



# Kivy Documentation

*Release 1.8.1-dev*

[www.kivy.org](http://www.kivy.org)



# CONTENTS

<b>I</b>	<b>User's Guide</b>	<b>3</b>
<b>1</b>	<b>Installation</b>	<b>5</b>
1.1	Stable Version . . . . .	5
1.2	Development Version . . . . .	22
<b>2</b>	<b>Philosophy</b>	<b>25</b>
2.1	Why bother? . . . . .	25
<b>3</b>	<b>Contributing</b>	<b>27</b>
3.1	Feedback . . . . .	27
3.2	Reporting an Issue . . . . .	27
3.3	Code Contributions . . . . .	28
3.4	Documentation Contributions . . . . .	30
3.5	Unit tests contributions . . . . .	31
3.6	GSOC . . . . .	34
<b>4</b>	<b>FAQ</b>	<b>41</b>
4.1	Technical FAQ . . . . .	41
4.2	Android FAQ . . . . .	42
4.3	Project FAQ . . . . .	43
<b>5</b>	<b>Contact Us</b>	<b>47</b>
5.1	Issue Tracker . . . . .	47
5.2	Mail . . . . .	47
5.3	IRC . . . . .	47
<b>II</b>	<b>Programming Guide</b>	<b>49</b>
<b>6</b>	<b>Kivy Basics</b>	<b>51</b>
6.1	Installation of the Kivy environment . . . . .	51
6.2	Create an application . . . . .	51
6.3	Kivy App Life Cycle . . . . .	52
6.4	Running the application . . . . .	53
6.5	Customize the application . . . . .	54
6.6	Platform specifics . . . . .	55
<b>7</b>	<b>Controlling the environment</b>	<b>57</b>
7.1	Configuration . . . . .	57
7.2	Path control . . . . .	57
7.3	Restrict core to specific implementation . . . . .	57
7.4	Metrics . . . . .	58

<b>8</b>	<b>Configure Kivy</b>	<b>59</b>
8.1	Locating the configuration file	59
8.2	Understanding config tokens	59
<b>9</b>	<b>Architectural Overview</b>	<b>61</b>
9.1	Core Providers and Input Providers	62
9.2	Graphics	62
9.3	Core	62
9.4	UIX (Widgets & Layouts)	63
9.5	Modules	63
9.6	Input Events (Touches)	63
9.7	Widgets and Event Dispatching	64
<b>10</b>	<b>Events and Properties</b>	<b>65</b>
10.1	Introduction to the Event Dispatcher	65
10.2	Main loop	66
10.3	Widget events	67
10.4	Creating custom events	68
10.5	Attaching callbacks	68
10.6	Introduction to Properties	68
10.7	Declaration of a Property	69
10.8	Dispatching a Property event	69
10.9	Compound Properties	72
<b>11</b>	<b>Input management</b>	<b>73</b>
11.1	Input architecture	73
11.2	Motion event profiles	74
11.3	Touch events	74
<b>12</b>	<b>Widgets</b>	<b>77</b>
12.1	Introduction to Widget	77
12.2	Manipulating the Widget tree	77
12.3	Traversing the Tree	78
12.4	Widgets Z Index	78
12.5	Organize with Layouts	78
12.6	Adding a Background to a Layout	86
12.7	Nesting Layouts	95
12.8	Size and position metrics	95
12.9	Screen Separation with Screen Manager	95
<b>13</b>	<b>Graphics</b>	<b>97</b>
13.1	Introduction to Canvas	97
13.2	Context instructions	97
13.3	Drawing instructions	98
13.4	Manipulating instructions	98
<b>14</b>	<b>Kv language</b>	<b>99</b>
14.1	Concept behind the language	99
14.2	How to load KV	99
14.3	Rule context	99
14.4	Special syntaxes	100
14.5	Instantiate children	100
14.6	Event Bindings	101
14.7	Extend canvas	101
14.8	Referencing Widgets	102
14.9	Accessing Widgets defined inside Kv lang in your python code	103
14.10	Dynamic Classes	104

14.11 Re-using styles in multiple widgets	104
14.12 Designing with the Kivy Language	105
<b>15 Integrating with other Frameworks</b>	<b>109</b>
15.1 Using Twisted inside Kivy	109
<b>16 Best Practices</b>	<b>113</b>
16.1 Designing your Application code	113
16.2 Handle Window re-sizing	113
16.3 Managing resources	113
16.4 Platform consideration	113
16.5 Tips and Tricks	113
<b>17 Advanced Graphics</b>	<b>115</b>
17.1 Create your own Shader	115
17.2 Rendering in a Framebuffer	115
17.3 Optimizations	115
<b>18 Packaging your application</b>	<b>117</b>
18.1 Create a package for Windows	117
18.2 Create a package for Android	119
18.3 The Kivy Android Virtual Machine	123
18.4 Kivy on Android	124
18.5 Creating packages for MacOSX	126
18.6 Create a package for IOS	128
18.7 IOS Prerequisites	130
18.8 Compiling for IOS	131
<b>III Tutorials</b>	<b>133</b>
<b>19 Pong Game Tutorial</b>	<b>135</b>
19.1 Introduction	135
19.2 Getting Started	136
19.3 Add Simple Graphics	136
19.4 Add the Ball	138
19.5 Adding Ball Animation	140
19.6 Connect Input Events	143
19.7 Where To Go Now?	146
<b>20 A Simple Paint App</b>	<b>147</b>
20.1 Basic Considerations	147
20.2 Paint Widget	147
<b>IV API Reference</b>	<b>157</b>
<b>21 Kivy framework</b>	<b>159</b>
21.1 Animation	160
21.2 Application	170
21.3 Asynchronous data loader	181
21.4 Atlas	183
21.5 Cache manager	186
21.6 Clock object	187
21.7 Compatibility module for Python 2.7 and > 3.3	192
21.8 Configuration object	192
21.9 Context	197

21.10	Event dispatcher	197
21.11	Factory object	201
21.12	Garden	202
21.13	Geometry utilities	203
21.14	Gesture recognition	203
21.15	Interactive launcher	205
21.16	Kivy Base	207
21.17	Kivy Language	210
21.18	Logger object	221
21.19	Metrics	222
21.20	Multistroke gesture recognizer	224
21.21	Parser utilities	232
21.22	Properties	232
21.23	Resources management	243
21.24	Support	243
21.25	Utils	243
21.26	Vector	246
21.27	Weak Method	249
<b>22</b>	<b>Adapters</b>	<b>251</b>
22.1	Adapter	251
22.2	DictAdapter	252
22.3	List Item View Argument Converters	253
22.4	ListAdapter	253
22.5	SelectableDataItem	256
22.6	SimpleListAdapter	256
<b>23</b>	<b>Adapter</b>	<b>259</b>
<b>24</b>	<b>List Item View Argument Converters</b>	<b>261</b>
<b>25</b>	<b>DictAdapter</b>	<b>263</b>
<b>26</b>	<b>ListAdapter</b>	<b>265</b>
<b>27</b>	<b>SelectableDataItem</b>	<b>269</b>
27.1	Data Models	269
<b>28</b>	<b>SimpleListAdapter</b>	<b>271</b>
<b>29</b>	<b>Animation</b>	<b>273</b>
29.1	Simple animation	273
29.2	Multiple properties and transitions	273
29.3	Sequential animation	273
29.4	Parallel animation	274
29.5	Repeating animation	274
<b>30</b>	<b>Application</b>	<b>285</b>
30.1	Creating an Application	285
30.2	Application configuration	286
30.3	Profiling with on_start and on_stop	289
30.4	Customising layout	289
30.5	Pause mode	290
<b>31</b>	<b>Atlas</b>	<b>297</b>
31.1	Definition of .atlas	297
31.2	How to create an Atlas	297

31.3	How to use an Atlas	298
31.4	Manual usage of the Atlas	299
<b>32</b>	<b>Kivy Base</b>	<b>301</b>
32.1	Event loop management	301
<b>33</b>	<b>Cache manager</b>	<b>305</b>
<b>34</b>	<b>Clock object</b>	<b>307</b>
34.1	Schedule before frame	308
34.2	Triggered Events	308
34.3	Threading	309
<b>35</b>	<b>Compatibility module for Python 2.7 and &gt; 3.3</b>	<b>313</b>
<b>36</b>	<b>Configuration object</b>	<b>315</b>
36.1	Usage of the Config object	315
36.2	Available configuration tokens	315
<b>37</b>	<b>Context</b>	<b>321</b>
<b>38</b>	<b>Core Abstraction</b>	<b>323</b>
38.1	Audio	323
38.2	Camera	325
38.3	Clipboard	325
38.4	OpenGL	326
38.5	Image	326
38.6	Spelling	329
38.7	Text	330
38.8	Video	336
38.9	Window	337
<b>39</b>	<b>Audio</b>	<b>345</b>
<b>40</b>	<b>Camera</b>	<b>347</b>
<b>41</b>	<b>Clipboard</b>	<b>349</b>
<b>42</b>	<b>OpenGL</b>	<b>351</b>
<b>43</b>	<b>Image</b>	<b>353</b>
<b>44</b>	<b>Spelling</b>	<b>357</b>
<b>45</b>	<b>Text</b>	<b>359</b>
45.1	Text Markup	361
45.2	Text layout	362
<b>46</b>	<b>Text Markup</b>	<b>367</b>
<b>47</b>	<b>Text layout</b>	<b>369</b>
<b>48</b>	<b>Video</b>	<b>373</b>
<b>49</b>	<b>Window</b>	<b>375</b>
<b>50</b>	<b>Effects</b>	<b>383</b>
50.1	Damped scroll effect	383
50.2	Kinetic effect	384

50.3	Opacity scroll effect	385
50.4	Scroll effect	385
<b>51</b>	<b>Damped scroll effect</b>	<b>387</b>
<b>52</b>	<b>Kinetic effect</b>	<b>389</b>
<b>53</b>	<b>Opacity scroll effect</b>	<b>391</b>
<b>54</b>	<b>Scroll effect</b>	<b>393</b>
<b>55</b>	<b>Event dispatcher</b>	<b>395</b>
<b>56</b>	<b>Extension Support</b>	<b>401</b>
<b>57</b>	<b>Factory object</b>	<b>403</b>
<b>58</b>	<b>Garden</b>	<b>405</b>
58.1	Packaging	406
<b>59</b>	<b>Geometry utilities</b>	<b>407</b>
<b>60</b>	<b>Gesture recognition</b>	<b>409</b>
<b>61</b>	<b>Graphics</b>	<b>411</b>
61.1	The basics	411
61.2	GL Reloading mechanism	411
61.3	Canvas	429
61.4	Context instructions	434
61.5	Context management	437
61.6	Framebuffer	438
61.7	GL instructions	440
61.8	Graphics compiler	441
61.9	OpenGL	442
61.10	OpenGL utilities	451
61.11	Shader	452
61.12	Stencil instructions	454
61.13	Texture	456
61.14	Transformation	461
61.15	Vertex Instructions	464
<b>62</b>	<b>Graphics compiler</b>	<b>473</b>
62.1	Reducing the context instructions	473
<b>63</b>	<b>Context management</b>	<b>475</b>
<b>64</b>	<b>Context instructions</b>	<b>477</b>
<b>65</b>	<b>Framebuffer</b>	<b>481</b>
65.1	Reloading the FBO content	481
<b>66</b>	<b>GL instructions</b>	<b>485</b>
66.1	Clearing an FBO	485
<b>67</b>	<b>Canvas</b>	<b>487</b>
<b>68</b>	<b>OpenGL</b>	<b>493</b>
<b>69</b>	<b>OpenGL utilities</b>	<b>503</b>



<b>70 Shader</b>	<b>505</b>
70.1 Header inclusion . . . . .	505
70.2 Single file glsl shader programs . . . . .	506
<b>71 Stencil instructions</b>	<b>507</b>
71.1 Limitations . . . . .	507
71.2 Example of stencil usage . . . . .	508
<b>72 Texture</b>	<b>509</b>
72.1 Blitting custom data . . . . .	509
72.2 BGR/BGRA support . . . . .	510
72.3 NPOT texture . . . . .	510
72.4 Texture atlas . . . . .	511
72.5 Mipmapping . . . . .	511
72.6 Reloading the Texture . . . . .	511
<b>73 Transformation</b>	<b>515</b>
<b>74 Input management</b>	<b>519</b>
74.1 Input Postprocessing . . . . .	521
74.2 Providers . . . . .	523
74.3 Input recorder . . . . .	529
74.4 Motion Event . . . . .	531
74.5 Motion Event Factory . . . . .	535
74.6 Motion Event Provider . . . . .	535
74.7 Motion Event Shape . . . . .	536
<b>75 Motion Event Factory</b>	<b>537</b>
<b>76 Motion Event</b>	<b>539</b>
76.1 Motion Event and Touch . . . . .	539
76.2 Listening to a Motion Event . . . . .	539
76.3 Profiles . . . . .	539
<b>77 Input Postprocessing</b>	<b>545</b>
77.1 Calibration . . . . .	545
77.2 Dejitter . . . . .	546
77.3 Double Tap . . . . .	546
77.4 Ignore list . . . . .	546
77.5 Retain Touch . . . . .	547
77.6 Triple Tap . . . . .	547
<b>78 Calibration</b>	<b>549</b>
<b>79 Dejitter</b>	<b>551</b>
<b>80 Double Tap</b>	<b>553</b>
<b>81 Ignore list</b>	<b>555</b>
<b>82 Retain Touch</b>	<b>557</b>
<b>83 Triple Tap</b>	<b>559</b>
<b>84 Motion Event Provider</b>	<b>561</b>
<b>85 Providers</b>	<b>563</b>
85.1 Android Joystick Input Provider . . . . .	563

85.2	Auto Create Input Provider Config Entry for Available MT Hardware (linux only).	563
85.3	Common definitions for a Windows provider	564
85.4	Leap Motion - finger only	564
85.5	Mouse provider implementation	564
85.6	Native support for HID input from the linux kernel	565
85.7	Native support for Multitouch devices on Linux, using libmtdev.	565
85.8	Native support of MultitouchSupport framework for MacBook (MaxOSX platform)	566
85.9	Native support of Wacom tablet from linuxwacom driver	566
85.10	Support for WM_PEN messages (Windows platform)	566
85.11	Support for WM_TOUCH messages (Windows platform)	567
85.12	TUIO Input Provider	567
<b>86</b>	<b>Android Joystick Input Provider</b>	<b>569</b>
<b>87</b>	<b>Native support for HID input from the linux kernel</b>	<b>571</b>
<b>88</b>	<b>Leap Motion - finger only</b>	<b>573</b>
<b>89</b>	<b>Native support of Wacom tablet from linuxwacom driver</b>	<b>575</b>
<b>90</b>	<b>Native support of MultitouchSupport framework for MacBook (MaxOSX platform)</b>	<b>577</b>
<b>91</b>	<b>Mouse provider implementation</b>	<b>579</b>
91.1	Using multitouch interaction with the mouse	579
<b>92</b>	<b>Native support for Multitouch devices on Linux, using libmtdev.</b>	<b>581</b>
<b>93</b>	<b>Auto Create Input Provider Config Entry for Available MT Hardware (linux only).</b>	<b>583</b>
<b>94</b>	<b>TUIO Input Provider</b>	<b>585</b>
94.1	Configure a TUIO provider in the config.ini	585
94.2	Configure a TUIO provider in the App	585
<b>95</b>	<b>Common definitions for a Windows provider</b>	<b>587</b>
<b>96</b>	<b>Support for WM_PEN messages (Windows platform)</b>	<b>589</b>
<b>97</b>	<b>Support for WM_TOUCH messages (Windows platform)</b>	<b>591</b>
<b>98</b>	<b>Input recorder</b>	<b>593</b>
98.1	Recording events	593
98.2	Manual play	593
98.3	Recording more attributes	594
98.4	Known limitations	594
<b>99</b>	<b>Motion Event Shape</b>	<b>597</b>
<b>100</b>	<b>Interactive launcher</b>	<b>599</b>
100.1	Creating an InteractiveLauncher	599
100.2	Interactive Development	599
100.3	Directly Pausing the Application	600
100.4	Adding Attributes Dynamically	600
<b>101</b>	<b>Kivy Language</b>	<b>603</b>
101.1	Overview	603
101.2	Syntax of a kv File	603
101.3	Value Expressions, on_property Expressions, and Reserved Keywords	605
101.4	Relation Between Values and Properties	606
101.5	Graphical Instructions	607

101.6 Dynamic classes . . . . .	607
101.7 Templates . . . . .	609
101.8 Redefining a widget's style . . . . .	611
101.9 Lang Directives . . . . .	611
<b>102 External libraries</b>	<b>617</b>
102.1 GstPlayer . . . . .	617
<b>103 GstPlayer</b>	<b>619</b>
<b>104 Asynchronous data loader</b>	<b>621</b>
104.1 Tweaking the asynchronous loader . . . . .	621
<b>105 Logger object</b>	<b>625</b>
105.1 Logger configuration . . . . .	625
105.2 Logger history . . . . .	625
<b>106 Metrics</b>	<b>627</b>
106.1 Dimensions . . . . .	627
106.2 Examples . . . . .	627
106.3 Manual control of metrics . . . . .	628
<b>107 Modules</b>	<b>631</b>
107.1 Activating a module . . . . .	631
107.2 Create your own module . . . . .	632
107.3 Inspector . . . . .	632
107.4 Keybinding . . . . .	633
107.5 Monitor module . . . . .	634
107.6 Recorder module . . . . .	634
107.7 Screen . . . . .	635
107.8 Touchring . . . . .	635
107.9 Web Debugger . . . . .	636
<b>108 Inspector</b>	<b>637</b>
108.1 Usage . . . . .	637
<b>109 Keybinding</b>	<b>639</b>
109.1 Usage . . . . .	639
<b>110 Monitor module</b>	<b>641</b>
110.1 Usage . . . . .	641
<b>111 Recorder module</b>	<b>643</b>
111.1 Configuration . . . . .	643
111.2 Usage . . . . .	643
<b>112 Screen</b>	<b>645</b>
<b>113 Touchring</b>	<b>647</b>
113.1 Configuration . . . . .	647
113.2 Example . . . . .	647
<b>114 Web Debugger</b>	<b>649</b>
<b>115 Multistroke gesture recognizer</b>	<b>651</b>
115.1 Conceptual Overview . . . . .	651
115.2 Usage examples . . . . .	651
115.3 Algorithm details . . . . .	652

<b>116Network support</b>	<b>661</b>
116.1 Url Request . . . . .	661
<b>117Url Request</b>	<b>665</b>
<b>118Parser utilities</b>	<b>669</b>
<b>119Properties</b>	<b>671</b>
119.1 Comparison Python vs. Kivy . . . . .	671
119.2 Observe Properties changes . . . . .	672
119.3 Binding to properties of properties. . . . .	673
<b>120Resources management</b>	<b>683</b>
<b>121Storage</b>	<b>685</b>
121.1 Usage . . . . .	685
121.2 Examples . . . . .	685
121.3 Synchronous / Asynchronous API . . . . .	686
121.4 Synchronous container type . . . . .	686
121.5 Dictionary store . . . . .	688
121.6 JSON store . . . . .	688
121.7 Redis Store . . . . .	689
<b>122Dictionary store</b>	<b>691</b>
<b>123JSON store</b>	<b>693</b>
<b>124Redis Store</b>	<b>695</b>
<b>125Support</b>	<b>697</b>
<b>126Widgets</b>	<b>699</b>
126.1 Abstract View . . . . .	699
126.2 Accordion . . . . .	700
126.3 Action Bar . . . . .	704
126.4 Anchor Layout . . . . .	708
126.5 Behaviors . . . . .	709
126.6 Box Layout . . . . .	716
126.7 Bubble . . . . .	717
126.8 Button . . . . .	721
126.9 Camera . . . . .	722
126.10Carousel . . . . .	723
126.11CheckBox . . . . .	725
126.12Code Input . . . . .	726
126.13Color Picker . . . . .	727
126.14Drop-Down List . . . . .	729
126.15EffectWidget . . . . .	731
126.16FileChooser . . . . .	735
126.17Float Layout . . . . .	740
126.18Gesture Surface . . . . .	741
126.19Grid Layout . . . . .	745
126.20Image . . . . .	748
126.21Label . . . . .	751
126.22Layout . . . . .	758
126.23List View . . . . .	759
126.24ModalView . . . . .	769
126.25PageLayout . . . . .	771

