.switch.Switch method), 641
`on_touch_move()` (kivy.uix.textinput.TextInput method), 658
`on_touch_move()` (kivy.uix.widget.Widget method), 688
`on_touch_up()` (kivy.core.window.WindowBase method), 287
`on_touch_up()` (kivy.modules.console.Console method), 434
`on_touch_up()` (kivy.uix.carousel.Carousel method), 525
`on_touch_up()` (kivy.uix.colorpicker.ColorWheel method), 531
`on_touch_up()` (kivy.uix.dropdown.DropDown method), 535
`on_touch_up()` (kivy.uix.filechooser.FileChooserController method), 548
`on_touch_up()` (kivy.uix.filechooser.FileChooserProgressBase method), 549
`on_touch_up()` (kivy.uix.gesturesurface.GestureSurface method), 554
`on_touch_up()` (kivy.uix.modalview.ModalView method), 579
`on_touch_up()` (kivy.uix.pagelayout.PageLayout method), 581
`on_touch_up()` (kivy.uix.relativelayout.RelativeLayout method), 591
`on_touch_up()` (kivy.uix.sandbox.Sandbox method), 596
`on_touch_up()` (kivy.uix.scatter.Scatter method), 599
`on_touch_up()` (kivy.uix.screenmanager.ScreenManager method), 608

on_touch_up() (kivy.uix.scrollview.ScrollView method), 616
 on_touch_up() (kivy.uix.settings.SettingItem method), 623
 on_touch_up() (kivy.uix.slider.Slider method), 630
 on_touch_up() (kivy.uix.switch.Switch method), 641
 on_touch_up() (kivy.uix.textinput.TextInput method), 659
 on_touch_up() (kivy.uix.vkeyboard.VKeyboard method), 679
 on_touch_up() (kivy.uix.widget.Widget method), 688
 on_transform_with_touch() (kivy.uix.scatter.Scatter method), 599
 on_triple_tap() (kivy.uix.textinput.TextInput method), 659
 opacity (kivy.graphics.Canvas attribute), 309
 opacity (kivy.graphics.instructions.Canvas attribute), 329
 opacity (kivy.uix.widget.Widget attribute), 688
 OpacityScrollEffect (class in kivy.effects.opacityscroll), 297
 open() (kivy.uix.dropdown.DropDown method), 535
 open() (kivy.uix.modalview.ModalView method), 579
 open_settings() (kivy.app.App method), 182
 opos (kivy.input.MotionEvent attribute), 383
 opos (kivy.input.motionEvent.MotionEvent attribute), 400
 option_cls (kivy.uix.spinner.Spinner attribute), 634
 OptionProperty (class in kivy.properties), 246
 options (kivy.app.App attribute), 182
 options (kivy.core.text.text_layout.LayoutWord attribute), 274
 options (kivy.properties.OptionProperty attribute), 246
 options (kivy.uix.settings.SettingOptions attribute), 625
 options (kivy.uix.video.Video attribute), 669
 options (kivy.uix.videoplayer.VideoPlayer attribute), 673
 order (kivy.uix.effectwidget.ChannelMixEffect attribute), 539
 orientation (kivy.uix.accordion.Accordion attribute), 501
 orientation (kivy.uix.accordion.AccordionItem attribute), 503
 orientation (kivy.uix.boxlayout.BoxLayout attribute), 515
 orientation (kivy.uix.bubble.Bubble attribute), 519
 orientation (kivy.uix.slider.Slider attribute), 630
 orientation (kivy.uix.stacklayout.StackLayout attribute), 638
 origin (kivy.graphics.context_instructions.Rotate attribute), 334
 origin (kivy.graphics.context_instructions.Scale attribute), 334
 origin (kivy.graphics.Rotate attribute), 322
 origin (kivy.graphics.Scale attribute), 323
 original_textures (kivy.atlas.Atlas attribute), 189
 osx (kivy.input.MotionEvent attribute), 383
 osx (kivy.input.motionEvent.MotionEvent attribute), 400
 osy (kivy.input.MotionEvent attribute), 383
 osy (kivy.input.motionEvent.MotionEvent attribute), 400
 osz (kivy.input.MotionEvent attribute), 383
 osz (kivy.input.motionEvent.MotionEvent attribute), 400
 out_back() (kivy.animation.AnimationTransition static method), 168
 out_bounce() (kivy.animation.AnimationTransition static method), 168
 out_circ() (kivy.animation.AnimationTransition static method), 169
 out_cubic() (kivy.animation.AnimationTransition static method), 169
 out_elastic() (kivy.animation.AnimationTransition static method), 169
 out_expo() (kivy.animation.AnimationTransition static method), 169
 out_quad() (kivy.animation.AnimationTransition static method), 170
 out_quart() (kivy.animation.AnimationTransition static method), 170
 out_quint() (kivy.animation.AnimationTransition static method), 170
 out_sine() (kivy.animation.AnimationTransition static method), 170
 outline_color (kivy.uix.label.Label attribute), 571
 outline_width (kivy.uix.label.Label attribute), 571
 overdraw_width (kivy.graphics.SmoothLine attribute), 319
 overdraw_width (kivy.graphics.vertex_instructions.SmoothLine attribute), 379
 overflow_group (kivy.uix.actionbar.ActionView attribute), 509
 overflow_image (kivy.uix.actionbar.ActionOverflow attribute), 508
 overlay_color (kivy.uix.modalview.ModalView attribute), 579
 overscroll (kivy.effects.scroll.ScrollEffect attribute), 298
 ox (kivy.input.MotionEvent attribute), 383
 ox (kivy.input.motionEvent.MotionEvent attribute), 400
 oy (kivy.input.MotionEvent attribute), 383
 oy (kivy.input.motionEvent.MotionEvent attribute), 400

tribute), 400

oz (kivy.input.MotionEvent attribute), 383

oz (kivy.input.motionevent.MotionEvent attribute), 400

P

pack_width (kivy.uix.actionbar.ActionItem attribute), 505

padding (kivy.uix.anchorlayout.AnchorLayout attribute), 512

padding (kivy.uix.boxlayout.BoxLayout attribute), 515

padding (kivy.uix.gridlayout.GridLayout attribute), 558

padding (kivy.uix.label.Label attribute), 571

padding (kivy.uix.slider.Slider attribute), 631

padding (kivy.uix.stacklayout.StackLayout attribute), 639

padding (kivy.uix.textinput.TextInput attribute), 659

padding_x (kivy.uix.label.Label attribute), 571

padding_x (kivy.uix.textinput.TextInput attribute), 659

padding_y (kivy.uix.label.Label attribute), 571

padding_y (kivy.uix.textinput.TextInput attribute), 659

page (kivy.uix.pagelayout.PageLayout attribute), 581

page_count (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 471

page_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461

PageLayout (class in kivy.uix.pagelayout), 580

panel (kivy.uix.settings.SettingItem attribute), 623

panels (kivy.uix.settings.ContentPanel attribute), 627

parent (kivy.core.window.WindowBase attribute), 287

parent (kivy.uix.behaviors.knspace.KNSpace attribute), 485

parent (kivy.uix.widget.Widget attribute), 689

parent_node (kivy.uix.treeview.TreeViewNode attribute), 668

parse() (kivy.lang.Parser method), 419

parse() (kivy.lang.parser.Parser method), 423

parse_bool() (in module kivy.parser), 237

parse_color() (in module kivy.parser), 236

parse_filename() (in module kivy.parser), 237

parse_float (in module kivy.parser), 237

parse_float4() (in module kivy.parser), 237

parse_gesture() (kivy.multistroke.Recognizer method), 231

parse_int (in module kivy.parser), 237

parse_int2() (in module kivy.parser), 237

parse_kivy_version() (in module kivy), 160

parse_level() (kivy.lang.Parser method), 419

parse_level() (kivy.lang.parser.Parser method), 423

parse_string() (in module kivy.parser), 237

Parser (class in kivy.lang), 419

Parser (class in kivy.lang.parser), 423

ParserException, 419, 423

password (kivy.uix.textinput.TextInput attribute), 659

password_mask (kivy.uix.textinput.TextInput attribute), 659

paste() (kivy.uix.textinput.TextInput method), 659

path (kivy.uix.filechooser.FileChooserController attribute), 548

path (kivy.uix.filechooser.FileChooserProgressBase attribute), 549

pause() (kivy.core.video.VideoBase method), 277

pause() (kivy.loader.LoaderBase method), 186

permute() (kivy.multistroke.MultistrokeGesture method), 235

perspective() (kivy.graphics.transformation.Matrix method), 368

pick() (kivy.modules.console.Console method), 434

pitch (kivy.core.audio.Sound attribute), 260

pixel_size (kivy.uix.effectwidget.PixelateEffect attribute), 539

PixelateEffect (class in kivy.uix.effectwidget), 539

pixels (kivy.graphics.Fbo attribute), 313

pixels (kivy.graphics.texture.Texture attribute), 338

pixels (kivy.graphics.texture.Texture attribute), 365

pixels (kivy.graphics.texture.TextureRegion attribute), 366

platform (in module kivy.utils), 253

play (kivy.input.recorder.Recorder attribute), 397

play (kivy.uix.camera.Camera attribute), 522

play (kivy.uix.video.Video attribute), 669

play (kivy.uix.videoplayer.VideoPlayer attribute), 673

play() (kivy.core.audio.Sound method), 260

play() (kivy.core.video.VideoBase method), 277

Point (class in kivy.graphics), 320

Point (class in kivy.graphics.vertex_instructions), 377

points (kivy.graphics.Bezier attribute), 306

points (kivy.graphics.Line attribute), 318

points (kivy.graphics.Point attribute), 321

points (kivy.graphics.Quad attribute), 321

points (kivy.graphics.Triangle attribute), 324

points (kivy.graphics.vertex_instructions.Bezier attribute), 379

points (kivy.graphics.vertex_instructions.Line attribute), 376

points (kivy.graphics.vertex_instructions.Point attribute), 377

points (kivy.graphics.vertex_instructions.Quad attribute), 370
 points (kivy.graphics.vertex_instructions.Triangle attribute), 370
 points_distance() (kivy.gesture.GestureStroke method), 219
 pointsize (kivy.graphics.Point attribute), 321
 pointsize (kivy.graphics.vertex_instructions.Point attribute), 377
 pop() (kivy.context.Context method), 208
 pop() (kivy.graphics.scissor_instructions.ScissorStack method), 352
 pop() (kivy.input.MotionEvent method), 383
 pop() (kivy.input.motionevent.MotionEvent method), 400
 PopMatrix (class in kivy.graphics), 321
 PopMatrix (class in kivy.graphics.context_instructions), 334
 PopState (class in kivy.graphics), 326
 Popup (class in kivy.uix.popup), 583
 popup (kivy.uix.settings.SettingOptions attribute), 625
 popup (kivy.uix.settings.SettingPath attribute), 624
 popup (kivy.uix.settings.SettingString attribute), 624
 PopupException, 584
 pos (kivy.graphics.Rectangle attribute), 321
 pos (kivy.graphics.vertex_instructions.Rectangle attribute), 371
 pos (kivy.input.MotionEvent attribute), 383
 pos (kivy.input.motionevent.MotionEvent attribute), 400
 pos (kivy.uix.widget.Widget attribute), 689
 pos_hint (kivy.uix.widget.Widget attribute), 689
 position (kivy.core.video.VideoBase attribute), 277
 position (kivy.uix.video.Video attribute), 669
 position (kivy.uix.videoplayer.VideoPlayer attribute), 673
 post_dispatch_input() (kivy.base.EventLoopBase method), 223
 post_idle() (kivy.clock.ClockBaseBehavior method), 200
 ppos (kivy.input.MotionEvent attribute), 383
 ppos (kivy.input.motionevent.MotionEvent attribute), 400
 pre_idle() (kivy.clock.ClockBaseBehavior method), 200
 preload() (kivy.uix.rst.RstDocument method), 594
 premultiplied_texture() (kivy.graphics.SmoothLine method), 319
 premultiplied_texture() (kivy.graphics.vertex_instructions.SmoothLine method), 379
 prepare() (kivy.multistroke.Candidate method), 236
 prepare() (kivy.multistroke.UnistrokeTemplate method), 235
 prepare_templates() (kivy.multistroke.Recognizer method), 231
 prev (kivy.clock.ClockEvent attribute), 197
 previous() (kivy.uix.screenmanager.ScreenManager method), 608
 previous_image (kivy.uix.actionbar.ActionPrevious attribute), 510
 previous_image_height (kivy.uix.actionbar.ActionPrevious attribute), 510
 previous_image_width (kivy.uix.actionbar.ActionPrevious attribute), 510
 previous_slide (kivy.uix.carousel.Carousel attribute), 525
 print_usage() (kivy.cache.Cache static method), 190
 profile (kivy.input.MotionEvent attribute), 383
 profile (kivy.input.motionevent.MotionEvent attribute), 400
 progress (kivy.multistroke.ProgressTracker attribute), 233
 progress_cls (kivy.uix.filechooser.FileChooserController attribute), 548
 ProgressBar (class in kivy.uix.progressbar), 585
 ProgressTracker (class in kivy.multistroke), 232
 project() (kivy.graphics.transformation.Matrix method), 368
 properties() (kivy.event.EventDispatcher method), 214
 Property (class in kivy.properties), 241
 property() (kivy.event.EventDispatcher method), 214
 property() (kivy.uix.behaviors.knspace.KNSpace method), 485
 proxy_ref (kivy.event.EventDispatcher attribute), 214
 proxy_ref (kivy.graphics.Instruction attribute), 313
 proxy_ref (kivy.graphics.instructions.Instruction attribute), 327
 proxy_ref (kivy.uix.widget.Widget attribute), 689
 ProxyImage (class in kivy.loader), 186
 psx (kivy.input.MotionEvent attribute), 383
 psx (kivy.input.motionevent.MotionEvent attribute), 400
 psy (kivy.input.MotionEvent attribute), 383
 psy (kivy.input.motionevent.MotionEvent attribute), 400
 psz (kivy.input.MotionEvent attribute), 383
 psz (kivy.input.motionevent.MotionEvent attribute), 400

tribute), 400

pt() (in module kivy.metrics), 227

push() (kivy.graphics.scissor_instructions.ScissorStack method), 352

push() (kivy.input.MotionEvent method), 383

push() (kivy.input.motionevent.MotionEvent method), 400

push_attrs_stack (kivy.input.MotionEvent attribute), 383

push_attrs_stack (kivy.input.motionevent.MotionEvent attribute), 400

PushMatrix (class in kivy.graphics), 321

PushMatrix (class in kivy.graphics.context_instructions), 334

PushState (class in kivy.graphics), 326

put() (kivy.storage.AbstractStore method), 448

px (kivy.input.MotionEvent attribute), 384

px (kivy.input.motionevent.MotionEvent attribute), 400

py (kivy.input.MotionEvent attribute), 384

py (kivy.input.motionevent.MotionEvent attribute), 401

PY2 (in module kivy.compat), 201

pz (kivy.input.MotionEvent attribute), 384

pz (kivy.input.motionevent.MotionEvent attribute), 401

Q

Quad (class in kivy.graphics), 321

Quad (class in kivy.graphics.vertex_instructions), 370

QueryDict (class in kivy.lib.ddsfile), 426

QueryDict (class in kivy.utils), 253

R

r (kivy.graphics.ClearColor attribute), 325

r (kivy.graphics.Color attribute), 311

r (kivy.graphics.context_instructions.Color attribute), 333

r (kivy.graphics.gl_instructions.ClearColor attribute), 339

r (kivy.uix.colorpicker.ColorWheel attribute), 531

radius (kivy.graphics.vertex_instructions.RoundedRectangle attribute), 371

raise_window() (kivy.core.window.WindowBase method), 287

range (kivy.uix.slider.Slider attribute), 631

read() (kivy.config.ConfigParser method), 207

read_pixel() (kivy.core.image.Image method), 265

readonly (kivy.uix.textinput.TextInput attribute), 659

rebind (kivy.properties.AliasProperty attribute), 247

rebind (kivy.properties.DictProperty attribute), 248

rebind (kivy.properties.ObjectProperty attribute), 244

recognize() (kivy.multistroke.Recognizer method), 231

Recognizer (class in kivy.multistroke), 229

record (kivy.input.recorder.Recorder attribute), 397

record_attrs (kivy.input.recorder.Recorder attribute), 397

record_profile_mask (kivy.input.recorder.Recorder attribute), 397

Recorder (class in kivy.input.recorder), 396

in Rect (class in kivy.graphics.scissor_instructions), 352

Rectangle (class in kivy.graphics), 321

Rectangle (class in kivy.graphics.vertex_instructions), 371

rectangle (kivy.graphics.Line attribute), 318

rectangle (kivy.graphics.vertex_instructions.Line attribute), 376

RecycleBoxLayout (class in kivy.uix.recycleboxlayout), 585

RecycleDataAdapter (class in kivy.uix.recycleview.views), 497

RecycleDataModel (class in kivy.uix.recycleview.datamodel), 493

RecycleDataModelBehavior (class in kivy.uix.recycleview.datamodel), 493

RecycleDataViewBehavior (class in kivy.uix.recycleview.views), 499

RecycleGridLayout (class in kivy.uix.recyclegridlayout), 586

RecycleLayout (class in kivy.uix.recyclelayout), 586

RecycleLayoutManagerBehavior (class in kivy.uix.recycleview.layout), 496

RecycleView (class in kivy.uix.recycleview), 492

recycleview (kivy.uix.recycleview.datamodel.RecycleDataModelBehavior attribute), 493