126.26	Popup	772
126.27	Progress Bar	774
126.28	Relative Layout	775
126.29	Sandbox	777
126.30	Scatter	778
126.31	Scatter Layout	781
126.32	Screen Manager	782
126.33	Scroll View	789
126.34	Settings	793
126.35	Slider	800
126.36	Spinner	802
126.37	Splitter	804
126.38	Stack Layout	806
126.39	Stencil View	807
126.40	Switch	808
126.41	TabbedPanel	809
126.42	Text Input	815
126.43	Toggle button	825
126.44	Tree View	825
126.45	VKeyboard	831
126.46	Video	835
126.47	Video player	836
126.48	Widget class	841
126.49	reStructuredText renderer	850
<b>127</b>	<b>Abstract View</b>	<b>855</b>
<b>128</b>	<b>Accordion</b>	<b>857</b>
128.1	Simple example	858
128.2	Customize the accordion	858
<b>129</b>	<b>Action Bar</b>	<b>863</b>
<b>130</b>	<b>Anchor Layout</b>	<b>867</b>
<b>131</b>	<b>Behaviors</b>	<b>869</b>
<b>132</b>	<b>Box Layout</b>	<b>877</b>
<b>133</b>	<b>Bubble</b>	<b>879</b>
133.1	Simple example	879
133.2	Customize the Bubble	880
<b>134</b>	<b>Button</b>	<b>883</b>
<b>135</b>	<b>Camera</b>	<b>885</b>
<b>136</b>	<b>Carousel</b>	<b>887</b>
<b>137</b>	<b>CheckBox</b>	<b>891</b>
<b>138</b>	<b>Code Input</b>	<b>893</b>
138.1	Usage example	893
<b>139</b>	<b>Color Picker</b>	<b>895</b>
<b>140</b>	<b>Drop-Down List</b>	<b>897</b>
140.1	Basic example	897

140.2 Extending dropdown in Kv . . . . .	898
<b>141EffectWidget</b>	<b>901</b>
141.1 Provided Effects . . . . .	901
141.2 Creating Effects . . . . .	902
<b>142FileChooser</b>	<b>905</b>
142.1 Simple example . . . . .	905
<b>143Float Layout</b>	<b>911</b>
<b>144Gesture Surface</b>	<b>913</b>
<b>145Grid Layout</b>	<b>917</b>
145.1 Background . . . . .	917
145.2 Column Width and Row Height . . . . .	918
145.3 Using a GridLayout . . . . .	918
<b>146Image</b>	<b>923</b>
146.1 Asynchronous Loading . . . . .	923
146.2 Alignment . . . . .	923
<b>147Label</b>	<b>927</b>
147.1 Markup text . . . . .	927
147.2 Interactive Zone in Text . . . . .	928
147.3 Usage example . . . . .	928
<b>148Layout</b>	<b>935</b>
148.1 Understanding the <i>size_hint</i> Property in <i>Widget</i> . . . . .	935
<b>149List View</b>	<b>937</b>
149.1 Introduction . . . . .	937
149.2 Basic Example . . . . .	938
149.3 Using an Adapter . . . . .	939
149.4 ListAdapter and DictAdapter . . . . .	940
149.5 Using an Args Converter . . . . .	941
149.6 An Example ListView . . . . .	942
149.7 Using a Custom Item View Class . . . . .	942
149.8 Using an Item View Template . . . . .	943
149.9 Using CompositeListItem . . . . .	944
149.10Uses for Selection . . . . .	945
<b>150ModalView</b>	<b>949</b>
150.1 Examples . . . . .	949
150.2 ModalView Events . . . . .	950
<b>151PageLayout</b>	<b>953</b>
<b>152Popup</b>	<b>955</b>
152.1 Examples . . . . .	955
152.2 Popup Events . . . . .	956
<b>153Progress Bar</b>	<b>959</b>
<b>154Relative Layout</b>	<b>961</b>
154.1 Coordinate Systems . . . . .	961

<b>155reStructuredText renderer</b>	<b>965</b>
155.1 Usage with Text . . . . .	965
155.2 Usage with Source . . . . .	966
<b>156Sandbox</b>	<b>969</b>
<b>157Scatter</b>	<b>971</b>
157.1 Usage . . . . .	971
157.2 Control Interactions . . . . .	971
157.3 Automatic Bring to Front . . . . .	972
157.4 Scale Limitation . . . . .	972
157.5 Behavior . . . . .	972
<b>158Scatter Layout</b>	<b>975</b>
<b>159Screen Manager</b>	<b>977</b>
159.1 Basic Usage . . . . .	977
159.2 Changing transitions . . . . .	978
<b>160Scroll View</b>	<b>985</b>
160.1 Scrolling Behavior . . . . .	985
160.2 Limiting to the X or Y Axis . . . . .	985
160.3 Managing the Content Size and Position . . . . .	985
160.4 Overscroll Effects . . . . .	986
<b>161Settings</b>	<b>991</b>
161.1 Create a panel from JSON . . . . .	992
161.2 Different panel layouts . . . . .	993
<b>162Slider</b>	<b>999</b>
<b>163Spinner</b>	<b>1003</b>
<b>164Splitter</b>	<b>1005</b>
<b>165Stack Layout</b>	<b>1007</b>
<b>166Stencil View</b>	<b>1011</b>
<b>167Switch</b>	<b>1013</b>
<b>168TabbedPanel</b>	<b>1015</b>
168.1 Simple example . . . . .	1015
168.2 Customize the Tabbed Panel . . . . .	1016
<b>169Text Input</b>	<b>1023</b>
169.1 Usage example . . . . .	1023
169.2 Selection . . . . .	1024
169.3 Filtering . . . . .	1024
169.4 Default shortcuts . . . . .	1025
<b>170Toggle button</b>	<b>1035</b>
<b>171Tree View</b>	<b>1037</b>
171.1 Introduction . . . . .	1037
171.2 Creating Your Own Node Widget . . . . .	1038
<b>172Video</b>	<b>1043</b>

<b>173Video player</b>	<b>1045</b>
173.1 Annotations	1045
173.2 Fullscreen	1046
173.3 End-of-stream behavior	1046
<b>174VKeyboard</b>	<b>1051</b>
174.1 Modes	1051
174.2 Layouts	1051
174.3 Request Keyboard	1052
<b>175Widget class</b>	<b>1057</b>
175.1 Using Properties	1057
175.2 Basic drawing	1058
175.3 Widget touch event bubbling	1058
<b>176Utils</b>	<b>1067</b>
<b>177Vector</b>	<b>1071</b>
177.1 Optimized usage	1071
177.2 Vector operators	1072
<b>178Weak Method</b>	<b>1075</b>
<b>V Appendix</b>	<b>1077</b>
<b>179License</b>	<b>1079</b>
<b>Python Module Index</b>	<b>1081</b>
<b>Index</b>	<b>1083</b>



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

## USER'S GUIDE

This part of the documentation explains the basic ideas behind Kivy's design and why you'd want to use it. It goes on with a discussion of the architecture and shows you how to create stunning applications in a short time using the framework.



---

# INSTALLATION

We try not to reinvent the wheel, but to bring something innovative to the market. As a consequence, we're focused on our own code and use pre-existing, high-quality third-party libraries where possible. To support the full, rich set of features that Kivy offers, several other libraries are required. If you do not use a specific feature (e.g. video playback), you don't need the corresponding dependency. That said, there is one dependency that Kivy **does** require: **Cython**.

In addition, you need a **Python** 2.x ( $2.7 \leq x < 3.0$ ) or 3.x ( $3.3 \leq x$ ) interpreter. If you want to enable features like windowing (i.e. open a Window), audio/video playback or spelling correction, additional dependencies must be available. For these, we recommend **Pygame**, **Gst-Python** and **Enchant**, respectively.

NOTE: Currently, packaging only works with Python 2.7. We are working on Python 3.3+ support, but for now if you plan to distribute your package on any platform you should use Python 2.7.

Other optional libraries (mutually independent) are:

- **OpenCV 2.0** – Camera input.
- **PIL** – Image and text display.
- **PyCairo** – Text display.
- **PyEnchant** – Spelling correction.
- **PyGST** – Audio/video playback and camera input.

That said, **DON'T PANIC!**

We don't expect you to install all those things on your own. Instead, we have created nice portable packages that you can use directly, and they already contain the necessary packages for your platform. We just want you to know that there are alternatives to the defaults and give you an overview of the things Kivy uses internally.

## 1.1 Stable Version

The latest stable version can be found on Kivy's website at <http://kivy.org/#download>. Please refer to the installation instructions for your specific platform:




### 1.1.1 Installation on Windows

For Windows, we provide what we call a 'portable package'. This is the easiest way to get Kivy running as you don't have to install anything "system" wide. You can just unzip & run it.

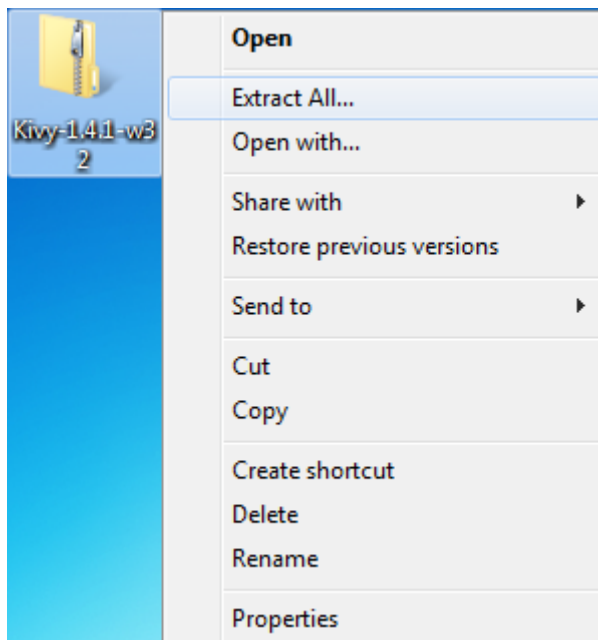
This installation method is simple because it bundles the Python interpreter together with the Kivy environment and libraries. If you wish to install Kivy into an existing Python environment or install the development environment, please see the *Other Environments* section below.

## Installing the portable version

1. Download the latest version from <http://kivy.org/#download>

Operating System	File	Instructions	Size
 Windows Seven (32/64 bits)	<a href="#">Kivy-1.1.1-w32.zip</a>	<a href="#">Installation for Windows</a>	72.9 Mb
 Mac OS X 10.6	<a href="#">Kivy-1.1.1-osx.dmg</a>	<a href="#">Installation for MacOSX</a>	43.7 Mb
 Linux (Ubuntu 11.04, 11.10)	<a href="#">Kivy-1.1.1.tar.gz</a>	<a href="#">Installation for Ubuntu</a>	7 Mb

2. Unzip the package



3. In the folder where you unzipped the package, you have a script called *kivy.bat*. Use this file for launching any kivy application as described below.

---

**Note:** Launching the *kivy.bat* file will open a command window already set up to run kivy's Python. The environment settings are only changed for this command window and will not effect the system environment.

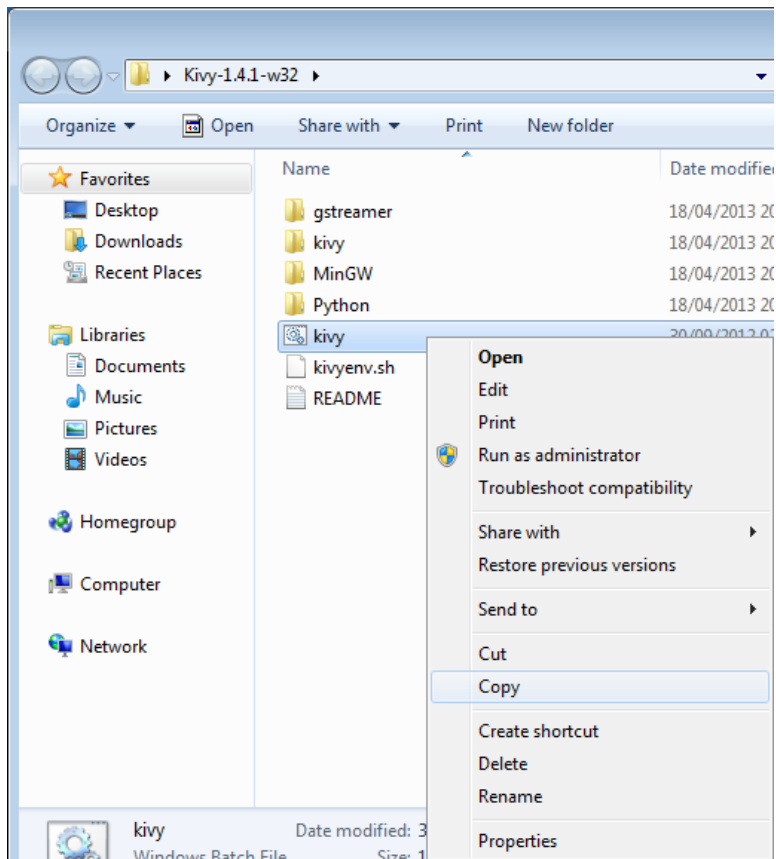
---

## Start a Kivy Application

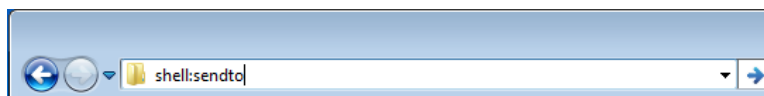
### Send-to method

You can launch a *.py* file with our Python using the Send-to menu:

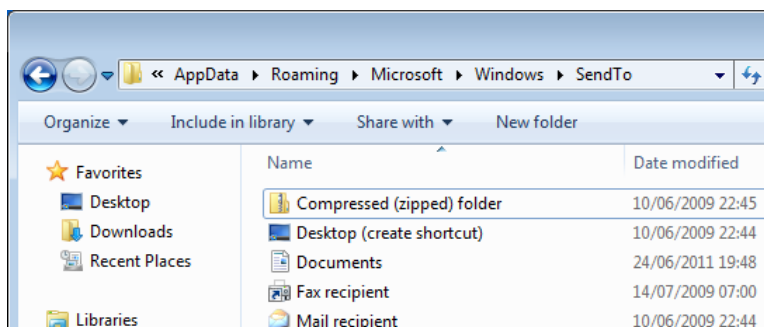
1. Copy the *kivy.bat* file to the Clipboard



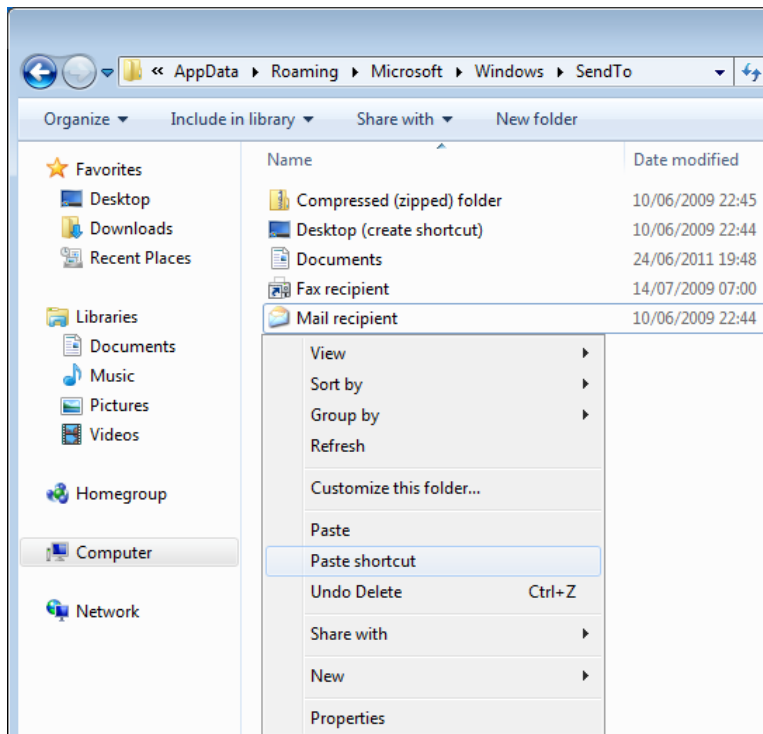
2. Open Windows explorer (File explorer in Windows 8), and to go the address 'shell:sendto'



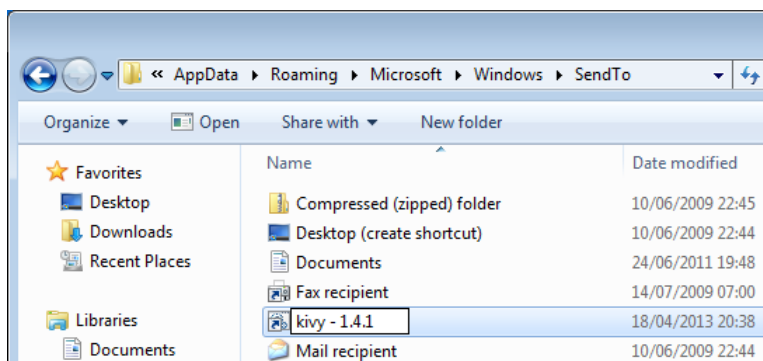
3. You should get the special Windows directory *SendTo*



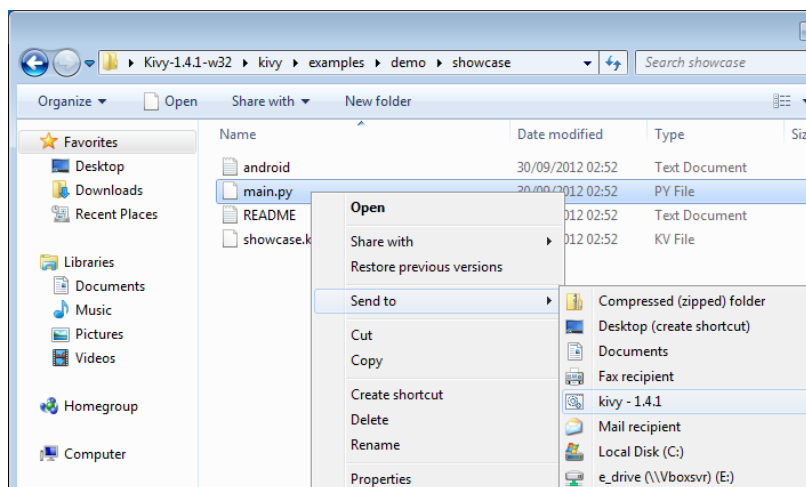
4. Paste the previously copied kivy.bat file **as a shortcut**



5. Rename it to Kivy <kivy-version>



You can now execute your application by right clicking on the .py file -> "Send To" -> "Kivy <version>".





## Double-click method

There are some simple steps that you need to complete in order to be able to launch Kivy applications by just double-clicking them:

1. Right click on the main Python file (.py file extension) of the application you want to launch
2. From the context menu that appears, select *Open With*
3. Browse your hard disk drive and find the file `kivy.bat` from the portable package. Select it.
4. Select “Always open the file with...” if you don’t want to repeat this procedure every time you double click a .py file.
5. You are done. Open the file.

The next time you double click a .py file, it will be executed with the version of Python that Kivy ships with.

---

**Note:** On Windows we have to ship our own version of Python since it’s not installed by default on Windows (unlike Mac OS X and Linux). By following the steps above, you will set Kivy’s version of Python as the default for opening .py files for your user. Normally this should not be harmful as it’s just a normal version of Python with the *necessary third party libraries* added to the module search path. If you do encounter unexpected problems, please [Contact Us](#).

---

## Start from the Command-line (using bash)

If you just want to use or develop with the latest stable Kivy version, this can be achieved using the console. You will need a minimalist GNU system installed. We recommend [msysGit](#).

When you install msysGit, you must select these options:

- Don’t replace windows shell
- Checkout as-is, commit as-is (no CLRF replacement!)