recycleview (kivy.uix.recycleview.views.RecycleDataAdapter attribute), 498

RecycleViewBehavior (class in kivy.uix.recycleview), 491

RedisStore (class in kivy.storage.redisstore), 449

reference_size (kivy.uix.behaviors.cover.CoverBehavior attribute), 474

reference_size (kivy.uix.behaviors.CoverBehavior attribute), 464

ReferenceListProperty (class in kivy.properties), 246

refresh() (kivy.core.text.LabelBase method), 271

refresh() (kivy.uix.vkeyboard.VKeyboard method), 679

refresh_fbo_setup() (kivy.uix.effectwidget.EffectWidget method), 538

refresh_from_data() (kivy.uix.recycleview.RecycleViewBehavior method), 288
 refresh_from_data() (kivy.uix.recycleview.RecycleViewBehavior method), 491
 refresh_from_layout() (kivy.uix.recycleview.RecycleViewBehavior reload() method), 491
 refresh_from_layout() (kivy.uix.recycleview.RecycleViewBehavior reload() method), 562
 refresh_from_viewport() (kivy.uix.recycleview.RecycleViewBehavior reload() method), 561
 refresh_from_viewport() (kivy.uix.recycleview.RecycleViewBehavior remove() method), 191
 refresh_from_viewport() (kivy.uix.recycleview.RecycleViewBehavior remove() method), 492
 refresh_from_viewport() (kivy.uix.recycleview.RecycleViewBehavior remove() method), 310
 refresh_view_attrs() (kivy.uix.recycleview.views.RecycleDataAdapter reload() method), 498
 refresh_view_attrs() (kivy.uix.recycleview.views.RecycleDataAdapter reload() method), 314
 refresh_view_attrs() (kivy.uix.recycleview.views.RecycleDataAdapter reload() method), 329
 refresh_view_attrs() (kivy.uix.recycleview.views.RecycleDataAdapter reload() method), 327
 refresh_view_layout() (kivy.uix.recyclelayout.RecycleLayout remove() method), 586
 refresh_view_layout() (kivy.uix.recyclelayout.RecycleLayout remove() method), 327
 refresh_view_layout() (kivy.uix.recyclelayout.RecycleLayout remove_android_splash() method), 586
 refresh_view_layout() (kivy.uix.recyclelayout.RecycleLayoutManagerBehavior remove_android_splash() method), 497
 refresh_view_layout() (kivy.uix.recyclelayout.RecycleLayoutManagerBehavior remove_android_splash() method), 207
 refresh_view_layout() (kivy.uix.recycleview.views.RecycleDataAdapter remove_android_splash() method), 498
 refresh_view_layout() (kivy.uix.recycleview.views.RecycleDataAdapter remove_android_splash() method), 223
 refresh_view_layout() (kivy.uix.recycleview.views.RecycleDataAdapter remove_android_splash() method), 499
 refresh_view_layout() (kivy.uix.recycleview.views.RecycleDataAdapter remove_android_splash() method), 265
 refs (kivy.core.text.markup.MarkupLabel attribute), 276
 refs (kivy.uix.label.Label attribute), 571
 register() (kivy.cache.Cache static method), 190
 register() (kivy.core.audio.SoundLoader static method), 261
 register() (kivy.core.text.LabelBase static method), 271
 register() (kivy.input.factory.MotionEventFactory static method), 402
 register() (kivy.input.MotionEventFactory static method), 385
 register() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 394
 register_context() (in module kivy.context), 208
 register_event_type() (kivy.event.EventDispatcher method), 214
 register_type() (kivy.uix.settings.Settings method), 622
 reify (class in kivy.utils), 254
 RelativeLayout (class in kivy.uix.relativelayout), 591
 release() (kivy.clock.ClockEvent method), 197
 release() (kivy.core.window.Keyboard method), 279
 release() (kivy.graphics.Fbo method), 313
 release() (kivy.graphics.fbo.Fbo method), 338
 release_all_keyboards() (kivy.core.window.WindowBase method), 287
 release_keyboard() (kivy.core.window.WindowBase method), 434
 reload() (kivy.graphics.context.Context method), 336
 reload() (kivy.uix.image.AsyncImage method), 562
 reload() (kivy.uix.image.Image method), 561
 reload() (kivy.cache.Cache static method), 191
 reload() (kivy.graphics.Canvas method), 310
 reload() (kivy.graphics.InstructionGroup method), 314
 reload() (kivy.graphics.instructions.Canvas method), 329
 reload() (kivy.graphics.instructions.InstructionGroup method), 327
 remove_android_splash() (kivy.base.EventLoopBase method), 586
 remove_callback() (kivy.config.ConfigParser method), 207
 remove_event_listener() (kivy.base.EventLoopBase method), 223
 remove_from_cache() (kivy.core.image.Image method), 265
 remove_group() (kivy.graphics.InstructionGroup method), 314
 remove_group() (kivy.graphics.instructions.InstructionGroup method), 327
 remove_handler() (kivy.base.ExceptionManagerBase method), 224
 remove_input_provider() (kivy.base.EventLoopBase method), 223
 remove_node() (kivy.uix.treeview.TreeView method), 666
 remove_postproc_module() (kivy.base.EventLoopBase method), 223
 remove_reload_observer() (kivy.graphics.context.Context method), 336
 remove_reload_observer() (kivy.graphics.Fbo method), 313
 remove_reload_observer() (kivy.graphics.fbo.Fbo method), 339
 remove_reload_observer() (kivy.graphics.texture.Texture method), 365
 remove_screen() (kivy.uix.screenmanager.ShaderTransition method), 610
 remove_screen() (kivy.uix.screenmanager.TransitionBase method), 609
 remove_toolbar_widget() (kivy.modules.console.Console method), 434

remove_widget() (kivy.core.window.WindowBase method), 288
 remove_widget() (kivy.uix.accordion.AccordionItem method), 503
 remove_widget() (kivy.uix.actionbar.ActionView method), 509
 remove_widget() (kivy.uix.boxlayout.BoxLayout method), 515
 remove_widget() (kivy.uix.bubble.Bubble method), 519
 remove_widget() (kivy.uix.carousel.Carousel method), 525
 remove_widget() (kivy.uix.dropdown.DropDown method), 535
 remove_widget() (kivy.uix.effectwidget.EffectWidget method), 538
 remove_widget() (kivy.uix.floatlayout.FloatLayout method), 551
 remove_widget() (kivy.uix.layout.Layout method), 577
 remove_widget() (kivy.uix.recycleview.RecycleView method), 492
 remove_widget() (kivy.uix.sandbox.Sandbox method), 596
 remove_widget() (kivy.uix.scatterlayout.ScatterLayout method), 602
 remove_widget() (kivy.uix.screenmanager.ScreenManager method), 608
 remove_widget() (kivy.uix.scrollview.ScrollView method), 616
 remove_widget() (kivy.uix.settings.ContentPanel method), 627
 remove_widget() (kivy.uix.splitter.Splitter method), 636
 remove_widget() (kivy.uix.tabbedpanel.TabbedPanel method), 646
 remove_widget() (kivy.uix.tabbedpanel.TabbedPanelItem method), 648
 remove_widget() (kivy.uix.widget.Widget method), 689
 render() (kivy.core.text.LabelBase method), 271
 render() (kivy.core.text.markup.MarkupLabel method), 276
 render() (kivy.uix.rst.RstDocument method), 594
 RenderContext (class in kivy.graphics), 321
 RenderContext (class in kivy.graphics.instructions), 330
 replace_crlf (kivy.uix.textinput.TextInput attribute), 659
 req_body (kivy.network.urlrequest.UrlRequest attribute), 443
 req_headers (kivy.network.urlrequest.UrlRequest attribute), 443
 request_keyboard() (kivy.core.window.WindowBase method), 288
 require() (in module kivy), 159
 rescale_with_parent (kivy.uix.splitter.Splitter attribute), 637
 reset() (kivy.effects.scroll.ScrollEffect method), 298
 reset_context (kivy.graphics.Callback attribute), 309
 reset_context (kivy.graphics.instructions.Callback attribute), 331
 reset_undo() (kivy.uix.textinput.TextInput method), 660
 resolution (kivy.core.camera.CameraBase attribute), 262
 resolution (kivy.uix.camera.Camera attribute), 522
 resolve_path() (kivy.uix.rst.RstDocument method), 594
 resource_add_path() (in module kivy.resources), 251
 resource_find() (in module kivy.resources), 251
 resource_remove_path() (in module kivy.resources), 251
 resp_headers (kivy.network.urlrequest.UrlRequest attribute), 444
 resp_status (kivy.network.urlrequest.UrlRequest attribute), 444
 reset() (kivy.core.window.WindowBase method), 288
 request (kivy.network.urlrequest.UrlRequest attribute), 444
 resume() (kivy.loader.LoaderBase method), 186
 rgb (kivy.graphics.ClearColor attribute), 325
 rgb (kivy.graphics.Color attribute), 311
 rgb (kivy.graphics.context_instructions.Color attribute), 333
 rgb (kivy.graphics.gl_instructions.ClearColor attribute), 339
 rgba (kivy.graphics.ClearColor attribute), 325
 rgba (kivy.graphics.Color attribute), 311
 rgba (kivy.graphics.context_instructions.Color attribute), 333
 rgba (kivy.graphics.gl_instructions.ClearColor attribute), 339
 rgba() (in module kivy.utils), 254
 right (kivy.uix.widget.Widget attribute), 689
 right_count (kivy.uix.behaviors.compoundselection.CompoundSelection attribute), 471
 right_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461
 ripple_duration_in (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 487
 ripple_duration_in (kivy.uix.behaviors.TouchRippleBehavior attribute), 464
 ripple_duration_out (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 487
 ripple_duration_out (kivy.uix.behaviors.TouchRippleBehavior attribute), 464