You’ll have an icon “Git bash” on your desktop. This is the console we want:

1. Start “Git bash”
2. `cd <directory of portable kivy>`
3. `source kivyenv.sh <full directory path of portable kivy> # (don’t use .)`

You are now ready to launch Python/Kivy from the command-line! Just do:

```
python <filename.py>
```

Also, all other scripts and binaries are available, such as:

- cython
- gcc / make...
- easy\_install
- gst-inspect-0.10

## Start from the Command-line or Double-click (using Python launcher for Windows)

The Python launcher for Windows is available as a separate download from [pylauncher](#), but is most conveniently installed by simply installing Python 3.3 (or later). Don’t worry, this installation is designed to cause minimum disruption, it will run your latest Python 2 by default.

The launcher defines a PY command which can launch scripts for any version of Python installed on the workstation. It also connects itself as the default processor for all files with a .py extension. It scans the Python file to see if the first line starts with the string “#!” and, if it does, uses that string to select the appropriate version of Python to run. We will define a customized command so that we can tell it to start the correct version of python for Kivy.

Create a file named `py.ini` and place it either in your users application data directory, or in `C:\Windows`. It will contain the path used to start Kivy. I put my Kivy installation at `C:\utils\kivy` so my copy says:

```
[commands]
kivy="c:\utils\kivy\kivy.bat"
```

(You could also add commands to start other script interpreters, such as jython or IronPython.)

Now add a new first line to your `main.py` specifying your Python of choice:

```
#!/usr/bin/kivy
```

You can now launch your Kivy (or any other Python script) either by double-clicking or typing:

```
py <filename.py>
```

Programs without a `#!` first line will continue to be run by the default Python version 2 interpreter. Programs beginning with `#!/usr/bin/python3` will launch Python 3.

The `/usr/bin` part will be ignored by the Windows launcher, we add it so that Linux users will also be able to pick a specific Python version. (On my Linux workstation, `/usr/bin/kivy` is soft-linked to a virtualenv.) NOTE: In order to work correctly on Linux, your Python file must be saved with Unix-style (LF-only) line endings.

Full documentation can be found at: [Python3.3 docs](#) and [PEP 397](#).

Use development Kivy

**Warning:** Using the latest development version can be risky and you might encounter issues during development. If you encounter any bugs, please report them.

If you want to use the latest development version of Kivy, you can follow these steps:

1. Download and install Kivy for Windows as explained above
2. Go into the portable Kivy directory. This contains the `kivy.bat` file and the `Python`, `kivy`, `Mingw` folders etc.
3. Rename the `kivy` directory to `kivy.stable`
4. [Download the latest development version of Kivy from GitHub](#)
5. Extract the zip into the Kivy portable directory
6. Rename the directory named “`kivy-<some hash>`” to just “`kivy`”
7. Launch `kivy.bat`
8. Go to the Kivy portable directory/`kivy`
9. Type:

```
make force
```

10. That’s all, you have a latest development version!

---

**Note:** If you get errors you may need to upgrade Cython:

1. Launch kivy.bat
  2. `pip install --upgrade cython`
- 

Other Environments

[Using Kivy with an existing Python installation.](#)

[Creating a 64 bit development environment with MinGW.](#)

Package Contents

The latest Windows package contains:

- Latest stable kivy version
- Python 2.7.1
- Glew 1.5.7
- Pygame 1.9.2
- Cython 0.14
- MinGW
- GStreamer
- Setuptools

### 1.1.2 Installation on MacOSX

---

**Note:** This method has only been tested on Mac OSX 10.7 Lion 64-bit. For versions prior to 10.7 or 10.7 32-bit, you have to install the components yourself. We suggest using [homebrew](#) to do that.

There can be a limitation on some OS X with more than one monitor. The application will crash when you try to start it on the second monitor.

---

For Mac OS X 10.7 and later, we provide a Kivy.app with all dependencies bundled. Download it from our [Download Page](#). It comes as a .dmg file that contains:

- Kivy.app
- Readme.txt
- An Examples folder
- A script to install a *kivy* command for shell usage

To install Kivy, you must:

1. Download the latest version from <http://kivy.org/#download>
2. Double-click to open it
3. Drag the Kivy.app into your Applications folder
4. Make sure to read the Readme.txt

## Installing the dev version

Step 1. Follow the procedure mentioned above to install kivy stable. step 2 Open a terminal and type the following commands into it:

```
cd /Applications/Kivy.app/Contents/Resources/  
mv kivy kivy_stable  
git clone http://github.com/kivy/kivy  
cd kivy  
make
```

That's it. You now have the latest kivy from github.

## Start any Kivy Application

You can run any Kivy application by simply dragging the application's main file onto the Kivy.app icon. Just try this with any python file in the examples folder.

## Start from the Command Line

If you want to use Kivy from the command line, double-click the **Make Symlinks** script after you have dragged the Kivy.app into the Applications folder. To test if it worked:

1. Open Terminal.app and enter:

```
$ kivy
```

You should get a Python prompt.

2. In there, type:

```
>>> import kivy
```

If it just goes to the next line without errors, it worked.

3. Running any Kivy application from the command line is now simply a matter of executing a command like the following:

```
$ kivy yourapplication.py
```

## 1.1.3 Installation on Linux

### Using software packages

For installing distribution relative packages .deb/.rpm/...

### Ubuntu / Kubuntu / Xubuntu / Lubuntu (Saucy and above)

0. In case you want to use Python3, add this Pygame PPA before

```
$ sudo add-apt-repository ppa:thopiekar/pygame
```

**\*\* These Pygame packages are neither provided nor supported by the Kivy project. \*\***  
Please contact the creator of the package(s) or maintainer of the sourcecode for further help.

1. Add one of the PPAs as you prefer

**stable builds** \$ sudo add-apt-repository ppa:kivy-team/kivy

**nightly builds** \$ sudo add-apt-repository ppa:kivy-team/kivy-daily

2. Update your packagelist using your package manager

3. Install Kivy

**Python2 - python-kivy** \$ sudo apt-get install python-kivy

**Python3 - python3-kivy** \$ sudo apt-get install python3-kivy

**optionally the examples - kivy-examples** \$ sudo apt-get install kivy-examples

## Debian (Jessie or newer)

1. Add one of the PPAs to your sources.list in apt manually or via Synaptic

- Jessie/Testing:

**stable builds** deb <http://ppa.launchpad.net/kivy-team/kivy/ubuntu>  
trusty main

**daily builds** deb <http://ppa.launchpad.net/kivy-team/kivy-daily/ubuntu>  
trusty main

- Sid/Unstable:

**stable builds** deb <http://ppa.launchpad.net/kivy-team/kivy/ubuntu>  
utopic main

**daily builds** deb <http://ppa.launchpad.net/kivy-team/kivy-daily/ubuntu>  
utopic main

**Notice:** Wheezy is not supported - You'll need to upgrade to Jessie at least!

2. Add the GPG key to your apt keyring by executing

as user:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
A863D2D6
```

as root:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
A863D2D6
```

3. Refresh your package list and install **python-kivy** and/or **python3-kivy** and optionally the examples found in **kivy-examples**

## Linux Mint

1. Find out on which Ubuntu release your installation is based on, using this [overview](#).

2. Continue as described for Ubuntu above, depending on which version your installation is based on.

## Bodhi Linux

1. Find out which version of the distribution you are running and use the table below to find out on which Ubuntu LTS it is based.

**Bodhi 1** Ubuntu 10.04 LTS aka Lucid (No packages, just manual install)

**Bodhi 2** Ubuntu 12.04 LTS aka Precise

**Bodhi 3** Ubuntu 14.04 LTS aka Trusty

2. Continue as described for Ubuntu above, depending on which version your installation is based on.

## OpenSuSE

1. Installing via One-Click-Install

- (a) [OpenSuSE Factory](#)
- (b) [OpenSuSE 12.2](#)
- (c) [OpenSuSE 12.1](#)
- (d) [OpenSuSE Tumbleweed](#)

2. If you would like access to the examples, use your preferred package-manager to install **python-Kivy-examples**

## Fedora

1. Adding the repository via the terminal:

### Fedora 18

```
$ sudo yum-config-manager --add-repo=http://download.opensuse.org/repositories/home:/thor
```

### Fedora 17

```
$ sudo yum-config-manager --add-repo=http://download.opensuse.org/repositories/home:/thor
```

### Fedora 16

```
$ sudo yum-config-manager --add-repo=http://download.opensuse.org/repositories/home:/thor
```

2. Use your preferred package-manager to refresh your packagelists
3. Install **python-Kivy** and optionally the examples, as found in **python-Kivy-examples**

## Gentoo

1. There is a kivy ebuild (kivy stable version)

emerge Kivy

2. available USE-flags are:

*cairo: Standard flag, let kivy use cairo graphical libraries. camera: Install libraries needed to support camera. doc: Standard flag, will make you build the documentation locally. examples: Standard flag, will give you kivy examples programs. garden: Install garden tool to manage user maintained widgets. gstreamer: Standard flag, kivy will be able to use audio/video streaming libraries. spell: Standard flag, provide enchant to use spelling in kivy apps.*

## 1.1.4 Using software bundles ( also known as tarballs )

*Providing dependencies*

### General

The following software is needed, even if your distribution is not listed above:

- Python  $\geq 2.7$  and Python  $< 3$
- PyGame
- PyEnchant
- gst-python
- Cython  $\geq 0.15$

We prefer to use a package-manager to provide these dependencies.

### Ubuntu

```
$ sudo apt-get install pkg-config python-setuptools python-pygame python-opengl \
python-gst0.10 python-enchant gstreamer0.10-plugins-good python-dev \
build-essential libgl1-mesa-dev libgles2-mesa-dev cython
```

*Upgrade Cython (  $\leq$  Oneiric [11.10] )*

#### Using Cython's daily PPA

```
$ sudo add-apt-repository ppa:cython-dev/master-ppa
$ sudo apt-get update
$ sudo apt-get install cython
```

#### Using PIP

```
$ sudo apt-get install python-pip
$ sudo pip install --upgrade cython
```

### Fedora

```
$ sudo yum install python-distutils-extra python-enchant freeglut PyOpenGL \
SDL_ttf-devel SDL_mixer-devel pygame pygame-devel khrplatform-devel \
mesa-libGL mesa-libGL-devel gstreamer-plugins-good gstreamer \
gstreamer-python mtdev-devel python-pip
$ sudo pip install --upgrade cython
$ sudo pip install pygments
```

### OpenSuse

```
$ sudo zypper install python-distutils-extra python-pygame python-opengl \
python-gstreamer-0_10 python-enchant gstreamer-0_10-plugins-good \
python-devel Mesa-devel python-pip
$ zypper install -t pattern devel_C_C++
$ sudo pip install --upgrade cython
$ sudo pip install pygments
```

## Mageia 1 onwards

```
$ su
# urpmi python-setuptools python-pygame python-opengl \
gstreamer0.10-python python-enchant gstreamer0.10-plugins-good \
python-cython lib64python-devel lib64mesaglib-devel lib64mesaegl-devel \
lib64mesaglesv2_2-devel make gcc
# easy_install pip
# pip install --upgrade cython
# pip install pygame
```

### 1.1.5 Installation in a Virtual Environment with System Site Packages

This is a recommended compromise between installing Kivy and its dependencies system wide and installing as much as possible into a virtual environment.

Ubuntu 12.04 with Python 2.7

### Install System-wide Dependencies

Note that these commands will remove any pre-existing versions of python-virtualenv and replace it with the current version. It will also remove cython, numpy, and pygame installed from your Linux distro's repository and replace them with current versions from pip or the pygame Mercurial repository.

```
# Install necessary system packages
sudo apt-get install -y build-essential mercurial git python2.7 \
python-setuptools python-dev ffmpeg libsdl-image1.2-dev \
libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev libsdl1.2-dev \
libportmidi-dev libswscale-dev libavformat-dev libavcodec-dev zlib1g-dev

# Bootstrap a current Python environment
sudo apt-get remove --purge -y python-virtualenv python-pip
sudo easy_install-2.7 -U pip
sudo pip2.7 install -U virtualenv

# Install current version of Cython
sudo apt-get remove --purge -y cython
sudo pip2.7 install -U cython

# Install other PyGame dependencies
sudo apt-get remove --purge -y python-numpy
sudo pip2.7 install -U numpy

# Install PyGame
sudo apt-get remove --purge python-pygame
hg clone https://bitbucket.org/pygame/pygame
cd pygame
python2.7 setup.py build
sudo python2.7 setup.py install
cd ..
sudo rm -rf pygame
```



## Create a Kivy Virtualenv

```
# Create a virtualenv
rm -rf venv
virtualenv -p python2.7 --system-site-packages venv

# Install stable version of Kivy into the virtualenv
venv/bin/pip install kivy
# For the development version of Kivy, use the following command instead
# venv/bin/pip install git+https://github.com/kivy/kivy.git@master

# Install development version of buildozer into the virtualenv
venv/bin/pip install git+https://github.com/kivy/buildozer.git@master

# Install development version of pleyer into the virtualenv
venv/bin/pip install git+https://github.com/kivy/plyer.git@master

# Install a couple of dependencies for KivyCatalog
venv/bin/pip install -U pygments docutils
```

## Ubuntu 12.04 with Python 3.3

### Install System-wide Dependencies

Note that these commands will remove any pre-existing versions of python-virtualenv and replace it with the current version. It will also remove cython, numpy, and pygame installed from your Linux distro's repository and replace them with current versions from pip or the pygame Mercurial repository.

```
# Bootstrap Python3.3
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:fkruell/deadsnakes
sudo apt-get update

# Install necessary system packages
sudo apt-get install -y build-essential mercurial git python3.3 \
python3.3-dev ffmpeg libsdl-image1.2-dev libsdl-mixer1.2-dev \
libsdl-ttf2.0-dev libsmpeg-dev libsdl1.2-dev libportmidi-dev \
libswscale-dev libavformat-dev libavcodec-dev zlib1g-dev

# Bootstrap current setuptools
wget https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py -O - | sudo python3.3

# Bootstrap a current Python environment
sudo apt-get remove --purge -y python-virtualenv python-pip
sudo easy_install-3.3 -U pip
sudo pip3.3 install -U virtualenv

# Install current version of Cython
sudo apt-get remove --purge -y cython
sudo pip3.3 install -U cython

# Install other PyGame dependencies
sudo apt-get remove --purge -y python-numpy
sudo pip3.3 install -U numpy

# Install PyGame
hg clone https://bitbucket.org/pygame/pygame
cd pygame
```

```
python3.3 setup.py build
sudo python3.3 setup.py install
cd ..
sudo rm -rf pygame
```

### Create a Kivy Virtualenv

```
# Create a virtualenv
rm -rf venv
virtualenv -p python3.3 --system-site-packages venv

# Install stable version of Kivy into the virtualenv
venv/bin/pip install kivy
# For the development version of Kivy, use the following command instead
# venv/bin/pip install git+https://github.com/kivy/kivy.git@master

# Install development version of buildozer into the virtualenv
#venv/bin/pip install git+https://github.com/kivy/buildozer.git@master

# Install development version of pleyer into the virtualenv
venv/bin/pip install git+https://github.com/kivy/plyer.git@master

# Install a couple of dependencies for KivyCatalog
venv/bin/pip install -U pygments docutils
```

### Start from the Command Line

We ship some examples that are ready-to-run. However, these examples are packaged inside the package. This means you must first know where `easy_install` has installed your current kivy package, and then go to the examples directory:

```
$ python -c "import pkg_resources; print(pkg_resources.resource_filename('kivy', '../share/kivy-e
```

And you should have a path similar to:

```
/usr/local/lib/python2.6/dist-packages/Kivy-1.0.4_beta-py2.6-linux-x86_64.egg/share/kivy-examples
```

Then you can go to the example directory, and run it:

```
# launch touchtracer
$ cd <path to kivy-examples>
$ cd demo/touchtracer
$ python main.py

# launch pictures
$ cd <path to kivy-examples>
$ cd demo/pictures
$ python main.py
```

If you are familiar with Unix and symbolic links, you can create a link directly in your home directory for easier access. For example:

1. Get the example path from the command line above
2. Paste into your console:

```
$ ln -s <path to kivy-examples> ~/
```

3. Then, you can access to kivy-examples directly in your home directory:

```
$ cd ~/kivy-examples
```

If you wish to start your Kivy programs as scripts (by typing `./main.py`) or by double-clicking them, you will want to define the correct version of Python by linking to it. Something like:

```
$ sudo ln -s /usr/bin/python2.7 /usr/bin/kivy
```

Or, if you are running Kivy inside a virtualenv, link to the Python interpreter for it, like:

```
$ sudo ln -s /home/your_username/Envs/kivy/bin/python2.7 /usr/bin/kivy
```

Then, inside each `main.py`, add a new first line:

```
#!/usr/bin/kivy
```

NOTE: Beware of Python files stored with Windows-style line endings (CR-LF). Linux will not ignore the <CR> and will try to use it as part of the file name. This makes confusing error messages. Convert to Unix line endings.

### 1.1.6 Installation on Android

Kivy is a Python framework, and simply installing it on an Android device the same way as on a desktop machine will do nothing. However, you can compile a Kivy application to a standard Android APK that will run just like a normal java app on (more or less) any device.

We provide several different tools to help you run code on an Android device, covered fully in the [Android packaging documentation](#). These include creating a fully standalone APK that may be released on an Android store, as well as the ability to run your Kivy apps without a compilation step using our pre-prepared Kivy Launcher app.

### 1.1.7 Installation on Raspberry Pi

---

**Note:** The current Kivy port works partially: keyboard is not currently working.

---

#### Manual installation

1. Add APT sources for Gstreamer 1.0 in `/etc/apt/sources.list`:

```
deb http://vontaene.de/raspbian-updates/ . main
```

2. Install the dependencies:

```
sudo apt-get update
sudo apt-get install pkg-config libgl1-mesa-dev libgles2-mesa-dev \
python-pygame python-setuptools libgstreamer1.0-dev git-core \
gstreamer1.0-plugins-{bad,base,good,ugly} gstreamer1.0-{omx,alsa}
```

3. Install pip from source:

```
wget https://raw.githubusercontent.com/pypa/pip/master/contrib/get-pip.py
sudo python get-pip.py
```

4. Install Cython from sources (debian package are outdated):

```
sudo pip install cython
```

5. Clone and compile Kivy:

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

6. Build and use kivy inplace (best for development):

```
make
echo "export PYTHONPATH=$(pwd):\$PYTHONPATH" >> ~/.profile
source ~/.profile
```

7. Or install Kivy globally on your system:

```
python setup.py build
sudo python setup.py install
```

Running the demo

Go to your *kivy/examples* folder, you'll have tons of demo you could try.

You could start the showcase:

```
cd kivy/examples/demo/showcase
python main.py
```

3d monkey demo is also fun too see:

```
cd kivy/examples/3Drendering
python main.py
```

Where to go ?

We made few games using GPIO / physical input we got during Pycon 2013: a button and a tilt. Check-out the <https://github.com/kivy/piki>. You will need to adapt the GPIO pin in the code.

A video to see what we were doing with it: <http://www.youtube.com/watch?v=NVM09gaX6pQ>

## 1.1.8 Troubleshooting on Mac OS X

Having trouble installing Kivy on Mac OS X? This page contains issues

“Unable to find any valuable Window provider” Error

If you get an error like this:

```
$ python main.py
[INFO ] Kivy v1.8.0-dev
[INFO ] [Logger      ] Record log in /Users/audreyr/.kivy/logs/kivy_13-07-07_2.txt
[INFO ] [Factory     ] 143 symbols loaded
[DEBUG ] [Cache       ] register <kv.lang> with limit=None, timeout=Nones
[DEBUG ] [Cache       ] register <kv.image> with limit=None, timeout=60s
[DEBUG ] [Cache       ] register <kv.atlas> with limit=None, timeout=Nones
[INFO ] [Image       ] Providers: img_imageio, img_tex, img_dds, img_pil, img_gif (img_pygame i
[DEBUG ] [Cache       ] register <kv.texture> with limit=1000, timeout=60s
[DEBUG ] [Cache       ] register <kv.shader> with limit=1000, timeout=3600s
```

```
[DEBUG ] [App          ] Loading kv <./pong.kv>
[DEBUG ] [Window        ] Ignored <egl_rpi> (import error)
[DEBUG ] [Window        ] Ignored <pygame> (import error)
[WARNING] [WinPygame     ] SDL wrapper failed to import!
[DEBUG ] [Window        ] Ignored <sdl> (import error)
[DEBUG ] [Window        ] Ignored <x11> (import error)
[CRITICAL] [Window        ] Unable to find any valuable Window provider at all!
[CRITICAL] [App           ] Unable to get a Window, abort.
```

Then most likely Kivy cannot import PyGame for some reason. Continue on to the next section.