ripple_fade() (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 296
 ripple_fade() (kivy.uix.behaviors.TouchRippleBehavior method), 488
 ripple_fade_from_alpha (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_fade_to_alpha (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_fade_to_alpha (kivy.uix.behaviors.TouchRippleBehavior attribute), 465
 ripple_func_in (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_func_in (kivy.uix.behaviors.TouchRippleBehavior attribute), 465
 ripple_func_out (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_func_out (kivy.uix.behaviors.TouchRippleBehavior attribute), 465
 ripple_rad_default (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_rad_default (kivy.uix.behaviors.TouchRippleBehavior attribute), 465
 ripple_scale (kivy.uix.behaviors.touchripple.TouchRippleBehavior attribute), 488
 ripple_scale (kivy.uix.behaviors.TouchRippleBehavior attribute), 465
 ripple_show() (kivy.uix.behaviors.touchripple.TouchRippleBehavior method), 488
 ripple_show() (kivy.uix.behaviors.TouchRippleBehavior method), 465
 RiseInTransition (class in kivy.uix.screenmanager), 611
 root (kivy.app.App attribute), 182
 root (kivy.uix.treeview.TreeView attribute), 666
 root_options (kivy.uix.treeview.TreeView attribute), 666
 root_window (kivy.app.App attribute), 182
 rootpath (kivy.uix.filechooser.FileChooserController attribute), 548
 Rotate (class in kivy.graphics), 322
 Rotate (class in kivy.graphics.context_instructions), 334
 rotate() (kivy.graphics.transformation.Matrix method), 368
 rotate() (kivy.vector.Vector method), 257
 rotation (kivy.core.window.WindowBase attribute), 288
 rotation (kivy.uix.scatter.Scatter attribute), 599
 round_value (kivy.effects.dampedscroll.DampedScrollEffect attribute), 296
 rounded_rectangle (kivy.graphics.Line attribute), 318
 rounded_rectangle (kivy.graphics.vertex_instructions.Line attribute), 376
 RoundedRectangle (class in kivy.graphics.vertex_instructions), 371
 row_default_height (kivy.uix.gridlayout.GridLayout attribute), 558
 row_force_default (kivy.uix.gridlayout.GridLayout attribute), 558
 row_height (kivy.core.image.ImageData attribute), 266
 rows (kivy.uix.gridlayout.GridLayout attribute), 558
 rows_minimum (kivy.uix.gridlayout.GridLayout attribute), 558
 RstDocument (class in kivy.uix.rst), 593
 run() (kivy.app.App method), 182
 run() (kivy.base.EventLoopBase method), 223
 run() (kivy.base.loader.LoaderBase method), 186
 run() (kivy.network.urlrequest.UrlRequest method), 444
 runTouchApp() (in module kivy.app), 183
 runTouchApp() (in module kivy.base), 224
 s (kivy.graphics.Color attribute), 311
 save() (kivy.graphics.context_instructions.Color attribute), 333
 saveIn() (kivy.interactive.SafeMembrane method), 222
 SafeMembrane (class in kivy.utils), 253
 SafeMembrane (class in kivy.interactive), 222
 SafeOut() (kivy.interactive.SafeMembrane method), 222
 Sandbox (class in kivy.uix.sandbox), 595
 save() (kivy.core.image.Image method), 265
 save() (kivy.graphics.texture.Texture method), 365
 Scale (class in kivy.graphics), 323
 Scale (class in kivy.graphics.context_instructions), 334
 scale (kivy.graphics.context_instructions.Scale attribute), 335
 scale (kivy.graphics.Scale attribute), 323
 scale (kivy.uix.scatter.Scatter attribute), 600
 scale() (kivy.graphics.transformation.Matrix method), 368
 scale_for_screen() (kivy.input.MotionEvent method), 384
 scale_for_screen() (kivy.input.motionevent.MotionEvent method), 401
 scale_max (kivy.uix.scatter.Scatter attribute), 600
 scale_min (kivy.uix.scatter.Scatter attribute), 600

scale_stroke() (kivy.gesture.GestureStroke method), 219
 ScanlinesEffect (class in kivy.uix.effectwidget), 539
 Scatter (class in kivy.uix.scatter), 597
 ScatterLayout (class in kivy.uix.scatterlayout), 601
 ScatterPlane (class in kivy.uix.scatter), 600
 ScatterPlaneLayout (class in kivy.uix.scatterlayout), 602
 schedule_del_safe() (kivy.clock.CyClockBase method), 198
 schedule_interval() (kivy.clock.CyClockBase method), 198
 schedule_interval() (kivy.clock.CyClockBaseFree method), 199
 schedule_interval_free() (kivy.clock.CyClockBaseFree method), 199
 schedule_once() (kivy.clock.CyClockBase method), 198
 schedule_once() (kivy.clock.CyClockBaseFree method), 199
 schedule_once_free() (kivy.clock.CyClockBaseFree method), 199
 ScissorPop (class in kivy.graphics.scissor_instructions), 352
 ScissorPush (class in kivy.graphics.scissor_instructions), 352
 ScissorStack (class in kivy.graphics.scissor_instructions), 352
 Screen (class in kivy.uix.screenmanager), 605
 screen_in (kivy.uix.screenmanager.TransitionBase attribute), 609
 screen_names (kivy.uix.screenmanager.ScreenManager attribute), 608
 screen_out (kivy.uix.screenmanager.TransitionBase attribute), 609
 ScreenManager (class in kivy.uix.screenmanager), 606
 ScreenManagerException, 609
 screens (kivy.uix.screenmanager.ScreenManager attribute), 608
 screenshot() (kivy.core.window.WindowBase method), 289
 scroll (kivy.effects.scroll.ScrollEffect attribute), 298
 scroll_count (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 471
 scroll_count (kivy.uix.behaviors.CompoundSelectionBehavior method), 496
 scroll_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 461
 scroll_distance (kivy.uix.carousel.Carousel attribute), 526
 scroll_distance (kivy.uix.scrollview.ScrollView attribute), 616
 scroll_timeout (kivy.uix.carousel.Carousel attribute), 526
 scroll_timeout (kivy.uix.scrollview.ScrollView attribute), 616
 scroll_to() (kivy.uix.scrollview.ScrollView method), 617
 scroll_type (kivy.uix.scrollview.ScrollView attribute), 617
 scroll_wheel_distance (kivy.uix.scrollview.ScrollView attribute), 617
 scroll_x (kivy.uix.scrollview.ScrollView attribute), 617
 scroll_x (kivy.uix.textinput.TextInput attribute), 660
 scroll_y (kivy.uix.scrollview.ScrollView attribute), 617
 scroll_y (kivy.uix.textinput.TextInput attribute), 660
 ScrollEffect (class in kivy.effects.scroll), 298
 ScrollView (class in kivy.uix.scrollview), 614
 section (kivy.uix.settings.SettingItem attribute), 623
 seek() (kivy.core.audio.Sound method), 260
 seek() (kivy.core.video.VideoBase method), 277
 seek() (kivy.uix.video.Video method), 669
 seek() (kivy.uix.videoplayer.VideoPlayer method), 674
 segment_intersection() (kivy.vector.Vector static method), 257
 segments (kivy.graphics.Bezier attribute), 307
 segments (kivy.graphics.Ellipse attribute), 312
 segments (kivy.graphics.vertex_instructions.Bezier attribute), 379
 segments (kivy.graphics.vertex_instructions.Ellipse attribute), 372
 segments (kivy.graphics.vertex_instructions.RoundedRectangle attribute), 371
 select() (kivy.uix.dropdown.DropDown method), 535
 select_all() (kivy.uix.textinput.TextInput method), 660
 select_language() (kivy.core.spelling.SpellingBase method), 267
 select_node() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 471
 select_node() (kivy.uix.behaviors.CompoundSelectionBehavior method), 471
 select_node() (kivy.uix.recycleview.layout.LayoutSelectionBehavior method), 471
 select_node() (kivy.uix.treeview.TreeView method), 666
 select_text() (kivy.uix.textinput.TextInput method), 660