### Check for Problems with Your PyGame Installation

First, check that you have a working version of PyGame.

Start up the interactive Python interpreter and try to import pygame:

```
$ python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012, 20:52:43)
Type "copyright", "credits" or "license" for more information.
>>> import pygame
```

If you can import pygame without problems, then skip to the next section.

But if you get an error, then PyGame is not working as it should.

Here's an example of a PyGame error:

```
ImportError                                Traceback (most recent call last)
<ipython-input-1-4a415d16fbed> in <module>()
----> 1 import pygame

/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pygame/__init__.py
 93
 94 #first, the "required" modules
--> 95 from pygame.base import *
 96 from pygame.constants import *
 97 from pygame.version import *

ImportError: dlopen(/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
  Referenced from: /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/
  Expected in: flat namespace
  in /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pygame/base.so
```

And here is another example of a PyGame error:

```
ImportError                                Traceback (most recent call last)
<ipython-input-1-4a415d16fbed> in <module>()
----> 1 import pygame

/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pygame/__init__.py
 93
 94 #first, the "required" modules
--> 95 from pygame.base import *
 96 from pygame.constants import *
 97 from pygame.version import *
```

```
ImportError: dlopen(/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pygame/base.so:
```

The easiest way to resolve these PyGame import errors is:

1. **Delete the `pygame` package.** (For example, if you get the error above, delete `/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pygame/` and the accompanying egg.
2. **Try installing a PyGame binary for your version of Mac OS X.** Download it from <http://www.pygame.org/download.shtml>.
3. Repeat this process and try different PyGame Mac OS X binaries until you find one that works.

## 1.2 Development Version

The development version is for developers and testers. Note that when running a development version, you're running potentially broken code at your own risk. To use the development version, you will first need to install the dependencies. Thereafter, you will need to set up Kivy on your computer in a way that allows for easy development. For that, please see our [Contributing](#) document.

### 1.2.1 Installing Dependencies

To install Kivy's dependencies, follow the guide below for your platform.

Ubuntu

For Ubuntu 12.04 and above (tested to 14.04), simply enter the following command that will install all necessary packages:

```
$ sudo apt-get install python-setuptools python-pygame python-opengl \
python-gst0.10 python-enchant gstreamer0.10-plugins-good python-dev \
build-essential libgl1-mesa-dev-lts-quantal libgles2-mesa-dev-lts-quantal\
python-pip
```

For older versions of Ubuntu, this one should work:

```
$ sudo apt-get install python-setuptools python-pygame python-opengl \
python-gst0.10 python-enchant gstreamer0.10-plugins-good python-dev \
build-essential libgl1-mesa-dev libgles2-mesa-dev zlib1g-dev python-pip
```

Kivy requires a recent version of Cython, so it's better to use the last version published on pypi:

```
$ sudo pip install --upgrade cython
```

Mac OS X

You will need to install at least the following:

- PyGame - we recommend installing from a binary packaged for your version of Mac OS X. Download it from <http://www.pygame.org/download.shtml>

If you run into problems, please read [Troubleshooting on Mac OS X](#).

## 1.2.2 Installing Kivy for Development

Now that you've installed all the required dependencies, it's time to download and compile a development version of Kivy:

```
$ # Download Kivy from GitHub
$ git clone git://github.com/kivy/kivy.git
$ cd kivy

$ # Compile:
$ python setup.py build_ext --inplace -f
```

If you have the `make` command available, you can also use the following shortcut to compile (does the same as the last command):

```
$ make
```

**Warning:** By default, versions 2.7 to 2.7.2 of Python use the gcc compiler which ships with earlier versions of XCode. As of version 4.2, only the clang compiler is shipped with XCode by default. This means that if you build using XCode 4.2 or above, you need to ensure you have at least Python 2.7.3 installed, but preferably the latest version (2.7.5 at the time of writing).

If you want to modify the Kivy code itself, set up the `PYTHONPATH` environment variable to point at your clone. This way you don't have to install (`setup.py install`) after every tiny modification. Python will instead import Kivy from your clone.

Alternatively, if you don't want to make any changes to Kivy itself, you can also run (as admin, e.g. with `sudo`):

```
$ python setup.py install
```

If you want to contribute code (patches, new features) to the Kivy code base, please read [Contributing](#).

## 1.2.3 Running the test suite

To help detect issues and behaviour changes in Kivy, a set of unittests are provided. A good thing to do is to run them just after your Kivy installation, and every time you intend to push a change. If you think something was broken in Kivy, perhaps a test will show this? If not, it might be a good time to write one.)

Kivy tests are based on nosetest, which you can install from your package manager or using pip :

```
$ pip install nose
```

To run the test suite, do :

```
$ make test
```

## 1.2.4 Uninstalling Kivy

If you are mixing multiple Kivy installations, you might be confused about where each Kivy version is located. Please note that you might need to follow these steps multiple times if you have multiple Kivy versions installed in the Python library path. To find your current installed version, you can use the command line:

```
$ python -c 'import kivy; print(kivy.__path__)'
```

Then, remove that directory recursively.

If you have installed Kivy with `easy_install` on linux, the directory may contain a “egg” directory. Remove that as well:

```
$ python -c 'import kivy; print(kivy.__path__)'  
['/usr/local/lib/python2.7/dist-packages/Kivy-1.0.7-py2.7-linux-x86_64.egg/kivy']  
$ sudo rm -rf /usr/local/lib/python2.7/dist-packages/Kivy-1.0.7-py2.7-linux-x86_64.egg
```

If you have installed with `apt-get`, do:

```
$ sudo apt-get remove --purge python-kivy
```



# PHILOSOPHY

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

## 2.1 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?

### 2.1.1 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.

### 2.1.2 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.

### 2.1.3 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.

#### 2.1.4 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.

#### 2.1.5 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.

#### 2.1.6 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.

## 3.1 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.

## 3.2 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.

## 3.3 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).

### 3.3.1 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.

### 3.3.2 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.

### 3.3.3 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 Screencasts](#) 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.

### 3.3.4 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 [Installing Kivy for Development](#), 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 *Installing Kivy for Development*).
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!

## 3.4 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.

### 3.4.1 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](#).

## 3.5 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.

### 3.5.1 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 work via image comparison

To be able to run unit tests, you need to install nose (<http://code.google.com/p/python-nose/>), and coverage (<http://nedbatchelder.com/code/coverage/>). You can use `easy_install` for that:

```
sudo easy_install nose 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`. Nose 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:

```
nosetests kivy/tests/test_yourtestcase.py
```

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
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 *nosetests* 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 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.

## 3.6 GSOC

### 3.6.1 Google Summer of Code - 2014

#### 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.

We are hoping to participate in Google Summer of Code 2014 under PSF. 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)
- 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, mac with xcode and a developer license, to work on pyjnius you would need an android device, and to work on pleyer you would need access to hardware for both platforms.

Additional desired skills may be listed with specific projects.

Familiarize yourself with the contributing guide <http://kivy.org/docs/contribute.html> We can help you get up to speed, however students demonstrating ability in advance will be given preference.

#### How to get setup

For Kivy 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.

eg.. for pyjnius:

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

## Project Ideas

The mentors list is only of potential mentors for a particular project and not final.

## Enhancements to Kivy

### Inspector:

#### Description:

**Redo or improve the inspector module to include the following features:**

- Use Python introspection to enhance current state of inspector.
- Extend Inspectors debugging capabilities to the whole app.
- Introduce automatic crash reporting.
- Possibly launch debugger automatically when Kivy app crashes.

#### Reference:

- <http://kivy.org/docs/api-kivy.modules.html>
- <http://kivy.org/docs/api-kivy.modules.inspector.html>

**Expected Outcome:** A fully functional Inspector module that facilitates debugging at any stage, including crash reports and a debugging console.

- **Mentors:** Akshay Arora, Gabriel Pettier
- **Task level:** Intermediate

### Graphics Pipeline Enhancements:

#### Description:

**We have a lot of ideas around the graphics pipeline:**

- Merging instructions
- VBOs to reduce GL calls
- Helpers to create shaders dynamically according to the current vertex format
- Improving 3D support.
- Add Bounding-Box calculation / selection on the tree only if requested
- Unit tests to quantify the amount of improvements achieved.

#### Reference:

- <http://kivy.org/docs/api-kivy.graphics.html>
- <http://www.khronos.org/opengles/>

**Expected Outcome:** Significant improvement in the graphics pipeline that can be quantified by tests.

- **Mentors:** Jacob Kovac, Mathieu Virbel
- **Task level:** Intermediate/Advanced

- **Desired Skills:** Familiarity with OpenGL ES and Cython, desire to learn about advanced rendering algorithms and solve difficult puzzles.

#### Embedded Support:

**Description:** Add full support for major embedded platforms like Beagle Board and Raspberry Pi. Kivy already has partial support for RPi. It would be great to have support for other major embedded platforms.

#### This would involve:

- Native Keyboard Provider.
- Window provider for Beagle board using hooks to the driver for hardware acceleration inspiration can be taken from the rpi window provider [https://github.com/kivy/kivy/blob/master/kivy/core/window/window\\_egl\\_rpi.py](https://github.com/kivy/kivy/blob/master/kivy/core/window/window_egl_rpi.py).
- Ensuring at least one of the backends for each of the core providers work on the embedded hardware with acceptable performance. Namely: Text, Window, Audio, Video, Keyboard, Clipboard, and Image Providers

#### Reference:

- [https://github.com/kivy/kivy/blob/master/kivy/core/window/window\\_egl\\_rpi.py](https://github.com/kivy/kivy/blob/master/kivy/core/window/window_egl_rpi.py).
- <http://kivy.org/docs/api-kivy.core.html>
- <http://kivy.org/docs/guide/architecture.html#architecture>
- <http://kivy.org/docs/guide/architecture.html#providers>

**Expected Outcome:** Full Working support for the embedded platforms. This would include support for at least one of the backends for each core providers mentioned above to achieve feature parity with other platforms.

- **Mentors:** Gabriel Pettier, Mathieu Virbel
- **Requirements:** Access to specific embedded hardware.
- **Task level:** Intermediate/Advanced
- **Desired Skills:** Familiarity with programming on the specific embedded hardware.

## Enhancements to Mobile Platforms

### Plyer:

**Description:** Plyer is a platform-independant api to use features commonly found on various platforms, especially mobile ones, in Python. The idea is to provide a stable API to the user for accessing features of their desktop or mobile device.

The student would work on facades and implementation for Accelerometer, GPS, SMS, Contact etc., including porting facades from SL4A (Scripting Layer for Android) to Plyer for easy integration and compatibility.

Under the hood you'll use PyJNIus and PyObjus. 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>

**Expected Outcome:** Platform independent api for accessing most platform specific features.

- **Mentors:** Gabriel Pettier, Akshay Arora, Alexander Taylor, Ben Rousch.
- **Requirements:** Access to Linux, Windows, OS X, iOS device, Android device.
- **Task level:** Intermediate/Advanced.
- **Desired Skills:** Familiarity with Pyjnius, PyObjus.

## Enhancements to Toolchain

### Python-for-iOS:

**Description:** An iOS interface for building an app for the app store based on the idea of Python for Android, <https://github.com/kivy/python-for-android>

**Current state:** Kivy iOS achieves this but in a more restricted monolith manner. We'd like to replace it with a more modular tool that is more extensible.

#### References:

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

**Expected Outcome:** A new, modular and extendable toolchain.

- **Mentors:** Thomas Hansen, Mathieu Virbel
- **Requirements:** Access to iOS, Android device along with a developer license.
- **Task level:** Intermediate/Advanced
- **Desired Skills:** Familiarity with xcode, objc. Cross-platform compilation skills are heavily required.

### Buildozer:

**Description:** Buildozer is a Python tool for creating application packages easily. The goal is to have one "buildozer.spec" file in your app directory, describing your application requirements and settings such as title, icon, included modules etc. Buildozer will use that spec to create a package for Android, iOS, Windows, OS X and/or Linux.

Buildozer currently supports packaging for Android via the python-for-android project, and for iOS via the kivy-ios project. This project would involve extending this support to other package formats and operating systems, e.g. RPM, DEB, DMG, EXE. You will need to write new buildozer target code to support these formats.

This project would also involve optimizations to the final package formats, e.g. introspect the python files, extract all the symbols (global variables / functions / class / class methods), mark all the necessary symbols (whitelist) and generate a stripped version of all the python files without the unnecessary symbols.

#### References:

- <https://github.com/kivy/Buildozer>
- <https://github.com/kivy/buildozer/tree/master/buildozer/targets>

**Expected Outcome:** Running *buildozer deb debug* or *buildozer deb release* should result in a .deb package that can be directly be used to install on the target machine, or the equivalent for other package and binary formats.

- **Mentors:** Gabriel Pettier, Akshay Arora, Alexander Taylor, Ben Rousch
- **Requirements:** Access to linux, Windows, OS X, iOS, Android.
- **Task level:** Intermediate

## SDL2 Backends:

**Description:** SDL2 backend providers for Kivy, including porting the mobile toolchains to SDL2. Part of the work is already done. What left is mostly

- Hashing out distribution mechanisms for the lib.
- Porting mobile backends for ios and android to SDL2. Partial work on this has already been going on.
- Unit tests for the new sdl2 backends making sure apps work the same on SDL2 as on other backends.
- Performance testing. Looking at the difference between SDL2 and other providers to ascertain whether sdl2 could be used as the default provider giving it priority

**References:** <https://github.com/kivy/kivy/tree/sdl2-support>

**Expected Outcome:** Completing the existing and adding new SDL2 core providers and support for using sdl2 on mobiles.

- **Mentors:** Akshay Arora, Jacob Kovac, Mathieu Virbel
- **Requirements:** Access to Linux, Windows, OS X, iOS, Android.
- **Task level:** Intermediate/Advanced

## Anything Else ?

- Let your imagination run wild, and show what Kivy is capable of!

## How to Contact devs

Ask your questions on the Kivy users forums <http://kivy.org/#forum>

Or send a mail at [kivy-users@googlegroups.com](mailto:kivy-users@googlegroups.com)

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

You can also try to contact us on IRC (online chat), To get the irc handles of the devs mentioned above visit <http://kivy.org/#aboutus>

but make sure to read the IRC rules before connecting. [http://webchat.freenode.net/?nick=kvuser\\_GSOC\\_&channels=#kivy-dev](http://webchat.freenode.net/?nick=kvuser_GSOC_&channels=#kivy-dev)

## 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, 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 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 pop 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 the 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, how, etc.
- Discuss that proposal with us in a timely manner. Get feedback.
- Be patient! Especially on IRC. 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.





---

# FAQ

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

## 4.1 Technical FAQ

### 4.1.1 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.

### 4.1.2 undefined symbol: glGenerateMipmap

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

### 4.1.3 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
```

### 4.1.4 Pip installation failed

Installing Kivy using Pip is not currently supported. Because Pip forces the usage of setuptools, setuptools hacks build\_ext to use pyrex for generating .c, meaning there is no clean solution to hack against

both weird behaviors to use Cython. (Reference: <http://mail.scipy.org/pipermail/nipy-devel/2011-March/005709.html>)

Solution: use *easy\_install*, as our documentation said.

## 4.1.5 GStreamer compatibility

Starting from 1.8.0 version, Kivy now use by default the Gi bindings, on the platforms that have Gi. We are still in a transition, as Gstreamer 0.10 is now unmaintained by the Gstreamer team. But 1.0 is not accessible with Python everywhere. Here is the compatibility table you can use.

Gstreamer version	Bindings	Status	Remarks
0.10	pygst	Works	Lot of issues remain with 0.10
0.10	gi	Buggy	Internal issues with pygobject, and video doesn't play.
1.0	pygst	•	No pygst bindings exists for 1.0
1.0	gi	Works*	Linux: works OSX: works with brew Windows: No python bindings available

Also, we have no reliable way to check if you have 1.0 installed on your system. Trying to import gi, and then pygst, will fail.

So currently:

- if you are on Windows: stay on Gstreamer 0.10 with pygst.
- if you are on OSX/Linux: install Gstreamer 1.0.x
- if you are on OSX/Linux and doesn't want to install 1.0: *export KIVY\_VIDEO=pygst*

If you are on OSX, Brew now have *pygobject3*. You must install it, and re-install gstreamer with introspection options:

```
$ brew install pygobject3
$ brew install gstreamer --with-gobject-introspection
$ brew install gst-plugins-base --with-gobject-introspection
$ brew install gst-plugins-good --with-gobject-introspection
$ brew install gst-plugins-bad --with-gobject-introspection
$ brew install gst-plugins-ugly --with-gobject-introspection

# then add the gi into your PYTHONPATH (as they don't do it for you)
$ export PYTHONPATH=$PYTHONPATH:/usr/local/opt/pygobject3/lib/python2.7/site-packages

# test it
$ python -c 'import gi; from gi.repository import Gst; print Gst.version()'
(1L, 2L, 1L, 0L)
```

## 4.2 Android FAQ

### 4.2.1 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.

#### 4.2.2 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.

#### 4.2.3 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](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.

#### 4.2.4 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.

### 4.3 Project FAQ

#### 4.3.1 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.

#### 4.3.2 Does Kivy support Python 3.x?

Yes! As of version 1.8.0 Kivy supports both Python 2 and Python 3 with the same codebase.

#### 4.3.3 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.

#### 4.3.4 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.

#### 4.3.5 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 IRC. 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:

## 5.1 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.

## 5.2 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](#).

## 5.3 IRC

### #Kivy on irc.freenode.net

IRC 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 an IRC client, you can also use [Freenode's web chat](#), but please, don't close the browser window too soon. Just enter `#kivy` in the channels field.

### Some #kivy guidelines

- Be polite and respectful towards other users.
- Please use English for all messages and in any code posted.
- Do not post large log/code copy and pastes to the channel, instead use a site such as <http://pastebin.com/> or <https://gist.github.com/>.

- If you have a question, be clear and succinct to ensure the fastest response. Don't just say "it doesn't work" or "help". Instead, describe what you expected to happen and what actually happened. Show code, pictures, or error logs as necessary.
- If your code follows [PEP8](#) you will most likely receive a quicker answer.
- Try to lower the barrier to helping you as much as possible, if we need to run your application, share it in one file like in [this template](#). Do not share your entire application, share a simple example that showcases only the issue you are experiencing.
- **Avoid topics that may make others uncomfortable or that are excessively incendiary.** #kivy is a community for learning and troubleshooting. There are other places for discussions on sex, drugs, politics, religion, and anything else you can think of to talk about that could potentially be uncomfortable or inflammatory in nature for others.
- **Also Avoid:**
  - Racial epithets of any kind regardless of context.
  - Excessive Swearing.
  - Misogynist, misandrist, homophobic or transphobic behavior or attitudes.