select_with_key_down() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 471	set() (kivy.properties.ListProperty method), 244
select_with_key_down() (kivy.uix.behaviors.CompoundSelectionBehavior method), 462	set() (kivy.properties.ReferenceListProperty method), 246
select_with_key_up() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 471	set_config() (kivy.properties.ConfigParserProperty method), 250
select_with_key_up() (kivy.uix.behaviors.CompoundSelectionBehavior method), 462	set_content() (kivy.modules.console.Console method), 434
select_with_touch() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 471	set_fbo_shader() (kivy.uix.effectwidget.AdvancedEffectBase method), 539
select_with_touch() (kivy.uix.behaviors.CompoundSelectionBehavior method), 462	set_fbo_shader() (kivy.uix.effectwidget.EffectBase method), 538
select_with_touch() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 471	set_min() (kivy.properties.BoundedNumericProperty method), 245
select_with_touch() (kivy.uix.behaviors.CompoundSelectionBehavior method), 462	set_min() (kivy.properties.BoundedNumericProperty method), 245
selected_alpha (kivy.uix.settings.SettingItem attribute), 623	set_system_cursor() (kivy.core.window.WindowBase method), 289
selected_node (kivy.uix.treeview.TreeView attribute), 666	set_system_cursor() (kivy.core.window.WindowBase method), 289
selected_nodes (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 472	set_title() (kivy.core.window.WindowBase method), 289
selected_nodes (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 462	set_titlebar_views() (kivy.uix.recyclelayout.RecycleLayout method), 587
selected_uid (kivy.uix.settings.MenuSidebar attribute), 628	set_visible_views() (kivy.uix.recycleview.layout.RecycleLayoutManager method), 497
selection (kivy.uix.filechooser.FileChooserController attribute), 548	set_visible_views() (kivy.uix.recycleview.views.RecycleDataAdapter method), 498
selection_color (kivy.uix.textinput.TextInput attribute), 660	set_vkeyboard_class() (kivy.core.window.WindowBase method), 289
selection_from (kivy.uix.textinput.TextInput attribute), 660	set_window() (kivy.base.EventLoopBase method), 223
selection_text (kivy.uix.textinput.TextInput attribute), 660	setall() (kivy.config.ConfigParser method), 208
selection_to (kivy.uix.textinput.TextInput attribute), 660	setdefault() (kivy.config.ConfigParser method), 208
sensitivity (kivy.uix.slider.Slider attribute), 631	setdefaults() (kivy.config.ConfigParser method), 208
separator_color (kivy.uix.popup.Popup attribute), 584	setitem() (kivy.properties.ReferenceListProperty method), 246
separator_height (kivy.uix.popup.Popup attribute), 584	setter() (kivy.event.EventDispatcher method), 215
separator_image (kivy.uix.actionbar.ActionGroup attribute), 507	SettingBoolean (class in kivy.uix.settings), 624
separator_width (kivy.uix.actionbar.ActionGroup attribute), 507	SettingItem (class in kivy.uix.settings), 622
set() (kivy.config.ConfigParser method), 207	SettingNumeric (class in kivy.uix.settings), 625
set() (kivy.graphics.context_instructions.Rotate method), 334	SettingOptions (class in kivy.uix.settings), 625
set() (kivy.graphics.Rotate method), 322	SettingPath (class in kivy.uix.settings), 624
set() (kivy.graphics.transformation.Matrix method), 368	Settings (class in kivy.uix.settings), 621
set() (kivy.properties.AliasProperty method), 247	settings (kivy.uix.settings.SettingsPanel attribute), 622
set() (kivy.properties.ConfigParserProperty method), 250	settings_cls (kivy.app.App attribute), 182
set() (kivy.properties.DictProperty method), 248	SettingsPanel (class in kivy.uix.settings), 622
	SettingString (class in kivy.uix.settings), 624
	SettingsWithNoMenu (class in kivy.uix.settings), 625
	SettingsWithSidebar (class in kivy.uix.settings),

[625](#)
 SettingsWithSpinner (class in `kivy.uix.settings`), [625](#)
 SettingsWithTabbedPanel (class in `kivy.uix.settings`), [625](#)
 SettingTitle (class in `kivy.uix.settings`), [625](#)
 setup_mode() (`kivy.uix.vkeyboard.VKeyboard` method), [679](#)
 setup_mode_dock() (`kivy.uix.vkeyboard.VKeyboard` method), [679](#)
 setup_mode_free() (`kivy.uix.vkeyboard.VKeyboard` method), [679](#)
 Shader (class in `kivy.graphics.shader`), [354](#)
 shader (`kivy.graphics.instructions.RenderContext` attribute), [330](#)
 shader (`kivy.graphics.RenderContext` attribute), [322](#)
 ShaderTransition (class in `kivy.uix.screenmanager`), [610](#)
 Shape (class in `kivy.input.shape`), [402](#)
 shape (`kivy.input.MotionEvent` attribute), [384](#)
 shape (`kivy.input.motionevent.MotionEvent` attribute), [401](#)
 shape_color_key (`kivy.core.window.WindowBase` attribute), [289](#)
 shape_cutoff (`kivy.core.window.WindowBase` attribute), [289](#)
 shape_image (`kivy.core.window.WindowBase` attribute), [290](#)
 shape_mode (`kivy.core.window.WindowBase` attribute), [290](#)
 shaped (`kivy.core.window.WindowBase` attribute), [290](#)
 ShapeRect (class in `kivy.input.shape`), [402](#)
 shorten (`kivy.uix.label.Label` attribute), [572](#)
 shorten() (`kivy.core.text.LabelBase` method), [271](#)
 shorten_from (`kivy.uix.label.Label` attribute), [572](#)
 shorten_post() (`kivy.core.text.markup.MarkupLabel` method), [276](#)
 show() (`kivy.core.window.WindowBase` method), [290](#)
 show_arrow (`kivy.uix.bubble.Bubble` attribute), [519](#)
 show_cursor (`kivy.core.window.WindowBase` attribute), [290](#)
 show_errors (`kivy.uix.rst.RstDocument` attribute), [594](#)
 show_hidden (`kivy.uix.filechooser.FileChooserController` attribute), [548](#)
 show_hidden (`kivy.uix.settings.SettingPath` attribute), [624](#)
 show_keyboard() (`kivy.uix.behaviors.focus.FocusBehaviors` method), [480](#)
 show_keyboard() (`kivy.uix.behaviors.FocusBehaviors` method), [459](#)
 single_points_test() (`kivy.uix.gesturesurface.GestureContainer` method), [555](#)
 sizable_from (`kivy.uix.splitter.Splitter` attribute), [637](#)
 size (`kivy.core.image.Image` attribute), [266](#)
 size (`kivy.core.image.ImageData` attribute), [266](#)
 size (`kivy.core.window.WindowBase` attribute), [290](#)
 size (`kivy.graphics.Fbo` attribute), [313](#)
 size (`kivy.graphics.fbo.Fbo` attribute), [339](#)
 size (`kivy.graphics.Rectangle` attribute), [321](#)
 size (`kivy.graphics.texture.Texture` attribute), [366](#)
 size (`kivy.graphics.vertex_instructions.Rectangle` attribute), [371](#)
 size (`kivy.uix.effectwidget.HorizontalBlurEffect` attribute), [540](#)
 size (`kivy.uix.effectwidget.VerticalBlurEffect` attribute), [540](#)
 size (`kivy.uix.widget.Widget` attribute), [689](#)
 size_hint (`kivy.uix.widget.Widget` attribute), [689](#)
 size_hint_max (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_max_x (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_max_y (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_min (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_min_x (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_min_y (`kivy.uix.widget.Widget` attribute), [690](#)
 size_hint_x (`kivy.uix.widget.Widget` attribute), [691](#)
 size_hint_y (`kivy.uix.widget.Widget` attribute), [691](#)
 Slider (class in `kivy.uix.slider`), [629](#)
 slides (`kivy.uix.carousel.Carousel` attribute), [526](#)
 SlideTransition (class in `kivy.uix.screenmanager`), [610](#)
 smooth_scroll_end (`kivy.uix.scrollview.ScrollView` attribute), [617](#)
 SmoothLine (class in `kivy.graphics`), [319](#)
 SmoothLine (class in `kivy.graphics.vertex_instructions`), [379](#)
 softinput_mode (`kivy.core.window.WindowBase` attribute), [291](#)
 sort_func (`kivy.uix.filechooser.FileChooserController` attribute), [548](#)
 Sound (class in `kivy.core.audio`), [260](#)
 SoundLoader (class in `kivy.core.audio`), [261](#)
 source (`kivy.core.audio.Sound` attribute), [261](#)
 source (`kivy.core.image.ImageData` attribute), [266](#)
 source (`kivy.graphics.BindTexture` attribute), [307](#)
 source (`kivy.graphics.context_instructions.BindTexture` attribute), [333](#)

source (kivy.graphics.instructions.VertexInstruction attribute), 328
 source (kivy.graphics.shader.Shader attribute), 354
 source (kivy.graphics.svg.Svg attribute), 357
 source (kivy.graphics.VertexInstruction attribute), 324
 source (kivy.uix.effectwidget.EffectBase attribute), 539
 source (kivy.uix.image.Image attribute), 561
 source (kivy.uix.rst.RstDocument attribute), 594
 source (kivy.uix.videoplayer.VideoPlayer attribute), 674
 source_encoding (kivy.uix.rst.RstDocument attribute), 594
 source_error (kivy.uix.rst.RstDocument attribute), 594
 source_stack (kivy.graphics.ApplyContextMatrix attribute), 326
 sp() (in module kivy.metrics), 228
 spacing (kivy.uix.boxlayout.BoxLayout attribute), 515
 spacing (kivy.uix.gridlayout.GridLayout attribute), 559
 spacing (kivy.uix.stacklayout.StackLayout attribute), 639
 SpellingBase (class in kivy.core.spelling), 267
 Spinner (class in kivy.uix.spinner), 633
 SpinnerOption (class in kivy.uix.spinner), 634
 split_str (kivy.uix.label.Label attribute), 572
 Splitter (class in kivy.uix.splitter), 635
 spos (kivy.input.MotionEvent attribute), 384
 spos (kivy.input.motionevent.MotionEvent attribute), 401
 spring_constant (kivy.effects.dampedscroll.DampedScrollEffect attribute), 296
 stack (kivy.graphics.context_instructions.MatrixInstruction attribute), 335
 stack (kivy.graphics.context_instructions.PopMatrix attribute), 334
 stack (kivy.graphics.context_instructions.PushMatrix attribute), 334
 stack (kivy.graphics.LoadIdentity attribute), 327
 stack (kivy.graphics.MatrixInstruction attribute), 319
 stack (kivy.graphics.PopMatrix attribute), 321
 stack (kivy.graphics.PushMatrix attribute), 321
 StackLayout (class in kivy.uix.stacklayout), 638
 start (kivy.uix.videoplayer.VideoPlayerAnnotation attribute), 675
 start() (in module kivy.modules.console), 432
 start() (kivy.animation.Animation method), 162
 start() (kivy.base.EventLoopBase method), 223
 start() (kivy.core.camera.CameraBase method), 262
 start() (kivy.effects.kinetic.KineticEffect method), 297
 start() (kivy.effects.scroll.ScrollEffect method), 298
 start() (kivy.input.MotionEventProvider method), 384
 start() (kivy.input.provider.MotionEventProvider method), 402
 start() (kivy.input.providers.androidjoystick.AndroidMotionEventProvider method), 388
 start() (kivy.input.providers.leapfinger.LeapFingerEventProvider method), 389
 start() (kivy.input.providers.mactouch.MacMotionEventProvider method), 392
 start() (kivy.input.providers.mouse.MouseMotionEventProvider method), 390
 start() (kivy.input.providers.tuio.TuioMotionEventProvider method), 394
 start() (kivy.loader.LoaderBase method), 186
 start() (kivy.uix.screenmanager.CardTransition method), 612
 start() (kivy.uix.screenmanager.SwapTransition method), 611
 start() (kivy.uix.screenmanager.TransitionBase method), 609
 state (kivy.core.audio.Sound attribute), 261
 state (kivy.core.video.VideoBase attribute), 277
 state (kivy.uix.behaviors.button.ButtonBehavior attribute), 467
 state (kivy.uix.behaviors.ButtonBehavior attribute), 455
 state (kivy.uix.video.Video attribute), 669
 state (kivy.uix.videoplayer.VideoPlayer attribute), 674
 status (kivy.core.audio.Sound attribute), 261
 StencilEffect (class in kivy.graphics), 323
 StencilPop (class in kivy.graphics.stencil_instructions), 356
 StencilPush (class in kivy.graphics), 323
 StencilPush (class in kivy.graphics.stencil_instructions), 355
 StencilUnUse (class in kivy.graphics), 324
 StencilUnUse (class in kivy.graphics.stencil_instructions), 356
 StencilUse (class in kivy.graphics), 323
 StencilUse (class in kivy.graphics.stencil_instructions), 356
 StencilView (class in kivy.uix.stencilview), 639
 step (kivy.uix.slider.Slider attribute), 631
 stop() (in module kivy.modules.console), 433
 stop() (in module kivy.modules.inspector), 435
 stop() (in module kivy.modules.joycursor), 436
 stop() (kivy.animation.Animation method), 163

stop() (kivy.app.App method), 183
 stop() (kivy.base.EventLoopBase method), 223
 stop() (kivy.core.audio.Sound method), 261
 stop() (kivy.core.camera.CameraBase method), 262
 stop() (kivy.core.video.VideoBase method), 277
 stop() (kivy.effects.kinetic.KineticEffect method), 297
 stop() (kivy.effects.scroll.ScrollEffect method), 299
 stop() (kivy.input.MotionEventProvider method), 385
 stop() (kivy.input.provider.MotionEventProvider method), 402
 stop() (kivy.input.providers.androidjoystick.AndroidJoystickProvider method), 388
 stop() (kivy.input.providers.mactouch.MacMotionEventProvider method), 392
 stop() (kivy.input.providers.mouse.MouseMotionEventProvider method), 390
 stop() (kivy.input.providers.tuio.TuioMotionEventProvider method), 394
 stop() (kivy.loader.LoaderBase method), 186
 stop() (kivy.multistroke.ProgressTracker method), 233
 stop() (kivy.uix.screenmanager.ShaderTransition method), 610
 stop() (kivy.uix.screenmanager.TransitionBase method), 610
 stop_all() (kivy.animation.Animation static method), 163
 stop_property() (kivy.animation.Animation method), 163
 stopTouchApp() (in module kivy.app), 184
 stopTouchApp() (in module kivy.base), 224
 str_to_gesture() (kivy.gesture.GestureDatabase method), 219
 strikethrough (kivy.uix.label.Label attribute), 572
 string_to_keycode() (kivy.core.window.Keyboard method), 279
 string_types (in module kivy.compat), 201
 StringProperty (class in kivy.properties), 243
 strip (kivy.uix.label.Label attribute), 572
 strip_border (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 strip_cls (kivy.uix.splitter.Splitter attribute), 637
 strip_comments() (kivy.lang.Parser method), 419
 strip_comments() (kivy.lang.parser.Parser method), 423
 strip_image (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 strip_size (kivy.uix.splitter.Splitter attribute), 637
 StripLayout (class in kivy.uix.tabbedpanel), 644
 stroke_length() (kivy.gesture.GestureStroke method), 219
 strtotuple() (in module kivy.utils), 252
 style (kivy.uix.codeinput.CodeInput attribute), 529
 style_name (kivy.uix.codeinput.CodeInput attribute), 529
 success (kivy.graphics.shader.Shader attribute), 354
 suggest() (kivy.core.spelling.SpellingBase method), 267
 suggestion_text (kivy.uix.textinput.TextInput attribute), 661
 Svg (class in kivy.graphics.svg), 356
 SwapTransition (class in kivy.uix.screenmanager), 611
 Switch (class in kivy.uix.switch), 640
 switch_to() (kivy.uix.screenmanager.ScreenManager method), 608
 switch_to() (kivy.uix.tabbedpanel.TabbedPanel method), 646
 sx (kivy.input.MotionEvent attribute), 384
 sx (kivy.input.motionevent.MotionEvent attribute), 401
 sy (kivy.input.MotionEvent attribute), 384
 sy (kivy.input.motionevent.MotionEvent attribute), 401
 sync() (kivy.lang.builder.BuilderBase method), 421
 sync() (kivy.lang.BuilderBase method), 417
 sync_height (kivy.uix.spinner.Spinner attribute), 634
 system_size (kivy.core.window.WindowBase attribute), 291
 sz (kivy.input.MotionEvent attribute), 384
 sz (kivy.input.motionevent.MotionEvent attribute), 401

T

 tab_height (kivy.uix.tabbedpanel.TabbedPanel attribute), 646
 tab_list (kivy.uix.tabbedpanel.TabbedPanel attribute), 647
 tab_pos (kivy.uix.tabbedpanel.TabbedPanel attribute), 647
 tab_width (kivy.uix.tabbedpanel.TabbedPanel attribute), 647
 tab_width (kivy.uix.textinput.TextInput attribute), 661
 tabbed_panel (kivy.uix.tabbedpanel.TabbedPanelStrip attribute), 648
 TabbedPanel (class in kivy.uix.tabbedpanel), 644
 TabbedPanelContent (class in kivy.uix.tabbedpanel), 647
 TabbedPanelException, 648
 TabbedPanelHeader (class in kivy.uix.tabbedpanel), 647
 TabbedPanelItem (class in kivy.uix.tabbedpanel), 647