Part II  
**PROGRAMMING GUIDE**



---

# KIVY BASICS

## 6.1 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.

- [Installation on Windows](#)
- [Installation on MacOSX](#)
- [Installation on Linux](#)

If you want to install everything yourself, ensure that you have at least [Cython](#) and [Pygame](#). A typical pip installation looks like this:

```
pip install cython
pip install hg+http://bitbucket.org/pygame/pygame
pip install kivy
```

The [development version](#) can be installed with git:

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

## 6.2 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 !

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

class MyApp(App):
```

```

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

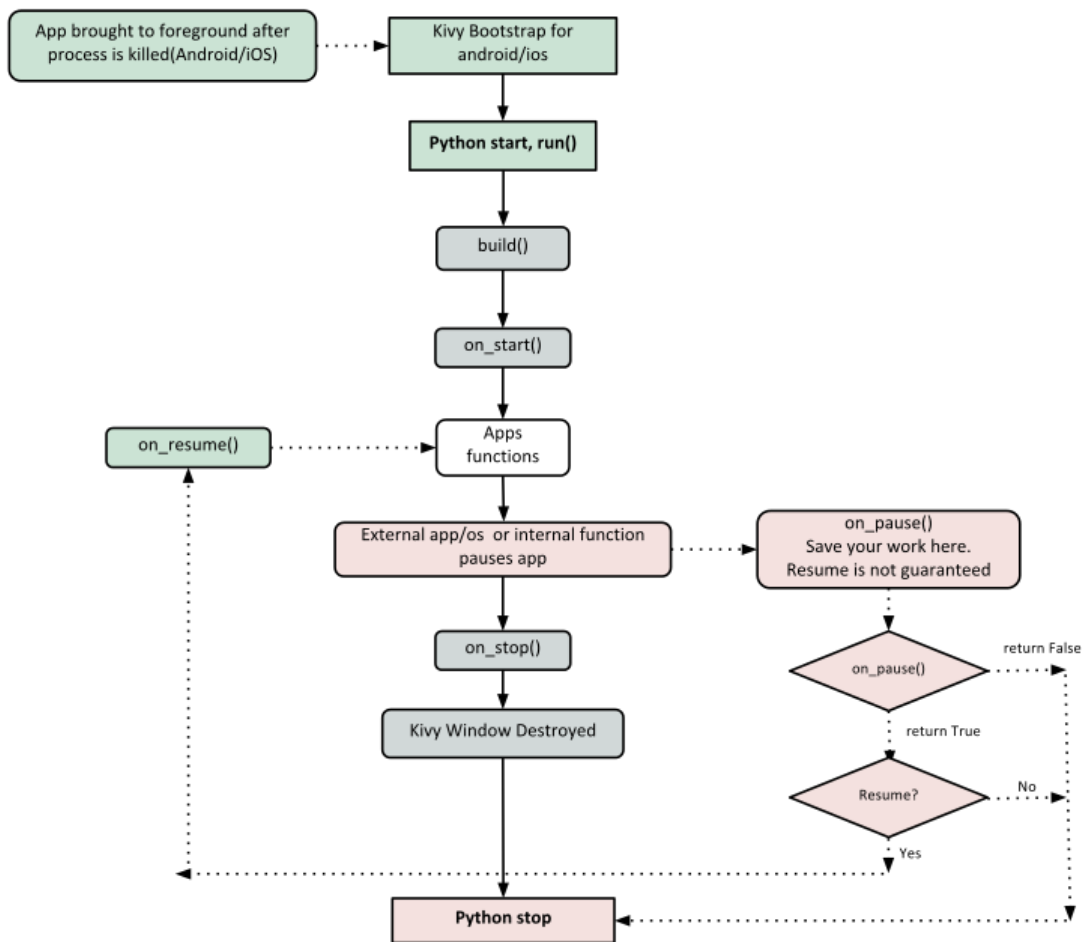
if __name__ == '__main__':
    MyApp().run()

```

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

### 6.3 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.

## 6.4 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*:

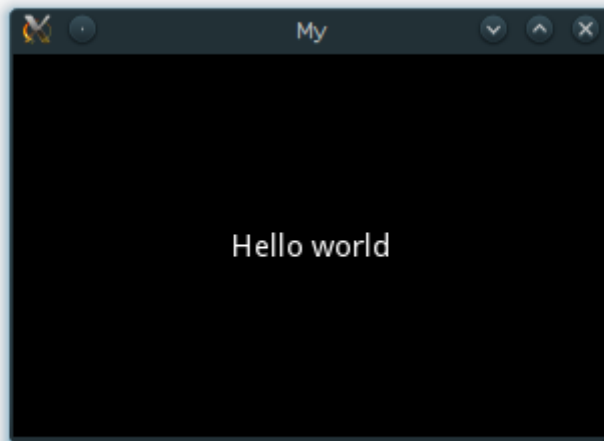
```
$ python main.py  
# or  
C:\appdir>kivy.bat main.py
```

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

```
$ 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.



## 6.5 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)

class MyApp(App):

    def build(self):
        return LoginScreen()

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

At the next line 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 overload the method `__init__()` so as to add widgets and to define their behavior:

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

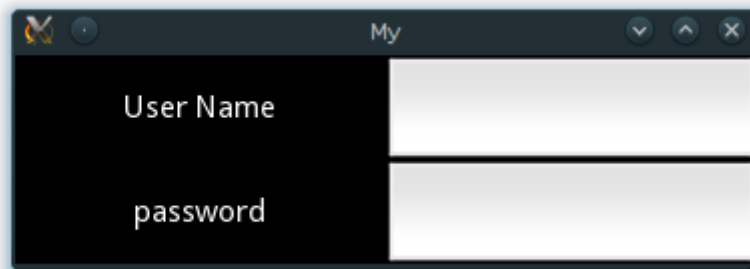
One should not forget to call `super` in order to implement the functionality of the original class being overloaded. Also note that it is good practice not to omit the `**kwargs` while calling `super`, as they are sometimes used internally.

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 it's 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.

## 6.6 Platform specifics

Opening a Terminal application and setting the kivy environment variables.

On Windows, just double click the `kivy.bat` and a terminal will be opened with all the required variables already set.

On nix\* systems, open the terminal of your choice and if kivy isn't installed globally:

```
export python=$PYTHONPATH:/path/to/kivy_installation
```





---

# CONTROLLING THE ENVIRONMENT

---

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

For example, for restricting text rendering to PIL implementation:

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

Environment variable can be set before importing kivy:

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

## 7.1 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 write, and no user configuration directory too.

**KIVY\_NO\_FILELOG** If set, logs will be not print on a file

**KIVY\_NO\_CONSOLELOG** If set, logs will be not print on the console

## 7.2 Path control

New in version 1.0.7.

You can control where the default directory of modules, extensions, and kivy data is located.

**KIVY\_DATA\_DIR** Location of the Kivy data, default to *<kivy path>/data*

**KIVY\_EXTS\_DIR** Location of the Kivy extensions, default to *<kivy path>/extensions*

**KIVY\_MODULES\_DIR** Location of the Kivy modules, default to *<kivy path>/modules*

## 7.3 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: pygame, x11, sdl, egl\_rpi

**KIVY\_TEXT** Implementation to use for rendering text

Values: pil, pygame, sdl\_ttf

**KIVY\_VIDEO** Implementation to use for rendering video

Values: pygst, gstplayer, pyglet, ffmpeg, null

**KIVY\_AUDIO** Implementation to use for playing audio

Values: gstplayer, pygst, sdl, pygame

**KIVY\_IMAGE** Implementation to use for reading image

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

**KIVY\_CAMERA** Implementation to use for reading camera

Values: videocapture, avfoundation, pygst, opencv

**KIVY\_SPELLING** Implementation to use for spelling

Values: enchant, osxappkit

**KIVY\_CLIPBOARD** Implementation to use for clipboard management

Values: pygame, dummy, android

## 7.4 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.

# CONFIGURE KIVY

The configuration file for kivy is named *config.ini*, and adheres to the [standard INI](#) format.

## 8.1 Locating the configuration file

The location of the configuration file is:

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

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

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

## 8.2 Understanding config tokens

All the configuration tokens are explained in the [kivy.config](#) module.



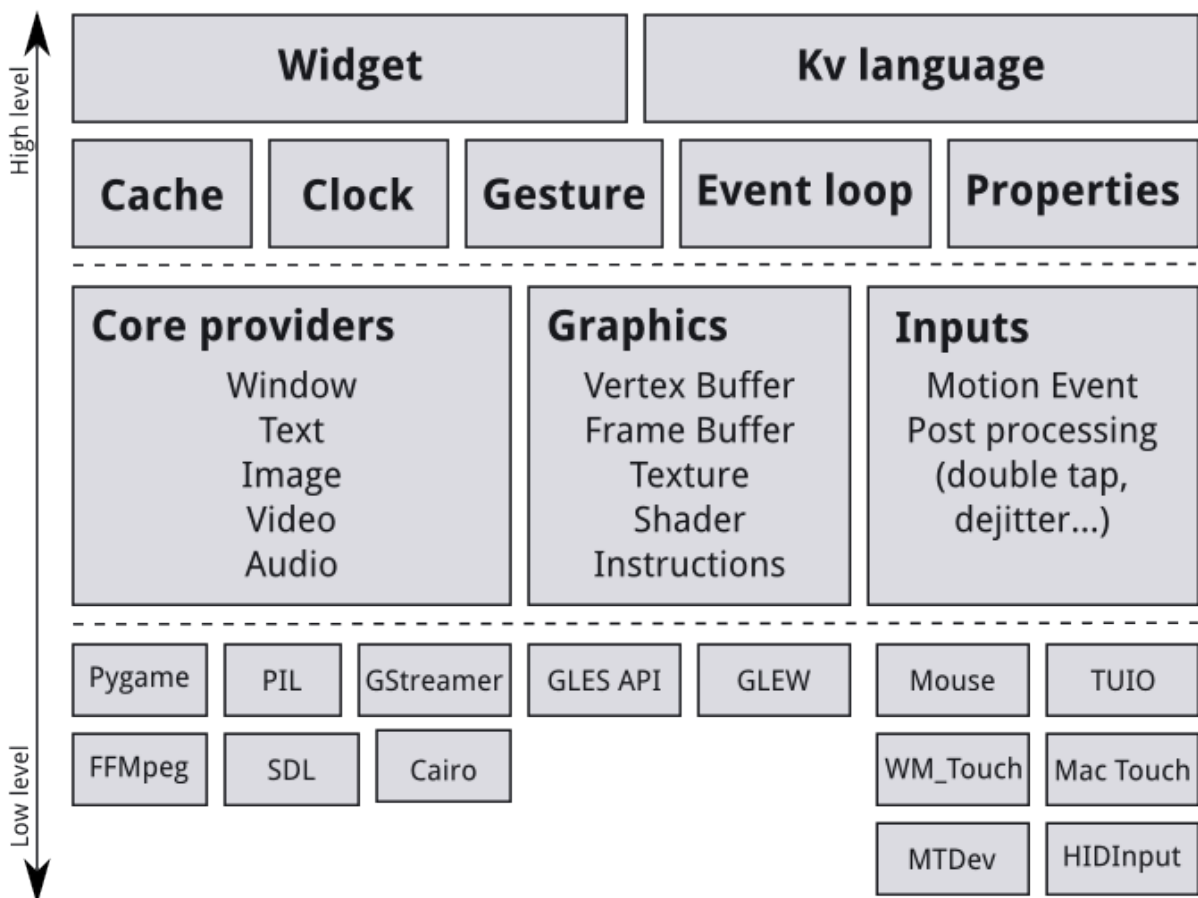
# ARCHITECTURAL OVERVIEW

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



## 9.1 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.

## 9.2 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 (GLES2) on all devices, so if you want to stay cross-platform compatible, we advise you to only use the GLES2 functions.

## 9.3 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.

## 9.4 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.

## 9.5 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.

## 9.6 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.

## 9.7 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[:]:
        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.

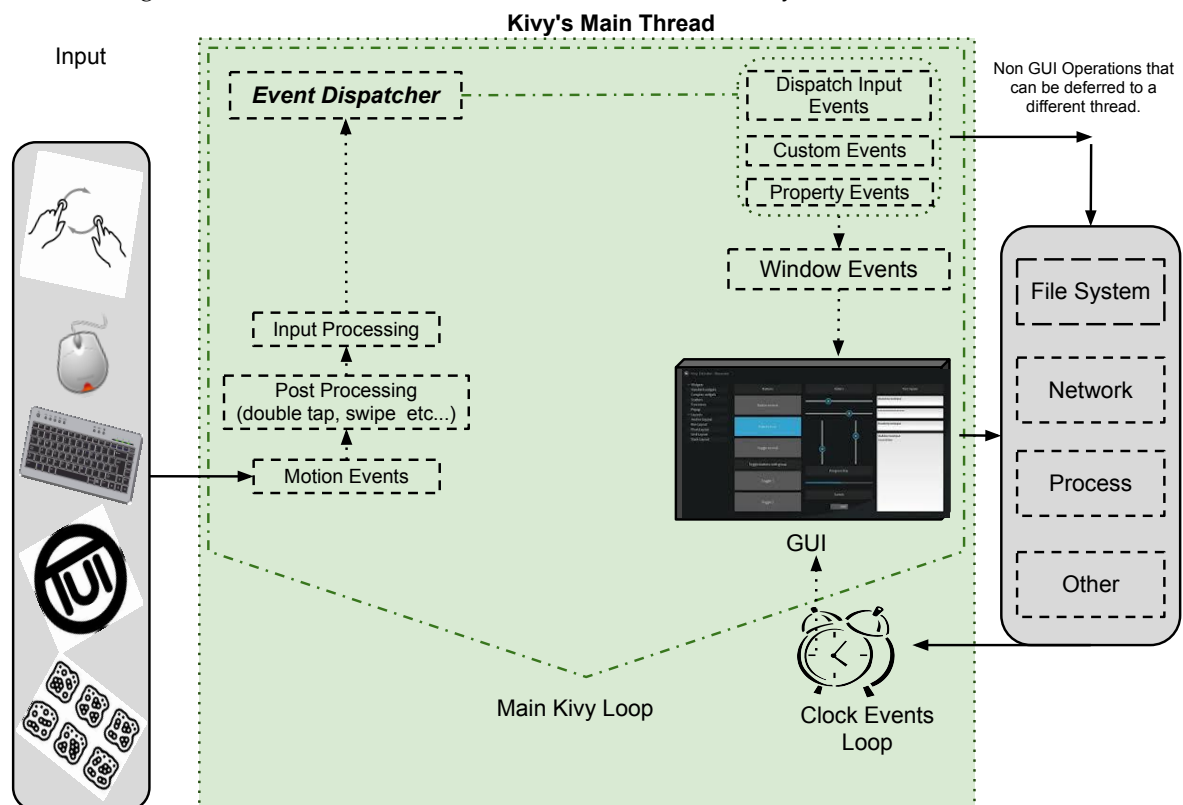
Often times you will want to restrict the *area* on the screen that a widget watches for touches. You can use a widget's `collide_point()` method to achieve this. You simply pass it the touch's position and it returns `True` if the touch is within the 'watched area' or `False` otherwise. By default, 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.



## 10.1 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.

## 10.2 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.

### 10.2.1 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
Clock.schedule_interval(my_callback, 1 / 30.)
```

You have two ways of unscheduling a previously scheduled event. The first would be to use `unschedule()`:

```
Clock.unschedule(my_callback)
```

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

### 10.2.2 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.

### 10.2.3 Trigger events

If you want to schedule a function to be called only once for the next frame, like a trigger, you might be tempted to achieve that like so:

```
Clock.unschedule(my_callback)
Clock.schedule_once(my_callback, 0)
```

This way of programming a trigger is expensive, since you'll always call `unschedule`, whether or not you've even scheduled it. In addition, `unschedule` needs to iterate the weakref list of the `Clock` in order to find your callback and remove it. 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.

## 10.3 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.

## 10.4 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
```

## 10.5 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()
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.

## 10.6 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

## 10.7 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)
```

## 10.8 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)
```

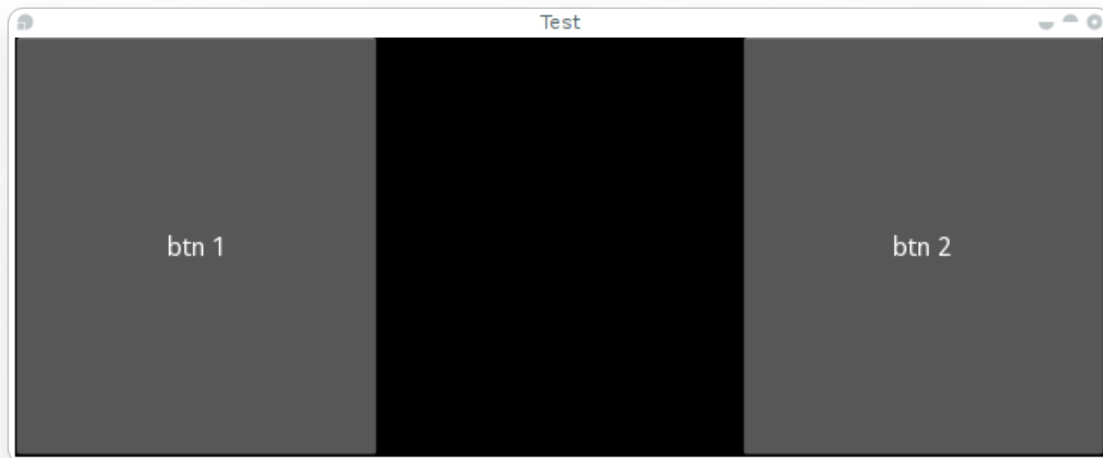
The first parameter would be the *instance* of the class the property is defined.

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()
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__':
```

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.

## 10.9 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, bind=(
2      'cursor', 'padding', 'pos', 'size', 'focus',
3      'scroll_x', 'scroll_y'))
4      '''Current position of the cursor, in (x, y).
5
6      :attr:'cursor_pos' is a :class:'~kivy.properties.AliasProperty', read-only.
7      '''

```

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

## 11.1 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()`

## 11.2 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.

## 11.3 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.

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
    touch.pop()

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

### 11.3.1 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))
    # ...
```

### 11.3.2 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)
    # ...
```

### 11.3.3 Triple tap

A triple tap is the action of tapping thrice within a time and a distance. It's calculated by the tripletap 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)
# ...

```

### 11.3.4 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.
- The touch coordinate is not translated to your widget space because the touch is coming directly from the Window. It's your job to convert the coordinate to your local space.

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 ourself, or you might have side effects
        touch.ungrab(self)

        # and accept the last up
        return True

```

### 11.3.5 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.

# WIDGETS

## 12.1 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 a in-depth explanation about the **Widget** class, look at the module documentation.

## 12.2 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`.

## 12.3 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.

## 12.4 Widgets Z Index

The order of drawing widgets is based on position in the widget tree. The last widget's canvas is drawn last (on top of everything else inside its parent). `add_widget` takes a `index` parameter:

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

for setting the z-index.