TabbedPanelStrip (class in kivy.uix.tabbedpanel), 648
 target (kivy.core.window.Keyboard attribute), 279
 target (kivy.graphics.texture.Texture attribute), 366
 target (kivy.uix.vkeyboard.VKeyboard attribute), 679
 target_stack (kivy.graphics.ApplyContextMatrix attribute), 326
 target_widget (kivy.effects.scroll.ScrollEffect attribute), 299
 template() (kivy.lang.builder.BuilderBase method), 421
 template() (kivy.lang.BuilderBase method), 417
 tessellate() (kivy.graphics.tessellator.Tessellator method), 359
 Tessellator (class in kivy.graphics.tessellator), 359
 tex_coords (kivy.graphics.instructions.VertexInstruction attribute), 328
 tex_coords (kivy.graphics.texture.Texture attribute), 366
 tex_coords (kivy.graphics.VertexInstruction attribute), 325
 text (kivy.core.text.LabelBase attribute), 271
 text (kivy.core.text.text_layout.LayoutWord attribute), 274
 text (kivy.uix.label.Label attribute), 573
 text (kivy.uix.rst.RstDocument attribute), 594
 text (kivy.uix.textinput.TextInput attribute), 661
 text_autoupdate (kivy.uix.spinner.Spinner attribute), 634
 text_entry_timeout (kivy.uix.behaviors.compoundselection.CompoundSelection attribute), 472
 text_entry_timeout (kivy.uix.behaviors.CompoundSelection attribute), 462
 text_language (kivy.uix.label.Label attribute), 573
 text_language (kivy.uix.textinput.TextInput attribute), 661
 text_size (kivy.core.text.LabelBase attribute), 272
 text_size (kivy.uix.label.Label attribute), 573
 text_validate_unfocus (kivy.uix.textinput.TextInput attribute), 661
 TextInput (class in kivy.uix.textinput), 651
 textinput (kivy.uix.settings.SettingPath attribute), 624
 textinput (kivy.uix.settings.SettingString attribute), 624
 Texture (class in kivy.graphics.texture), 363
 texture (kivy.core.camera.CameraBase attribute), 262
 texture (kivy.core.image.Image attribute), 266
 texture (kivy.core.video.VideoBase attribute), 277
 texture (kivy.graphics.Fbo attribute), 313
 texture (kivy.graphics.fbo.Fbo attribute), 339
 texture (kivy.graphics.instructions.VertexInstruction attribute), 328
 texture (kivy.graphics.VertexInstruction attribute), 325
 texture (kivy.uix.effectwidget.EffectWidget attribute), 538
 texture (kivy.uix.image.Image attribute), 561
 texture (kivy.uix.label.Label attribute), 573
 texture_size (kivy.uix.image.Image attribute), 561
 texture_size (kivy.uix.label.Label attribute), 574
 texture_update() (kivy.uix.label.Label method), 574
 TextureRegion (class in kivy.graphics.texture), 366
 textures (kivy.atlas.Atlas attribute), 189
 thumbnail (kivy.uix.videoplayer.VideoPlayer attribute), 674
 tick() (kivy.clock.ClockBaseBehavior method), 200
 tick() (kivy.clock.ClockEvent method), 197
 tick_draw() (kivy.clock.ClockBaseBehavior method), 200
 time (kivy.clock.ClockBaseBehavior attribute), 200
 time_end (kivy.input.MotionEvent attribute), 384
 time_end (kivy.input.motionevent.MotionEvent attribute), 401
 time_start (kivy.input.MotionEvent attribute), 384
 time_start (kivy.input.motionevent.MotionEvent attribute), 401
 time_update (kivy.input.MotionEvent attribute), 384
 time_update (kivy.input.motionevent.MotionEvent attribute), 401
 timeval (kivy.clock.ClockBaseBehavior attribute), 197
 timeval (class in kivy.lib.mtdev), 427
 title (kivy.uix.behaviors.CompoundSelection attribute), 183
 title (kivy.uix.accordion.AccordionItem attribute), 503
 title (kivy.uix.actionbar.ActionPrevious attribute), 510
 title (kivy.uix.popup.Popup attribute), 584
 title (kivy.uix.rst.RstDocument attribute), 594
 title (kivy.uix.settings.SettingItem attribute), 624
 title (kivy.uix.settings.SettingsPanel attribute), 622
 title_align (kivy.uix.popup.Popup attribute), 584
 title_args (kivy.uix.accordion.AccordionItem attribute), 503
 title_color (kivy.uix.popup.Popup attribute), 584
 title_font (kivy.uix.popup.Popup attribute), 584
 title_size (kivy.uix.popup.Popup attribute), 584
 title_template (kivy.uix.accordion.AccordionItem attribute), 503
 to_local() (kivy.uix.relativelayout.RelativeLayout method), 591
 to_local() (kivy.uix.scatter.Scatter method), 600
 to_local() (kivy.uix.scrollview.ScrollView method), 617

to_local() (kivy.uix.widget.Widget method), 691
 to_parent() (kivy.uix.relativelayout.RelativeLayout method), 591
 to_parent() (kivy.uix.scatter.Scatter method), 600
 to_parent() (kivy.uix.scrollview.ScrollView method), 618
 to_parent() (kivy.uix.widget.Widget method), 691
 to_widget() (kivy.uix.widget.Widget method), 691
 to_window() (kivy.uix.widget.Widget method), 691
 toctrees (kivy.uix.rst.RstDocument attribute), 594
 toggle_fullscreen() (kivy.core.window.WindowBase method), 291
 toggle_node() (kivy.uix.treeview.TreeView method), 666
 ToggleButton (class in kivy.uix.togglebutton), 662
 ToggleButtonBehavior (class in kivy.uix.behaviors), 455
 ToggleButtonBehavior (class in kivy.uix.behaviors.togglebutton), 486
 tolist() (kivy.graphics.transformation.Matrix method), 369
 top (kivy.core.window.WindowBase attribute), 291
 top (kivy.uix.widget.Widget attribute), 691
 total (kivy.uix.filechooser.FileChooserProgressBase attribute), 549
 touch_control (kivy.uix.switch.Switch attribute), 641
 touch_deselect_last (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 472
 touch_deselect_last (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 462
 touch_distance (kivy.uix.switch.Switch attribute), 641
 touch_multiselect (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 472
 touch_multiselect (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 463
 touches (kivy.base.EventLoopBase attribute), 223
 TouchRippleBehavior (class in kivy.uix.behaviors), 464
 TouchRippleBehavior (class in kivy.uix.behaviors.touchripple), 487
 TouchRippleButtonBehavior (class in kivy.uix.behaviors), 465
 TouchRippleButtonBehavior (class in kivy.uix.behaviors.touchripple), 488
 transfer_gesture() (kivy.multistroke.Recognizer method), 232
 transform (kivy.uix.scatter.Scatter attribute), 600
 transform_inv (kivy.uix.scatter.Scatter attribute), 600
 transform_point() (kivy.graphics.transformation.Matrix method), 369
 transition (kivy.animation.Animation attribute), 163
 transition (kivy.uix.screenmanager.ScreenManager attribute), 608
 transition_progress (kivy.uix.screenmanager.Screen attribute), 606
 transition_state (kivy.uix.screenmanager.Screen attribute), 606
 TransitionBase (class in kivy.uix.screenmanager), 609
 Translate (class in kivy.graphics), 324
 Translate (class in kivy.graphics.context_instructions), 335
 translate() (kivy.graphics.transformation.Matrix method), 369
 translation_touches (kivy.uix.scatter.Scatter attribute), 600
 transpose() (kivy.graphics.transformation.Matrix method), 369
 TreeView (class in kivy.uix.treeview), 664
 TreeViewException, 666
 TreeViewLabel (class in kivy.uix.treeview), 666
 TreeViewNode (class in kivy.uix.treeview), 666
 Triangle (class in kivy.graphics), 324
 Triangle (class in kivy.graphics.vertex_instructions), 370
 trigger_action() (kivy.uix.behaviors.button.ButtonBehavior method), 467
 trigger_action() (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior method), 455
 trigger_change() (kivy.properties.AliasProperty method), 248
 trigger_change() (kivy.properties.ReferenceListProperty method), 246
 trigger_change() (kivy.properties.ReferenceListProperty method), 336
 trigger_change() (kivy.properties.ReferenceListProperty method), 336
 trigger_time (kivy.input.MotionEvent attribute), 384
 triple_tap_time (kivy.input.motionevent.MotionEvent attribute), 401
 TuiO2dCurMotionEvent (class in kivy.input.providers.tuio), 395
 TuiO2dObjMotionEvent (class in kivy.input.providers.tuio), 395
 TuiOMotionEventProvider (class in kivy.input.providers.tuio), 394
 U
 ud (kivy.input.MotionEvent attribute), 384
 ud (kivy.input.motionevent.MotionEvent attribute), 401
 uid (kivy.input.MotionEvent attribute), 384
 uid (kivy.input.motionevent.MotionEvent attribute), 401

unbind() (kivy.event.EventDispatcher method), 215
 unbind() (kivy.event.Observable method), 216
 unbind() (kivy.lang.builder.Observable method), 419
 unbind() (kivy.lang.Observable method), 415
 unbind() (kivy.properties.Property method), 243
 unbind_property() (kivy.lang.builder.BuilderBase method), 421
 unbind_property() (kivy.lang.BuilderBase method), 417
 unbind_uid() (kivy.event.EventDispatcher method), 215
 unbind_uid() (kivy.event.Observable method), 216
 unbind_uid() (kivy.lang.builder.Observable method), 420
 unbind_uid() (kivy.lang.Observable method), 415
 unbind_uid() (kivy.properties.Property method), 243
 unbind_widget() (kivy.lang.builder.BuilderBase method), 422
 unbind_widget() (kivy.lang.BuilderBase method), 418
 underline (kivy.uix.label.Label attribute), 574
 underline_color (kivy.uix.rst.RstDocument attribute), 594
 unfocus_on_touch (kivy.uix.behaviors.focus.FocusBehavior attribute), 480
 unfocus_on_touch (kivy.uix.behaviors.FocusBehavior attribute), 459
 ungrab() (kivy.input.MotionEvent method), 384
 ungrab() (kivy.input.motionevent.MotionEvent method), 401
 ungrab_mouse() (kivy.core.window.WindowBase method), 291
 unicode_errors (kivy.uix.label.Label attribute), 574
 uniforms (kivy.uix.effectwidget.AdvancedEffectBase attribute), 539
 uninstall_twisted_reactor() (in module kivy.support), 252
 UnistrokeTemplate (class in kivy.multistroke), 235
 unload() (kivy.core.audio.Sound method), 261
 unload() (kivy.core.video.VideoBase method), 277
 unload() (kivy.uix.video.Video method), 670
 unload_file() (kivy.lang.builder.BuilderBase method), 422
 unload_file() (kivy.lang.BuilderBase method), 418
 unregister() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 395
 unregister_event_types() (kivy.event.EventDispatcher method), 216
 unschedule() (kivy.clock.CyClockBase method), 198
 up_count (kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior attribute), 472
 up_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 463
 update() (kivy.effects.kinetic.KineticEffect method), 297
 update() (kivy.effects.scroll.ScrollEffect method), 299
 update() (kivy.input.MotionEventProvider method), 385
 update() (kivy.input.provider.MotionEventProvider method), 402
 update() (kivy.input.providers.androidjoystick.AndroidMotionEventProvider method), 388
 update() (kivy.input.providers.leapfinger.LeapFingerEventProvider method), 389
 update() (kivy.input.providers.mactouch.MacMotionEventProvider method), 392
 update() (kivy.input.providers.mouse.MouseMotionEventProvider method), 390
 update() (kivy.input.providers.tuio.TuioMotionEventProvider method), 395
 update_bbox() (kivy.uix.gesturesurface.GestureContainer method), 555
 update_config() (kivy.config.ConfigParser method), 208
 update_from_scroll() (kivy.uix.scrollview.ScrollView method), 618
 update_velocity() (kivy.effects.dampedscroll.DampedScrollEffect method), 296
 update_velocity() (kivy.effects.kinetic.KineticEffect method), 297
 UpdateNormalMatrix (class in kivy.graphics), 326
 url (kivy.network.urlrequest.UrlRequest attribute), 444
 URLRequest (class in kivy.network.urlrequest), 442
 use_bubble (kivy.uix.textinput.TextInput attribute), 661
 use_handles (kivy.uix.textinput.TextInput attribute), 661
 use_kivy_settings (kivy.app.App attribute), 183
 use_parent_frag_modelview (kivy.graphics.instructions.RenderContext attribute), 330
 use_parent_frag_modelview (kivy.graphics.RenderContext attribute), 322
 use_parent_modelview (kivy.graphics.instructions.RenderContext attribute), 330
 use_parent_modelview (kivy.graphics.RenderContext attribute), 322
 use_parent_projection (kivy.graphics.instructions.RenderContext attribute), 322