## 12.5 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.

## Box Layout

vertical

vertical

## Grid Layout

cols = 3

cols = 3

cols = 3

cols = 3

cols = 3

cols = 3

cols = 3

cols = 3

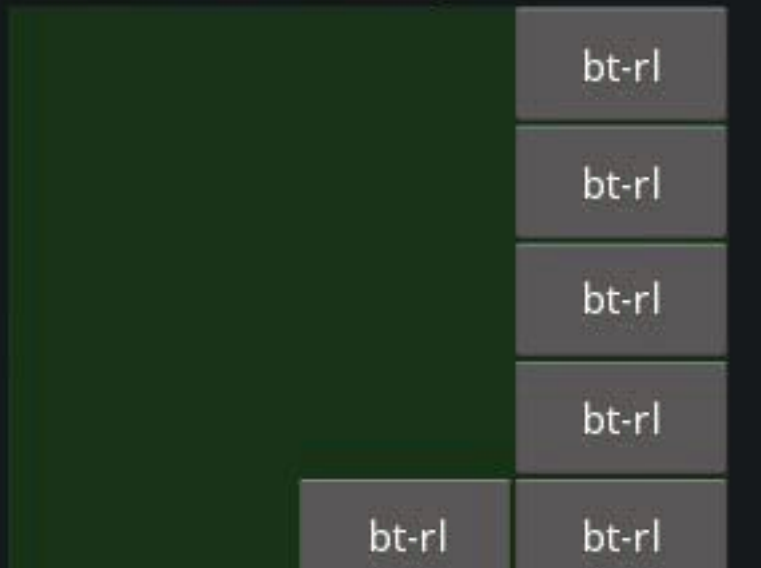
cols = 3

cols = 3

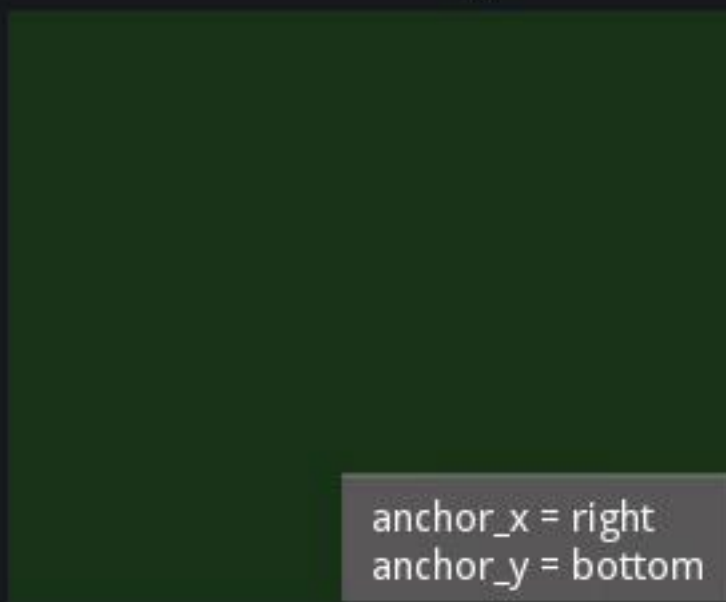
cols = 3



## 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 layout's 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

```

load kivy catalog:

```

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

```

Replace \$KIVYDIR with the directory of your installation of Kivy. Click on the button labeled *Box Layout* from the left. Now paste the code from above into the editor panel on the right.

The screenshot shows the Kivy IDE interface. On the left is a sidebar with a list of widget categories: Welcome, Float Layout, Box Layout (highlighted), Anchor Layout, Grid Layout, Stack Layout, Buttons, Labels, Booleans, Progress Bar, Media, Text, Popups, Selectors, File Choosers, Scatter, and ReST. The main editor area displays a preview of a BoxLayout widget with the following text: "I think my iPhone is broken. I pressed the `Home` button and I am still at school" followed by "--The iGeneration." on a new line. On the right is a code editor showing the Python code for the widget:
 

```

#:kivy 1.4

BoxLayout:
    orientation: 'vertical'
    padding: 20
    spacing: 10
    Button:
        text: '"I think my
        iPhone is broken. I pressed
        the `Home` button and I am
        still at school" \n\n --The
        iGeneration.'
        text_size: self.size
        halign: 'center'
        valign: 'middle'
        size_hint: 1, 1
    
```

 At the bottom right of the code editor, there are two buttons: "Auto Reload" (checked) and "Render Now".

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_x` 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.

Welcome  
 Float Layout  
**Box Layout**  
 Anchor Layout  
 Grid Layout  
 Stack Layout  
 Buttons  
 Labels  
 Booleans  
 Progress Bar  
 Media  
 Text  
 Popups  
 Selectors  
 File Choosers  
 Scatter  
 ReST

Twilight is like soccer. They all run around for 2 hours, nobody scores and it's billion fans insist,  
 "you just don't understand"

The four stages of life

1. You believe in Santa Claus
2. You don't believe in Santa Claus
3. You are Santa Claus
4. You look like Santa Claus

```

#:kivy 1.4
BoxLayout:
    padding: 20
    spacing: 10
    Button:
        font_size: '13sp'
        text: 'Twilight is like
        soccer. They all run around
        for 2 hours, nobody scores and
        it\'s billion fans insist,\n\n
        "you just don\'t understand"'
        text_size: self.size
        halign: 'center'
        valign: 'middle'
    Button:
        text: ' The four stages
        of life\n\n 1. You believe in
        Santa Claus\n 2. You don\'t
        believe in Santa Claus\n 3.
        You are Santa Claus\n\n 4. You
        look like Santa Claus'
        text_size: self.size
        halign: 'center'
        valign: 'middle'
  
```

Auto Reload

**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.

Welcome  
 Float Layout  
**Box Layout**  
 Anchor Layout  
 Grid Layout  
 Stack Layout  
 Buttons  
 Labels  
 Booleans  
 Progress Bar  
 Media  
 Text  
 Popups  
 Selectors  
 File Choosers  
 Scatter  
 ReST

Life's journey is not to arrive at the grave safely in a well preserved body but rather to skid in sideways, totally worn out, shouting  
 "Holy Shit... what a ride"

```

#:kivy 1.4
BoxLayout:
    padding: 20
    spacing: 10
    Button:
        size_hint: .5, 1
    Button:
        text: 'Life\'s
        journey is not to arrive at
        the grave safely in a well
        preserved body but rather to
        skid in sideways, totally
        worn out, shouting,\n\n
        "Holy Shit... what a ride"'
        text_size: self.size
        valign: 'middle'
        halign: 'center'
        size_hint: 1, 1
  
```

Auto Reload

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.

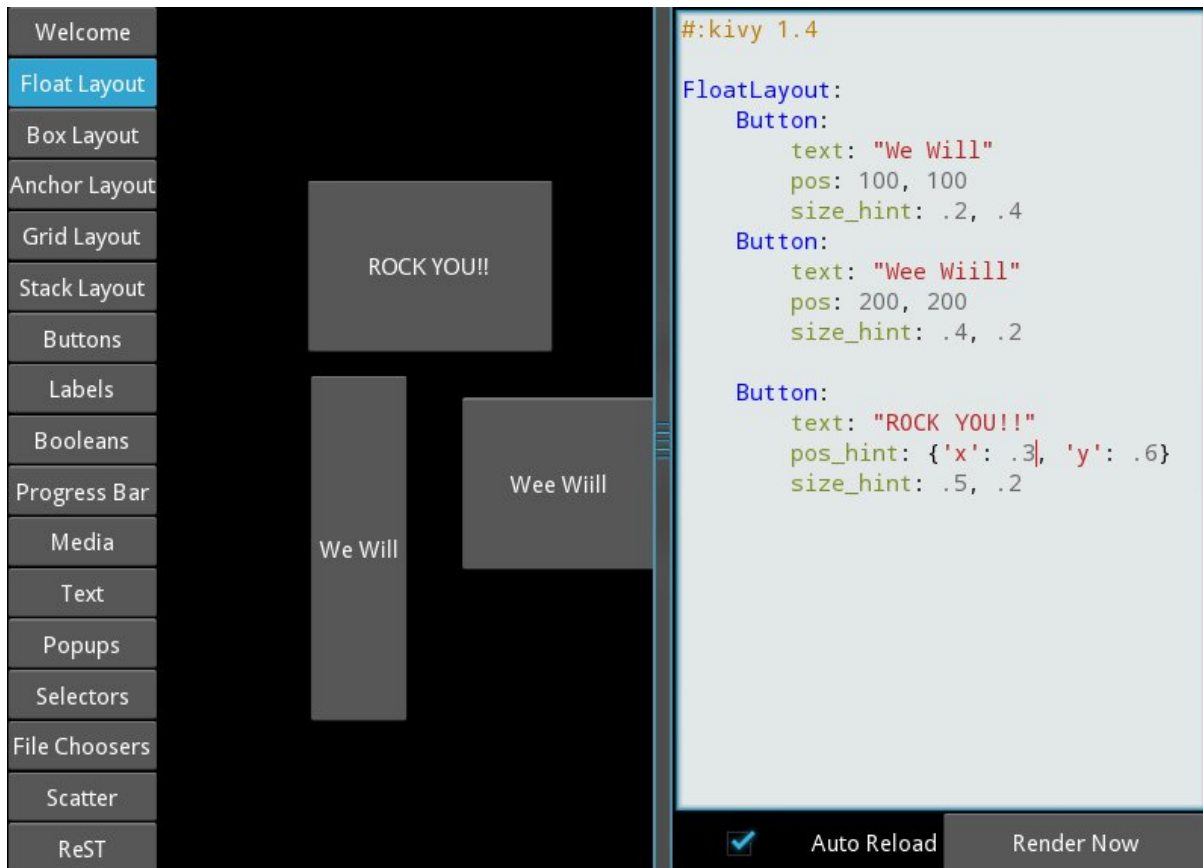
`pos_hint` is a dict, which defaults to empty. As for `size_hint`, layouts honor `pos_hint` differently, but generally you can add values to any of the `pos` attributes (`x`, `y`, `left`, `top`, `center_x`, `center_y`) to have the `Widget` positioned relative to its `parent`.

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"
        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.

## 12.6 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}))

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

```

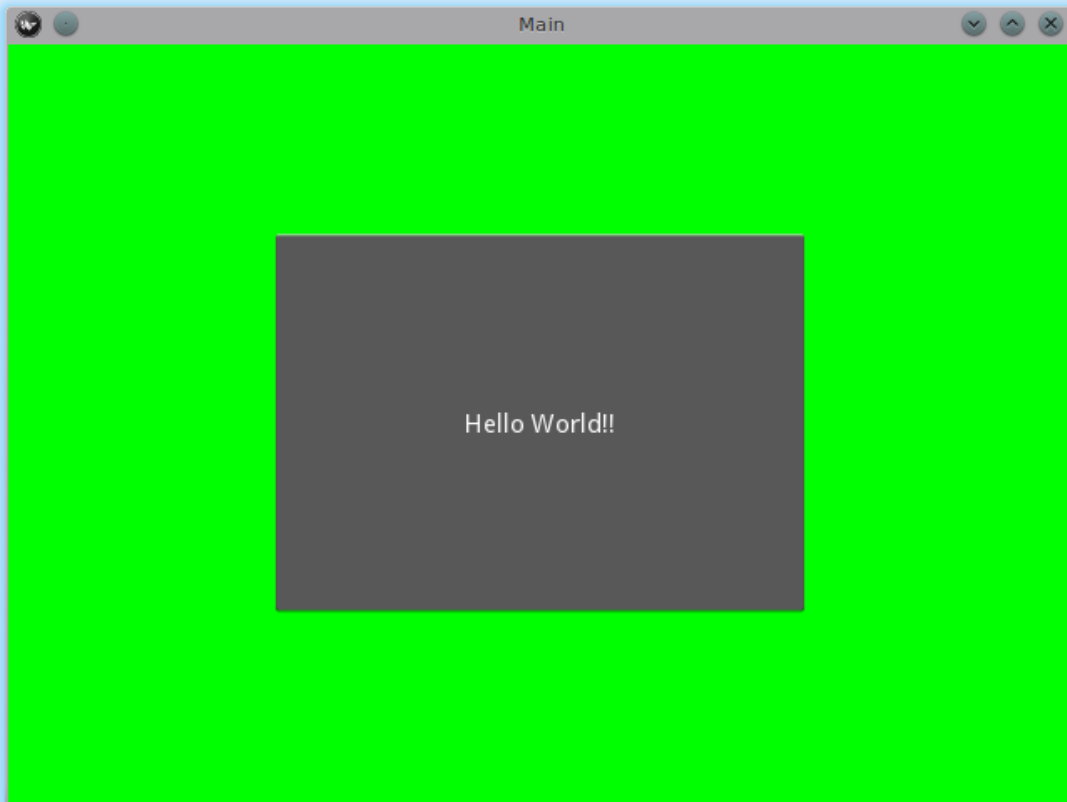
```
if __name__ == '__main__':  
    MainApp().run()
```

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:





### To 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
            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-289x289.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-289x289.jpg',
        c = CustomLayout()
        c.add_widget(
            AsyncImage(
                source="http://www.stuffistumbledupon.com/wp-content/uploads/2012/04/Get-a-Girlfriend-289x289.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-289x289.jpg'
            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-289x289.jpg'
    CustomLayout:
        AsyncImage:
            source: 'http://www.stuffistumbledupon.com/wp-content/uploads/2012/04/Get-a-Girlfriend-289x289.jpg'
            size_hint: 1, .5
            pos_hint: {'center_x':.5, 'center_y': .5}

```

```

'''
class RootWidget(BoxLayout):
    pass

class CustomLayout(FloatLayout):
    pass

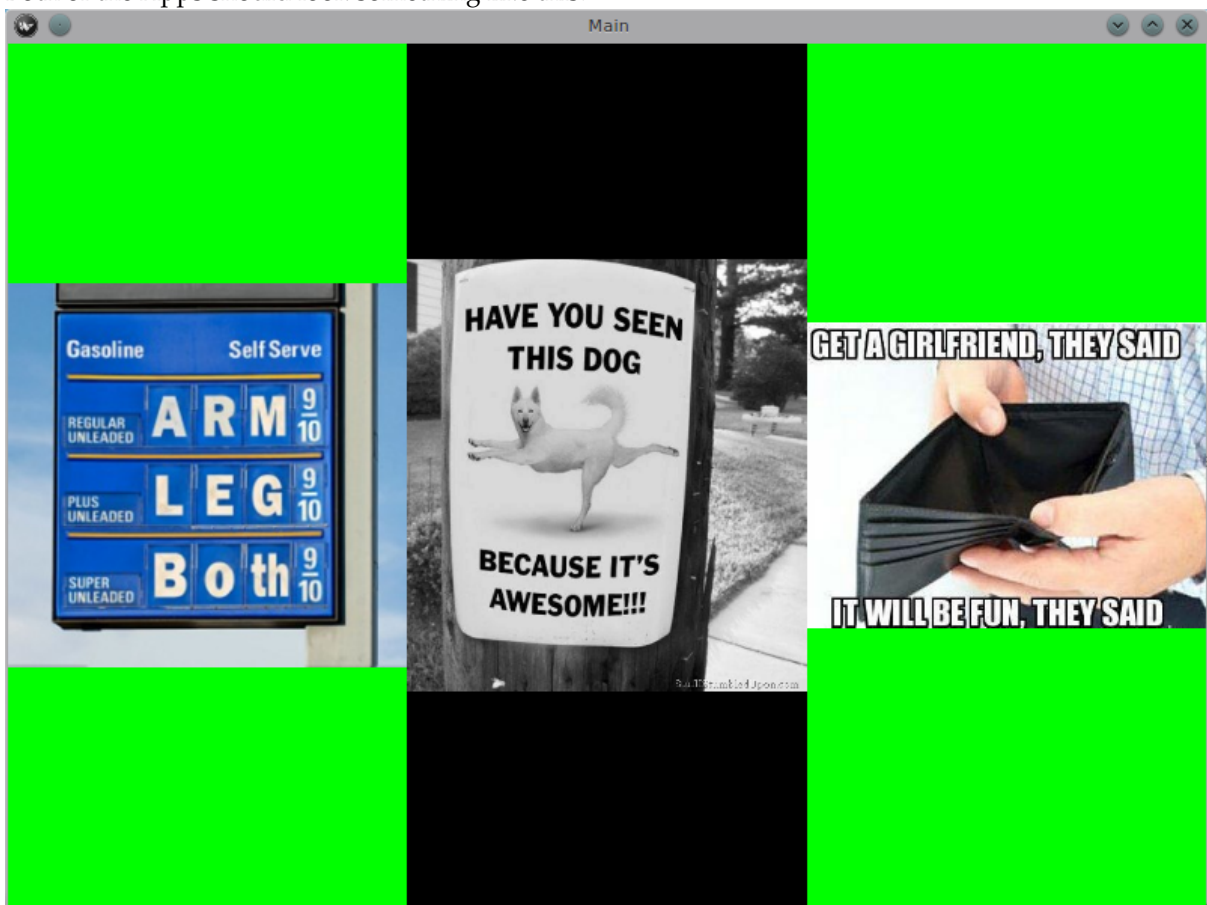
class MainApp(App):

    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
            text_size: self.width-20, self.height-20
            valign: 'bottom'
            halign: 'justify'
''')

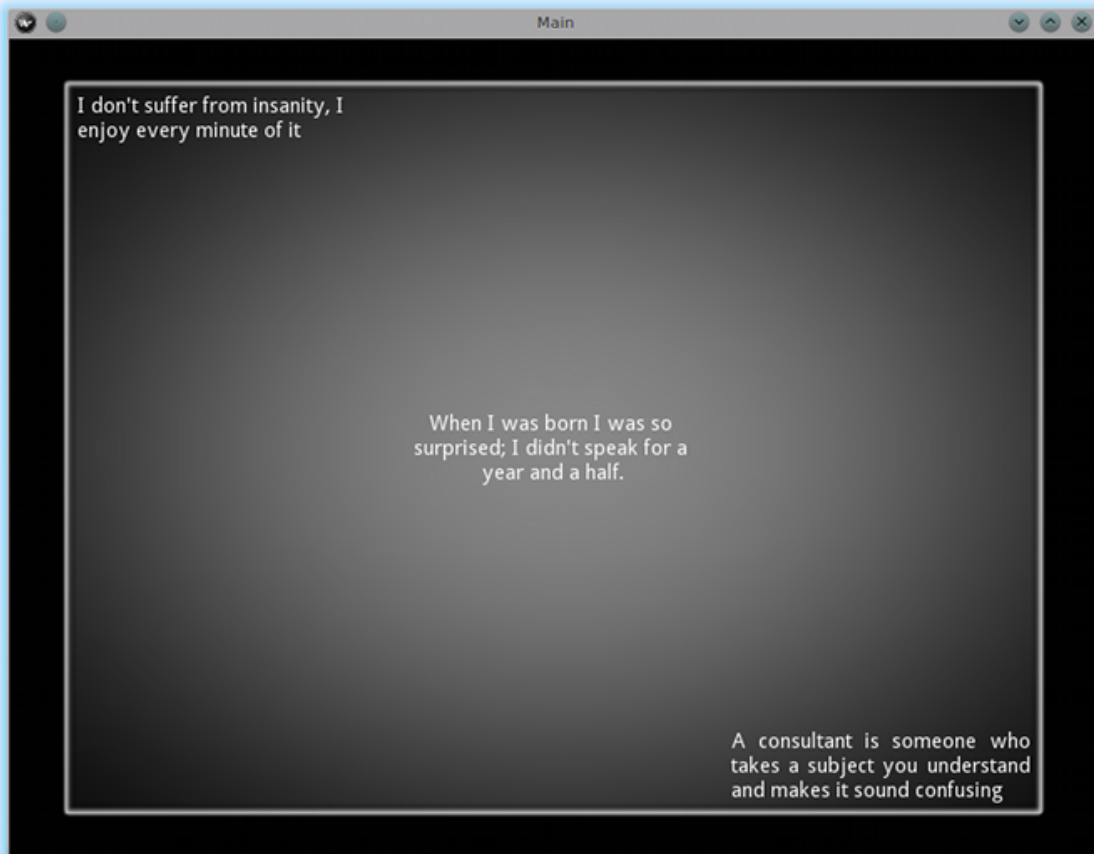
class RootWidget(FloatLayout):
    pass

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
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
      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**.

## 12.7 Nesting Layouts

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

## 12.8 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.

## 12.9 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

## 13.1 Introduction to Canvas

Widgets graphical representation is done using a canvas, which you can see both as an unlimited drawing board, and as a set of drawing instructions, there are numerous different instructions you can apply (add) to your canvas, but there is two main kind of them:

- `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. Thoses 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
```

## 13.2 Context instructions

Context instructions manipulate the opengl context, you can rotate, translate, and scale your canvas, 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')
```

## 13.3 Drawing instructions

Drawing instructions are ranging from very simple ones, to draw 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)
```

## 13.4 Manipulating instructions

Sometime, you want to update or remove the instructions you 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):  
        self.canvas.clear()  
  
        with self.canvas:  
            self.rect = Rectangle(pos=self.pos, size=self.size)
```

# KV LANGUAGE

## 14.1 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 *kvlng*, or *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 prototyping and agile changes to your UI. It also facilitates a good separation between the logic of your application and its User Interface.

## 14.2 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)
```

## 14.3 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 how any instance of that class will be graphically represented:

<MyWidget>:

Rules use indentation for delimitation, as python, indentation should be of four spaces per level, like the python good practice recommendations.

There are three keywords specific to 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

## 14.4 Special syntaxes

There are two special syntax to define values for the whole Kv context:

To import something from python:

```
#:import name x.y.z
```

Is equivalent to:

```
from x.y import z as name
```

in python.

To set a global value:

```
#:set name value
```

Is equivalent to:

```
name = value
```

in python.

## 14.5 Instantiate children

To declare the widget has a child widget, instance of some class, just declare this child inside the rule:

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

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

A python equivalent of this code could 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)
```

## 14.6 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)
```

## 14.7 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*.

## 14.8 Referencing Widgets

In a widget tree there is often a need to access/reference other widgets. 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.

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__

```

## 14.9 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))
        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 the ids as defined in the kv language for example:

```
<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: >_<!!!"
```

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))
```

## 14.10 Dynamic Classes

Consider the code below:

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

## 14.11 Re-using styles in multiple widgets

Consider the code below in my.kv:

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

<MySecondWidget>:
```



```

Button:
    on_press: self.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))

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 the style for both widgets. You can do this in .kv as follows. In my.kv:

```

<MyFirstWidget,MySecondWidget>:
    Button:
        on_press: self.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.

## 14.12 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.

### 14.12.1 The code goes in py files

Let's start with a little example. First, the 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.

### 14.12.2 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.

## 15.1 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 serv.py)
- [http://twistedmatrix.com/documents/current/\\_downloads/simple client.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.

### 15.1.1 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 EchoProtocol(protocol.Protocol):
    def dataReceived(self, data):
        response = self.factory.app.handle_message(data)
        if response:
            self.transport.write(response)

class EchoFactory(protocol.Factory):
    protocol = EchoProtocol
    def __init__(self, app):
        self.app = app

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

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

    def handle_message(self, msg):
        self.label.text = "received: %s\n" % msg

        if msg == "ping": msg = "pong"
        if msg == "plop": msg = "kivy rocks"
        self.label.text += "responded: %s\n" % msg
        return msg

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

```

## 15.1.2 Client App

```

#install_twisted_reactor must be called before importing the reactor
from kivy.support import 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)

class EchoFactory(protocol.ClientFactory):
    protocol = EchoClient
    def __init__(self, app):
        self.app = app

    def clientConnectionLost(self, conn, reason):
        self.app.print_message("connection lost")

```

```

def clientConnectionFailed(self, conn, 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

    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')
        self.layout = BoxLayout(orientation='vertical')
        self.layout.add_widget(self.label)
        self.layout.add_widget(self.textbox)
        return self.layout

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

    def on_connection(self, connection):
        self.print_message("connected succesfully!")
        self.connection = connection

    def send_message(self, *args):
        msg = self.textbox.text
        if msg and self.connection:
            self.connection.write(str(self.textbox.text))
            self.textbox.text = ""

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

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

```





# BEST PRACTICES

## 16.1 Designing your Application code

## 16.2 Handle Window re-sizing

## 16.3 Managing resources

- Atlas
- **Cache**
  - Images
  - Text

## 16.4 Platform consideration

## 16.5 Tips and Tricks

- Skinning
- **Using Modules**
  - Monitor
  - Inspector
  - Screen
- Kivy-Remote-Shell



# ADVANCED GRAPHICS

- 17.1 Create your own Shader
- 17.2 Rendering in a Framebuffer
- 17.3 Optimizations



# PACKAGING YOUR APPLICATION

## 18.1 Create a package for Windows

Packaging your application for the Windows platform can only be done inside the Windows OS. The following process has been tested on Windows 7 and the portable package of Kivy.

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

NOTE: Currently, packages for Windows can only be generated with Python 2.7. Python 3.3+ support is on the way...

### 18.1.1 Requirements

- Latest Kivy (the whole portable package, not only the github sourcecode)
- [PyInstaller 2.1](#)

### 18.1.2 Create the spec file

For this example, we'll package the touchtracer example and embed a custom icon. The touchtracer example is the `kivy\examples\demo\touchtracer` directory and the main file is named `main.py`.

1. Double click on the Kivy.bat and a console will open.
2. Go to the pyinstaller 2.1 directory and create the initial spec:

```
cd pyinstaller-2.1
python pyinstaller.py --name touchtracer ..\kivy\examples\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 pyinstaller.py --name touchtracer --icon ..\kivy\examples\demo\touchtracer\icon.ico .
```

For more options, please consult the [PyInstaller 2 Manual](#).

3. The spec file will be `touchtracer.spec` located in inside the pyinstaller + `touchtracer` directory. Now we need to edit the spec file to add kivy hooks to correctly build the exe. Open the spec file with your favorite editor and add theses lines at the beginning of the spec:

```
from kivy.tools.packaging.pyinstaller_hooks import install_hooks
install_hooks(globals())
```

In the *Analysis()* function, remove the *hookspath=None* parameter. If you don't do this, the kivy package hook will not be used at all.

Then you need to change the *COLLECT()* call to add the data for 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:

```
coll = COLLECT( exe, Tree('../kivy/examples/demo/touchtracer/'),
                a.binaries,
                #...
                )
```

4. We are done. Your spec is ready to be executed!

### 18.1.3 Build the spec

1. Double click on *Kivy.bat*
2. Go to the pyinstaller directory, and build the spec:

```
cd pyinstaller-2.1
python pyinstaller.py touchtracer\touchtracer.spec
```

3. The package will be in the *touchtracer\dist\touchtracer* directory.

### 18.1.4 Including Gstreamer

If you wish to use Gstreamer, you'll need to further modify the spec file.

1. Kivy does some magic when trying to find which version of gstreamer and its bindings are available. In order for pyinstaller to find the correct gstreamer modules, you have to import *core.video* in the spec file before doing anything:

```
from kivy.tools.packaging.pyinstaller_hooks import install_hooks
import kivy.core.video
```

2. You'll need to include the gstreamer directory, found in the kivy distribution, in the *COLLECT* call. You can specify the direct path, or get it from the environment. In addition, the contents of the *gstreamer/bin* directory need to be included in the top level directory, otherwise the build process may have trouble finding dlls (this will create a second copy of the contents of bin):

```
import os
gst_plugin_path = os.environ.get('GST_PLUGIN_PATH').split('lib')[0]
COLLECT(exe, Tree(...),
        Tree(gst_plugin_path),
        Tree(os.path.join(gst_plugin_path, 'bin')),
        ...)
```

Following is an example of how to bundle the videoplayer at *kivy/examples/widgets/videoplayer.py*. From *kivy.bat*:

```
cd pyinstaller-2.1
python pyinstaller.py --name gstvideo ..\kivy\examples\widgets\videoplayer.py
```

Now edit the spec file. At the top of the file add:

```
import os
from kivy.tools.packaging.pyinstaller_hooks import install_hooks
import kivy.core.video
```

```
install_hooks(globals())
gst_plugin_path = os.environ.get('GST_PLUGIN_PATH').split('lib')[0]
```

Remove the hookspath parameter, and change:

```
coll = COLLECT(exe,
                a.binaries,
                ...
```

to:

```
coll = COLLECT(exe, Tree('../kivy/examples/widgets'),
                Tree(gst_plugin_path),
                Tree(os.path.join(gst_plugin_path, 'bin')),
                a.binaries,
                ...
```

This will include gstreamer and the example video files in examples/widgets. To build, run:

```
python pyinstaller.py gstvideo/gstvideo.spec
```

Then you should find gstvideo.exe in PyInstaller-2.1/gstvideo/dist/gstvideo, which when run will play a video.

## 18.2 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](#)), use the prebuilt [Kivy Android VM](#) image, 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](#).

NOTE: Currently, packages for Android can only be generated with Python 2.7. Python 3.3+ support is on the way...

### 18.2.1 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 python2.7 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.

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>.

## 18.2.2 Packaging with python-for-android

This section describes how to download and use python-for-android directly.

You'll need:

- A linux computer or a *virtual machine*
- Java
- Python 2.7 (not 2.6.)
- Jinja2 (python module)
- Apache ant
- Android SDK

### Setup Python for Android

First, install the prerequisites needed for the project:

<http://python-for-android.readthedocs.org/en/latest/prerequisites/>

Then open a console and type:

```
git clone git://github.com/kivy/python-for-android
```

### Build your distribution

The distribution is a “directory” containing a specialized python compiled for Android, including only the modules you asked for. You can, from the same python-for-android, compile multiple distributions. For example:

- One containing a minimal support without audio / video
- Another containing audio, openssl etc.

To do that, you must use the script named *distribute.sh*:

```
./distribute.sh -m "kivy"
```

The result of the compilation will be saved into *dist/default*. Here are other examples of building distributions:



```
./distribute.sh -m "openssl kivy"  
./distribute.sh -m "pil ffmpeg kivy"
```

---

**Note:** The order of modules provided are important, as a general rule put dependencies first and then the dependent modules, C libs come first then python modules.

---

To see the available options for `distribute.sh`, type:

```
./distribute.sh -h
```

---

**Note:** To use the latest Kivy development version to build your distribution, link "P4A\_kivy\_DIR" to the kivy folder environment variable to the kivy folder location. On linux you would use the `export` command, like this:

```
export P4A_kivy_DIR=/path/to/cloned/kivy/
```

---

## Package your application

Inside the distribution (*dist/default* by default), you have a tool named *build.py*. This is the script that will create the APK for you:

```
./build.py --dir <path to your app>  
            --name "<title>"  
            --package <org.of.your.app>  
            --version <human version>  
            --icon <path to an icon to use>  
            --orientation <landscape|portrait>  
            --permission <android permission like VIBRATE> (multiple allowed)  
            <debug|release> <installd|installr|...>
```

An example of using multiple permissions:

```
--permission INTERNET --permission WRITE_EXTERNAL_STORAGE
```

Full list of available permissions are documented here: <http://developer.android.com/reference/android/Manifest.permission>

For example, if we imagine that the touchtracer demo of Kivy is in the directory `~/kivy/examples/demo/touchtracer`, you can do:

```
./build.py --dir ~/kivy/examples/demo/touchtracer \  
            --package org.demo.touchtracer \  
            --name "Kivy Touchtracer" --version 1.1.0 debug installd
```

You need to be aware that the default target Android SDK version for the build will be SDK v.8, which is the minimum required SDK version for kivy. You should either install this API version, or change the `AndroidManifest.xml` file (under `dist/.../`) to match your own target SDK requirements.

The debug binary will be generated in `bin/KivyTouchtracer-1.1.0-debug.apk`. The *debug* and *installd* parameters are commands from the Android project itself. They instruct *build.py* to compile the APK in debug mode and install on the first connected device.

You can then install the APK directly to your Android device as follows:

```
adb install -r bin/KivyTouchtracer-1.1.0-debug.apk
```

### 18.2.3 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 <http://kivy.googlecode.com/files/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...

### 18.2.4 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 <http://developer.android.com/guide/publishing/app-signing.html> - all the necessary tools come with the Android SDK.

## 18.2.5 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.

## 18.3 The Kivy Android Virtual Machine

### 18.3.1 Introduction

Currently, Kivy Android applications can only be built in a Linux environment configured with python-for-android, the Android SDK and the Android NDK. As this environment is not only tricky to setup but also impossible on Windows or MacOSX operating systems, we provide a fully configured [VirtualBox](#) disk image to ease your building woes.

If you are not familiar with virtualization, we encourage you to read the [Wikipedia Virtualization page](#).

### 18.3.2 Getting started

1. Download the disc image from [here](#), in the *Virtual Machine* section. It is approximately 1GB. Extract the file and remember the location of the extracted vdi file.
2. Download the version of VirtualBox for your machine from the [VirtualBox download area](#) and install it.
3. Start VirtualBox, click on “New” in the left top. Then select “linux” and “Ubuntu 32”.
4. Under “Hard drive”, choose “Use an existing virtual hard drive file”. Search for your vdi file and select it.
5. Go to the “Settings” for your virtual machine. In the “Display -> Video” section, increase video ram to 32mb or above. Enable 3d acceleration to improve the user experience.
6. Start the Virtual machine and follow the instructions in the readme file on the desktop.

### 18.3.3 Building the APK

Once the VM is loaded, you can follow the instructions from [Packaging with python-for-android](#). You don't need to download with *git clone* though, as python-for-android is already installed and set up in the virtual machine home directory.

### 18.3.4 Hints and tips

#### 1. Shared folders

Generally, your development environment and toolset are set up on your host machine but the APK is built in your guest. VirtualBox has a feature called ‘Shared folders’ which allows your guest direct access to a folder on your host.

It is often convenient to use this feature (usually with ‘Permanent’ and ‘Auto-mount’ options) to copy the built APK to the host machine so it can form part of your normal dev environment. A simple script can easily automate the build and copy/move process.

#### 2. Copy and paste

By default, you will not be able to share clipboard items between the host and the guest machine. You can achieve this by enabling the “bi-directional” shared clipboard option under “Settings -> General -> Advanced”.

### 3. Snapshots

If you are working on the Kivy development branch, pulling the latest version can sometimes break things (as much as we try not to). You can guard against this by taking a snapshot before pulling. This allows you to easily restore your machine to it’s previous state should you have the need.

## 18.4 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.

### 18.4.1 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.

### 18.4.2 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 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.

### 18.4.3 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 NotImplemented-Error 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.

### 18.4.4 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.

## 18.5 Creating packages for MacOSX

Packaging your application for the MacOSX 10.6 platform can only be done inside MacOSX. The following method has only been tested inside VirtualBox and MacOSX 10.6, using the portable package of Kivy.

The package will only work for the 64 bit MacOSX. We no longer support 32 bit MacOSX platforms.

NOTE: Currently, packages for OSX can only be generated with Python 2.7. Python 3.3+ support is on the way...

### 18.5.1 Requirements

- Latest Kivy (the whole portable package, not only the github sourcecode)
- [PyInstaller 2.0](#)

Please ensure that you have installed the Kivy DMG and installed the *make-symlink* script. The *kivy* command must be accessible from the command line.

Thereafter, download and decompress the PyInstaller 2.0 package.

**Warning:** It seems that the latest PyInstaller has a bug affecting Mach-O binaries. (<http://www.pyinstaller.org/ticket/614>). To correct the issue, type:

```
cd pyinstaller-2.0/PyInstaller/lib/macholib
curl -O https://bitbucket.org/ronaldoussoren/macholib/raw/e32d04b5361950a9343ca453d75602b65787f2
```

In version 2.1, the issue has already been corrected.

## 18.5.2 Create the spec file

As an example, we'll package the touchtracer demo, using a custom icon. The touchtracer code is in the `../kivy/examples/demo/touchtracer/` directory, and the main file is named `main.py`. Replace both path/filename according to your system.

1. Open a console.
2. Go to the pyinstaller directory, and create the initial specs:

```
cd pyinstaller-2.0
kivy pyinstaller.py --windowed --name touchtracer ../kivy/examples/demo/touchtracer/main.py
```

3. The specs file is named `touchtracer/touchtracer.spec` and located inside the pyinstaller directory. Now we need to edit the spec file to add kivy hooks to correctly build the executable. Open the spec file with your favorite editor and put these lines at the start of the spec:

```
from kivy.tools.packaging.pyinstaller_hooks import install_hooks
install_hooks(globals())
```

In the `Analysis()` method, remove the `hookspath=None` parameter. If you don't do this, the kivy package hook will not be used at all.

Then, 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:

```
coll = COLLECT( exe, Tree('../kivy/examples/demo/touchtracer/'),
                a.binaries,
                #...
                )
```

4. We are done. Your spec is ready to be executed!

## 18.5.3 Build the spec and create a DMG

1. Open a console.
2. Go to the pyinstaller directory, and build the spec:

```
cd pyinstaller-2.0
kivy pyinstaller.py touchtracer/touchtracer.spec
```

3. The package will be the `touchtracer/dist/touchtracer` directory. Rename it to `.app`:

```
pushd touchtracer/dist
mv touchtracer touchtracer.app
hdiutil create ./Touchtracer.dmg -srcfolder touchtracer.app -ov
popd
```

4. You will now have a `Touchtracer.dmg` available in the `touchtracer/dist` directory.

## 18.5.4 Including Gstreamer

If you want to read video files, audio, or camera, you will need to include gstreamer. By default, only `pygst/gst` files are discovered, but all the `gst` plugins and libraries are missing. You need to include them in your `.spec` file too, by adding one more arguments to the `COLLECT()` method:

```
import os
gst_plugin_path = os.environ.get('GST_PLUGIN_PATH').split(':')[0]

coll = COLLECT( exe, Tree('../kivy/examples/demo/touchtracer/'),
                Tree(os.path.join(gst_plugin_path, '..')),
                a.binaries,
                #...
                )
```

For Kivy.app < 1.4.1, you also need to update one script included in our Kivy.app. Go to */Applications/Kivy.app/Contents/Resources/kivy/kivy/tools/packaging/pyinstaller\_hooks/*, and edit the file named *rt-hook-kivy.py*, and add this line at the end:

```
environ['GST_PLUGIN_PATH'] = join(root, '..', 'gst-plugins')
```

## 18.6 Create a package for IOS

New in version 1.2.0.

**Warning:** This process is still under development.

The overall process for creating a package for IOS can be explained in 4 steps:

1. Compile python + modules for IOS
2. Create an Xcode project
3. Populate the Xcode project with your application source code
4. Customize

This process has been tested with Xcode 4.2.

NOTE: Currently, packages for iOS can only be generated with Python 2.7. Python 3.3+ support is on the way...

### 18.6.1 Prerequisites

You need to install some dependencies, like cython or mercurial. If you're using Xcode 4.3, then you also need to install autotools. We encourage you to use [Homebrew](#) to install those dependencies:

```
brew install autoconf automake libtool pkg-config mercurial
brew link libtool
brew link mercurial
sudo easy_install pip
sudo pip install cython
```

For more detail, see [IOS Prerequisites](#). Just ensure that everything is ok before starting the second step!

### 18.6.2 Compile the distribution

Open a terminal, and type:

```
$ git clone git://github.com/kivy/kivy-ios
$ cd kivy-ios
$ tools/build-all.sh
```



If you don't want to compile all the things needed for kivy, edit and change `tools/build-all.sh` to your needs.

Most of the python distribution will be packed into a `python27.zip`. If you experience any issues or would like more detail on this process, please refer to [Compiling for IOS](#).

### 18.6.3 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:

```
$ tools/create-xcode-project.sh test /path/to/your/appdir
```

---

**Note:** You must use a fully qualified path to your application directory.

---

Now you can open the Xcode project:

```
$ open app-test/test.xcodeproj
```

### 18.6.4 Customize

You can customize the build in many ways:

1. Minimize the `build/python/lib/python27.zip`: this contains all the python modules. You can edit the zip file and remove all the files you'll not use (reduce encodings, remove xml, email...)
2. Remove the .a not used: in Xcode, select your target, go in *Build Phases*, then check the *Link Binary With Libraries*. You can remove the libraries not used by your application.
3. Change the icon, orientation, etc... According to the Apple policy :)
4. Go to the settings panel > build, search for "strip" options, and triple-check that they are all set to NO. Stripping does not work with Python dynamic modules and will remove needed symbols.
5. Indicate a launch image in portrait/landscape for iPad with and without retina display.

Launch Images are supported. By default, XCode want you to build an [Image Sets](#). This is your responsibility to fill all the images needed for the Sets, depending of your target. However, Kivy use SDL, and as soon as the application starts the SDL main, the launch image will disappear. To prevent that, you need to have 2 files named `Default.png` and `Default-Landscape.png`, and put them in the *Resources* folder in Xcode (not in your application folder)

### 18.6.5 Known issues

Currently, the project has a few known issues (we'll fix these in future versions):

- You can't export your project outside the kivy-ios directory because the libraries included in the project are relative to it.
- Removing some libraries (like `SDL_Mixer` for audio) is currently not possible because the kivy project requires it.
- And more, just too technical to be written here.

## 18.6.6 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](#).

## 18.7 IOS Prerequisites

The following guide assumes:

- XCode 5.1
- MacOSX 10.9

Your experience may vary with different versions.

### 18.7.1 Getting started

In order to submit any application to the iTunes store, you will need an [iOS Developer License](#). For testing, you will need an actual device as kivy does not yet support the iOS emulators that Apple supplies.

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.

### 18.7.2 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
```

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](#)

## 18.8 Compiling for iOS

(work in progress)

### 18.8.1 Creating your distribution

Kivy uses a shell script to build your distribution and package all its contents such that it can be used by XCode to compile your final iOS program (an *.ipa* file).

This process involves running all the required code and libraries through a compiler and linker in order to create a single, stand-alone set of binaries and source files. These files comprise your *distribution*.

### 18.8.2 Using the “build\_all.sh” script

The kivy-ios package provides a generic script, “tools/build\_all.sh”, that creates a complete distribution for you.

You may want edit/copy this file in order to customize your distribution for various reasons:

```
* Minimize the size of your package by removing unused libraries
* Customize the packing by adding/removing script items
* Troubleshooting
```

The minimizing and customizing options are obviously desirable and relatively simple as the build script is a standard [bash shell script](#).