attribute), 330
 use_parent_projection (kivy.graphics.RenderContext attribute), 322
 use_separator (kivy.uix.actionbar.ActionGroup attribute), 507
 use_separator (kivy.uix.actionbar.ActionView attribute), 509
 user_data_dir (kivy.app.App attribute), 183
 usersize (kivy.core.text.LabelBase attribute), 272
 usleep() (kivy.clock.ClockBase method), 201
 usleep() (kivy.clock.ClockBaseBehavior method), 200
 usleep() (kivy.clock.ClockBaseInterruptBehavior method), 200
 uvpos (kivy.graphics.texture.Texture attribute), 366
 uvsize (kivy.graphics.texture.Texture attribute), 366
V
 v (kivy.graphics.Color attribute), 311
 v (kivy.graphics.context_instructions.Color attribute), 333
 valign (kivy.uix.label.Label attribute), 574
 value (kivy.effects.kinetic.KineticEffect attribute), 297
 value (kivy.uix.progressbar.ProgressBar attribute), 585
 value (kivy.uix.settings.SettingItem attribute), 624
 value (kivy.uix.slider.Slider attribute), 631
 value_normalized (kivy.uix.progressbar.ProgressBar attribute), 585
 value_normalized (kivy.uix.slider.Slider attribute), 631
 value_pos (kivy.uix.slider.Slider attribute), 632
 value_track (kivy.uix.slider.Slider attribute), 632
 value_track_color (kivy.uix.slider.Slider attribute), 632
 value_track_width (kivy.uix.slider.Slider attribute), 632
 values (kivy.uix.settings.SettingBoolean attribute), 625
 values (kivy.uix.spinner.Spinner attribute), 634
 VariableListProperty (class in kivy.properties), 248
 vbar (kivy.uix.scrollview.ScrollView attribute), 618
 Vector (class in kivy.vector), 255
 velocity (kivy.effects.kinetic.KineticEffect attribute), 297
 vertex_count (kivy.graphics.tesselator.Tesselator attribute), 359
 VertexInstruction (class in kivy.graphics), 324
 VertexInstruction (class in kivy.graphics.instructions), 328
 VerticalBlurEffect (class in kivy.uix.effectwidget), 540
 vertices (kivy.graphics.Mesh attribute), 320
 vertices (kivy.graphics.tesselator.Tesselator attribute), 360
 vertices (kivy.graphics.vertex_instructions.Mesh attribute), 378
 Video (class in kivy.uix.video), 668
 VideoBase (class in kivy.core.video), 277
 VideoPlayer (class in kivy.uix.videoplayer), 672
 VideoPlayerAnnotation (class in kivy.uix.videoplayer), 674
 view_adapter (kivy.uix.recycleview.RecycleViewBehavior attribute), 492
 view_clip() (kivy.graphics.transformation.Matrix method), 369
 view_list (kivy.uix.filechooser.FileChooser attribute), 546
 view_mode (kivy.uix.filechooser.FileChooser attribute), 546
 viewclass (kivy.uix.recycleview.layout.RecycleLayoutManagerBehavior attribute), 497
 viewclass (kivy.uix.recycleview.RecycleView attribute), 493
 viewport_size (kivy.uix.scrollview.ScrollView attribute), 618
 VKeyboard (class in kivy.uix.vkeyboard), 677
 volume (kivy.core.audio.Sound attribute), 261
 volume (kivy.core.video.VideoBase attribute), 278
 volume (kivy.uix.video.Video attribute), 670
 volume (kivy.uix.videoplayer.VideoPlayer attribute), 674
 vs (kivy.graphics.shader.Shader attribute), 354
 vs (kivy.uix.screenmanager.ShaderTransition attribute), 610
W
 w (kivy.core.text.text_layout.LayoutLine attribute), 275
 wait() (kivy.network.urlrequest.UrlRequest method), 444
 walk() (kivy.uix.widget.Widget method), 691
 walk_reverse() (kivy.uix.widget.Widget method), 692
 weak_callback (kivy.clock.ClockEvent attribute), 197
 WeakMethod (class in kivy.weakmethod), 258
 WeakProxy (class in kivy.weakproxy), 258
 wheel (kivy.uix.colorpicker.ColorPicker attribute), 531
 Widget (class in kivy.uix.widget), 683
 widget (kivy.core.window.Keyboard attribute), 279
 widget (kivy.modules.console.Console attribute), 434
 WidgetException, 693
 width (kivy.core.image.Image attribute), 266

width (kivy.core.image.ImageData attribute), 267
width (kivy.core.window.WindowBase attribute), 291
width (kivy.graphics.Line attribute), 319
width (kivy.graphics.svg.Svg attribute), 357
width (kivy.graphics.texture.Texture attribute), 366
width (kivy.graphics.vertex_instructions.Line attribute), 377
width (kivy.input.shape.ShapeRect attribute), 402
width (kivy.uix.widget.Widget attribute), 693
window (kivy.core.window.Keyboard attribute), 279
window (kivy.input.recorder.Recorder attribute), 397
WindowBase (class in kivy.core.window), 279
WipeTransition (class in kivy.uix.screenmanager), 611
with_previous (kivy.uix.actionbar.ActionPrevious attribute), 510
WM_MotionEvent (class in kivy.input.providers.wm_touch), 393
WM_Pen (class in kivy.input.providers.wm_pen), 393
words (kivy.core.text.text_layout.LayoutLine attribute), 275
wrap (kivy.graphics.texture.Texture attribute), 366
write() (kivy.config.ConfigParser method), 208
write_tab (kivy.uix.textinput.TextInput attribute), 662

X

x (kivy.core.text.text_layout.LayoutLine attribute), 275
x (kivy.graphics.context_instructions.Scale attribute), 335
x (kivy.graphics.context_instructions.Translate attribute), 335
x (kivy.graphics.Scale attribute), 323
x (kivy.graphics.Translate attribute), 324
x (kivy.input.MotionEvent attribute), 384
x (kivy.input.motionEvent.MotionEvent attribute), 401
x (kivy.uix.widget.Widget attribute), 693
x (kivy.vector.Vector attribute), 257
xy (kivy.graphics.context_instructions.Translate attribute), 335
xy (kivy.graphics.Translate attribute), 324
xyz (kivy.graphics.context_instructions.Scale attribute), 335
xyz (kivy.graphics.context_instructions.Translate attribute), 335
xyz (kivy.graphics.Scale attribute), 323
xyz (kivy.graphics.Translate attribute), 324

Y

y (kivy.core.text.text_layout.LayoutLine attribute), 275
y (kivy.graphics.context_instructions.Scale attribute), 335
y (kivy.graphics.context_instructions.Translate attribute), 335
y (kivy.graphics.Scale attribute), 323
y (kivy.graphics.Translate attribute), 324
y (kivy.input.MotionEvent attribute), 384
y (kivy.input.motionEvent.MotionEvent attribute), 401
y (kivy.uix.widget.Widget attribute), 693
y (kivy.vector.Vector attribute), 257

Z

z (kivy.graphics.context_instructions.Scale attribute), 335
z (kivy.graphics.context_instructions.Translate attribute), 335
z (kivy.graphics.Scale attribute), 323
z (kivy.graphics.Translate attribute), 324
z (kivy.input.MotionEvent attribute), 384
z (kivy.input.motionEvent.MotionEvent attribute), 401