### 18.8.3 The build process

Initially, your kivy-ios checkout will contain two folders: *tools* and *src*. The first time you run it, the script will download the latest versions of the packages kivy-ios uses. This means it might fail if any packages are not available or cannot be downloaded.

These downloaded packages are stored in a hidden *.cache* subfolder. The build process then extracts these files to a *tmp* subfolder, builds the packages and places the build in the *build* subfolder. Be careful: if this process is interrupted, it might leave corrupt files in any of these locations.

If you wish to force a fresh build of all the packages, you should delete all of these other folders (*.cache*, *tmp* and *build*) and re-run the ‘build\_all.sh’ script.

## 18.8.4 Troubleshooting

### Isolating problems

The kivy-ios project uses many libraries which may change and break things independently of kivy. It may thus sometimes be necessary to remove any packages which do not compile in order to complete your build or isolate the offending package.

The 'build-all.sh' script assembles many sub-scripts into one, comprehensive build script. If you open this file, you will see something similar to:

```
#!/bin/bash

. $(dirname $0)/environment.sh

try $(dirname $0)/build-libffi.sh
try $(dirname $0)/build-python.sh
try $(dirname $0)/reduce-python.sh
...
```

You can comment out problematic scripts using the hash (#) symbol. Some scripts are essential (e.g. *build-python.sh*), but others can be safely removed if your application does not require them.

We hope you never have to care about this, but if you encounter an error which you cannot resolve, this may help. Remember, you can always [contact us](#) for help.

### Clang compiler issues

Some dependencies for compiling cython with pip on OSX may fail to compile with the Clang (Apple's C) compiler displaying the message:

```
clang: error: unknown argument: '-mno-fused-madd' [-Wunused-command-line-argument-hard-error-in-future]
clang: note: this will be a hard error (cannot be downgraded to a warning) in the future
error: command 'cc' failed with exit status 1
```

Here is a workaround:

```
export CFLAGS=-Qunused-arguments
sudo -E pip install cython
```

The -E flag passes the environment to the sudo shell.

## 18.8.5 Further reading

Kivy iOS support is a work-in-progress and we are busy trying to improve and document this process. Until such time as this is complete, you may find the following links useful.

- [iOS Tips](#)
- [HTTPS \(SSL\) support](#)

Part III  
TUTORIALS

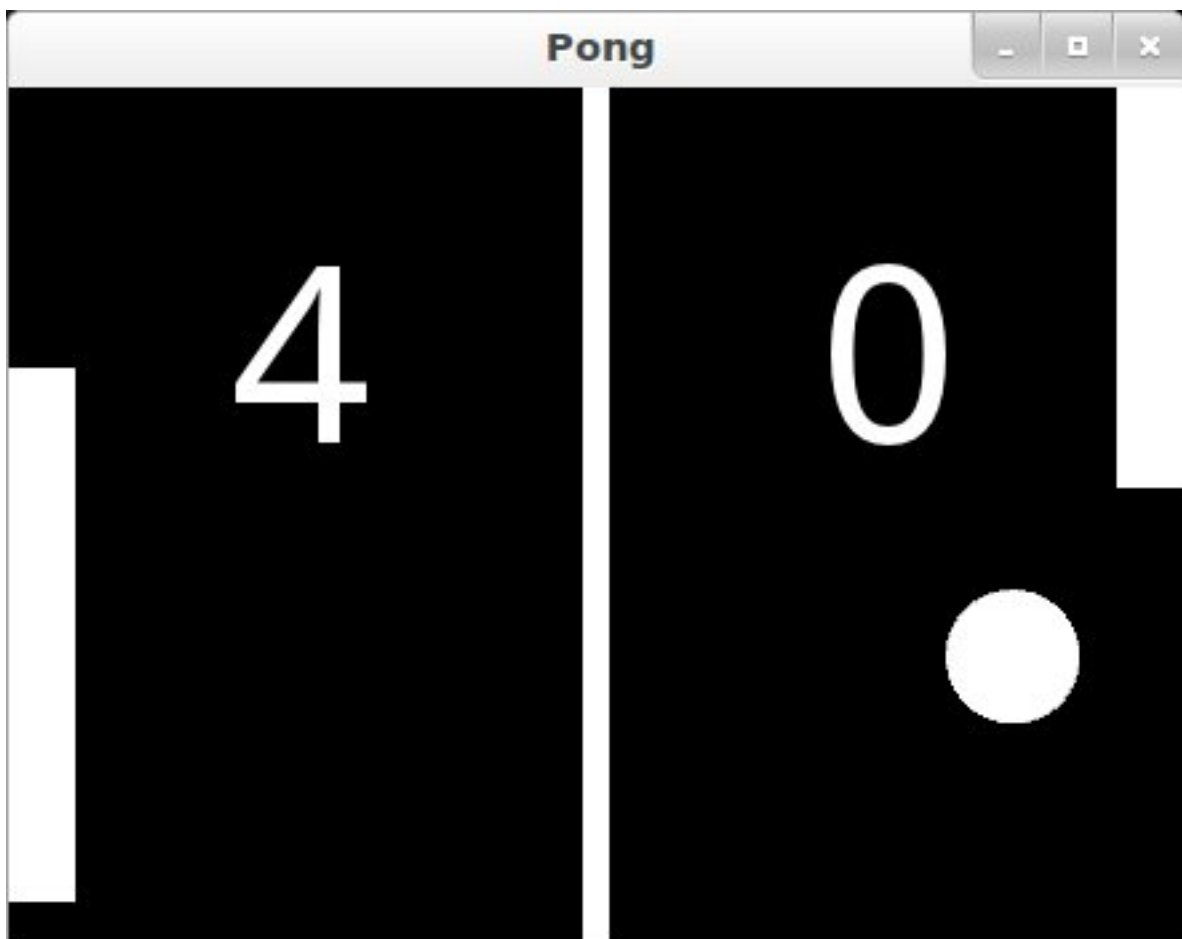


# PONG GAME TUTORIAL

## 19.1 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* 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!

## 19.2 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.

## 19.3 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
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.

### 19.3.1 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 widgets' 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"
```

## 19.4 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.

#### 19.4.1 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

```

Note that not only a `<PongBall>` widget rule has been added, but also a child widget `PongBall` in the `<PongGame>` widget rule.

## 19.5 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.

### 19.5.1 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).

### 19.5.2 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 players 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 your 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 apps `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 NumericProperty, ReferenceListProperty,\
4     ObjectProperty
5 from kivy.vector import Vector
6 from kivy.clock import Clock
7 from random import randint
8
9
10 class PongBall(Widget):
11     velocity_x = NumericProperty(0)
12     velocity_y = NumericProperty(0)

```

```

13     velocity = ReferenceListProperty(velocity_x, velocity_y)
14
15     def move(self):
16         self.pos = Vector(*self.velocity) + self.pos
17
18
19     class PongGame(Widget):
20         ball = ObjectProperty(None)
21
22         def serve_ball(self):
23             self.ball.center = self.center
24             self.ball.velocity = Vector(4, 0).rotate(randint(0, 360))
25
26         def update(self, dt):
27             self.ball.move()
28
29             #bounce off top and bottom
30             if (self.ball.y < 0) or (self.ball.top > self.height):
31                 self.ball.velocity_y *= -1
32
33             #bounce off left and right
34             if (self.ball.x < 0) or (self.ball.right > self.width):
35                 self.ball.velocity_x *= -1
36
37
38     class PongApp(App):
39         def build(self):
40             game = PongGame()
41             game.serve_ball()
42             Clock.schedule_interval(game.update, 1.0 / 60.0)
43             return game
44
45
46     if __name__ == '__main__':
47         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     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

```

## 19.6 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)

```

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 NumericProperty, ReferenceListProperty,\
4     ObjectProperty
5 from kivy.vector import Vector
6 from kivy.clock import Clock
7
8
9 class PongPaddle(Widget):
10     score = NumericProperty(0)
11
12     def bounce_ball(self, ball):
13         if self.collide_widget(ball):
14             vx, vy = ball.velocity
15             offset = (ball.center_y - self.center_y) / (self.height / 2)
16             bounced = Vector(-1 * vx, vy)
17             vel = bounced * 1.1
18             ball.velocity = vel.x, vel.y + offset
19
20
21 class PongBall(Widget):
22     velocity_x = NumericProperty(0)
23     velocity_y = NumericProperty(0)
24     velocity = ReferenceListProperty(velocity_x, velocity_y)
25
26     def move(self):
27         self.pos = Vector(*self.velocity) + self.pos
28
29
30 class PongGame(Widget):
31     ball = ObjectProperty(None)
32     player1 = ObjectProperty(None)
33     player2 = ObjectProperty(None)
34
35     def serve_ball(self, vel=(4, 0)):
36         self.ball.center = self.center
37         self.ball.velocity = vel
38
39     def update(self, dt):
40         self.ball.move()
41
42         #bounce of paddles
43         self.player1.bounce_ball(self.ball)
44         self.player2.bounce_ball(self.ball)
45
46         #bounce ball off bottom or top
47         if (self.ball.y < self.y) or (self.ball.top > self.top):
48             self.ball.velocity_y *= -1
49
50         #went of to a side to score point?
51         if self.ball.x < self.x:
52             self.player2.score += 1
53             self.serve_ball(vel=(4, 0))
54         if self.ball.x > self.width:
55             self.player1.score += 1
56             self.serve_ball(vel=(-4, 0))
57
58     def on_touch_move(self, touch):
```



```

59     if touch.x < self.width / 3:
60         self.player1.center_y = touch.y
61     if touch.x > self.width - self.width / 3:
62         self.player2.center_y = touch.y
63
64
65     class PongApp(App):
66         def build(self):
67             game = PongGame()
68             game.serve_ball()
69             Clock.schedule_interval(game.update, 1.0 / 60.0)
70             return game
71
72
73     if __name__ == '__main__':
74         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:
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
```

## 19.7 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 for it)
- 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` & `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.

## 20.1 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](#)).

## 20.2 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!

## 20.2.1 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.

---

## 20.2.2 Adding Behaviour

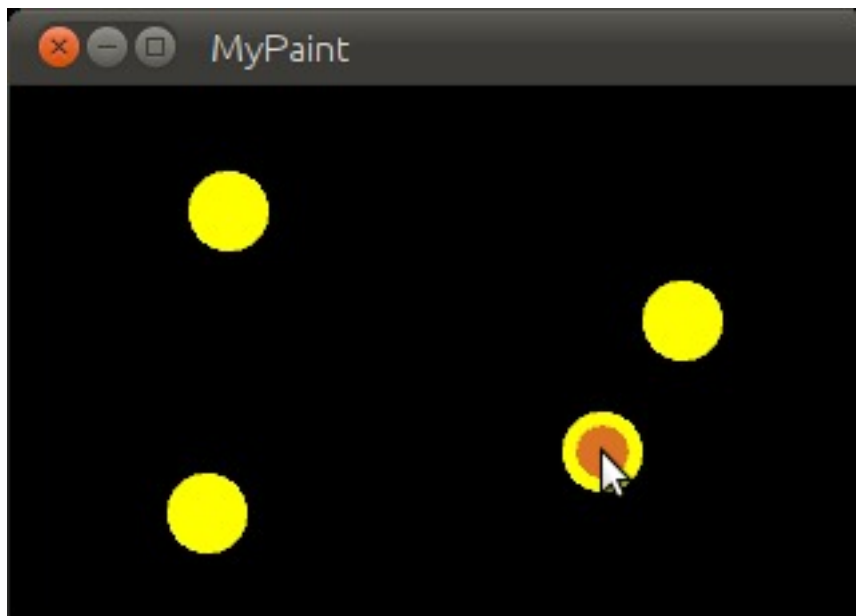
Let's now add some actual behaviour to the widget, i.e. make it react to user input. Change the code like so:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3
4
5 class MyPaintWidget(Widget):
6     def on_touch_down(self, touch):
7         print(touch)
8
9 class MyPaintApp(App):
10    def build(self):
11        return MyPaintWidget()
12
13
14 if __name__ == '__main__':
15    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 8: 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 9: 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 10: 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 11: 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):
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:

```

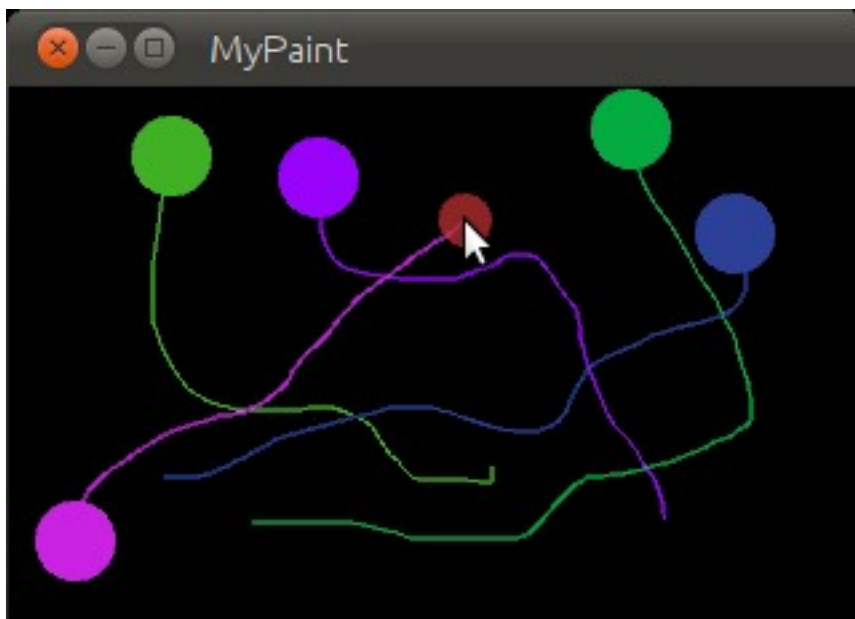
1 from random import random
2 from kivy.app import App
3 from kivy.uix.widget import Widget
4 from kivy.graphics import Color, Ellipse, Line
5
6

```

```

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.

---

### 20.2.3 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.

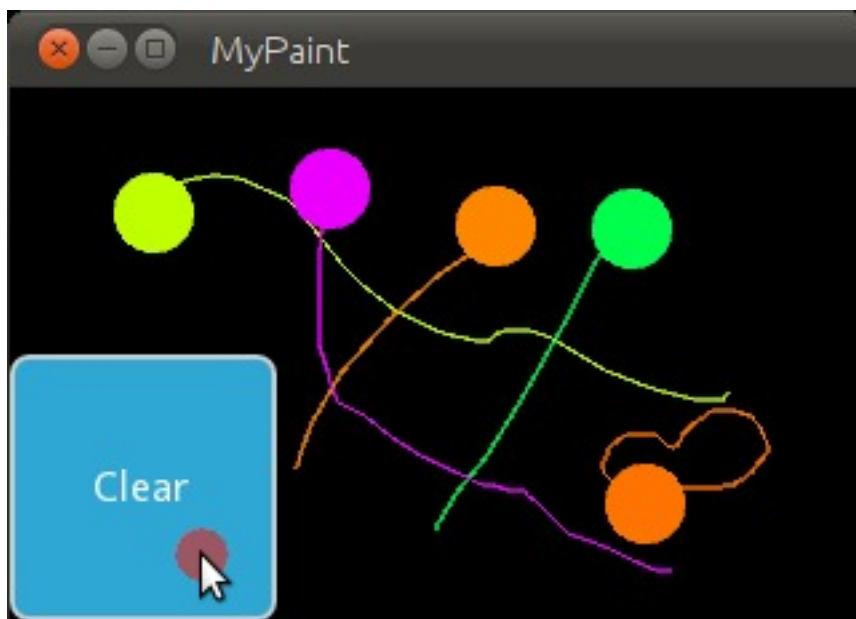
For our simple example, it doesn't really matter that much. For larger applications you should give some thought to who does what in your app. We'll go with the second option here so that you see how you can build up your application's widget tree in your app class's `build()` method. We'll also change 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')
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         painter = MyPaintWidget()
```

```

27 clearbtn = Button(text='Clear')
28 parent.add_widget(painter)
29 parent.add_widget(clearbtn)
30
31 def clear_canvas(obj):
32     painter.canvas.clear()
33     clearbtn.bind(on_release=clear_canvas)
34
35     return parent
36
37
38 if __name__ == '__main__':
39     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 & 29: 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.
- Lines 31 & 32: 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.
- Line 33: We bind the button's `on_release` event (which is fired when the button is pressed and then released) to the callback we just defined.

---

**Note:** The Kivy `Widget` class, by design, is kept simple. There are no general properties such as back-

---

ground 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.



## 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 completely cross-platform (Linux/OSX/Win) 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 in the C language.

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', 'alpha', or 'beta'. The tagrevision is the revision 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.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 = {'window': ('egl_rpi', 'pygame', 'sdl', 'x11'), 'camera': ('opencv', 'gi', 'pygst', 'videocapture',
    Global settings options for kivy

kivy.kivy_base_dir = '/home/kivy/Buildbot-2.7/doc/build/kivy'
    Kivy directory

kivy.kivy_modules_dir = '/home/kivy/Buildbot-2.7/doc/build/kivy/modules'
    Kivy modules directory

kivy.kivy_data_dir = '/home/kivy/Buildbot-2.7/doc/build/kivy/data'
    Kivy data directory

kivy.kivy_shader_dir = '/home/kivy/Buildbot-2.7/doc/build/kivy/data/glsl'
    Kivy glsl shader directory

kivy.kivy_icons_dir = '/home/kivy/Buildbot-2.7/doc/build/kivy/data/icons/'
    Kivy icons config path (don't remove the last '/')

kivy.kivy_home_dir = ""
    Kivy user-home storage directory

kivy.kivy_userexts_dir = ""
    Kivy user extensions directory

kivy.kivy_config_fn = ""
    Kivy configuration filename

kivy.kivy_usermodules_dir = ""
    Kivy user modules directory

```

## 21.1 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

### 21.1.1 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).

### 21.1.2 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.



### 21.1.3 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)
```

### 21.1.4 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)
```

### 21.1.5 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 1 / 60.

#### Events

**on\_start:** widget Fired when the animation is started on a widget.

**on\_complete:** widget Fired when the animation is completed or stopped on a widget.

**on\_progress:** widget, progression Fired when the progression of the animation is changing.

Changed in version 1.4.0: Added s/step parameter.

#### 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.

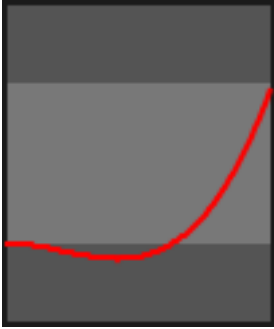
## **class kivy.animation.AnimationTransition**

Bases: **object**

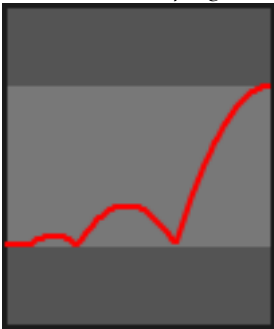
Collection of animation functions to be used with the Animation object. Easing Functions ported to Kivy from the Clutter Project <http://www.clutter-project.org/docs/clutter/stable/ClutterAlpha.html>

The *progress* parameter in each animation function is in the range 0-1.

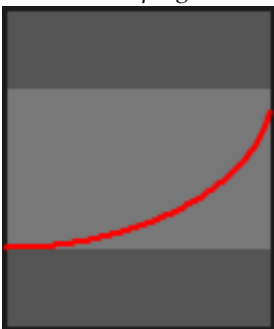
`static in_back(progress)`



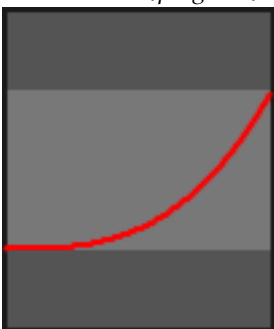
`static in_bounce(progress)`



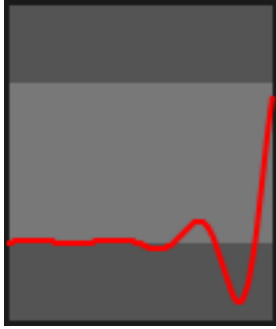
`static in_circ(progress)`



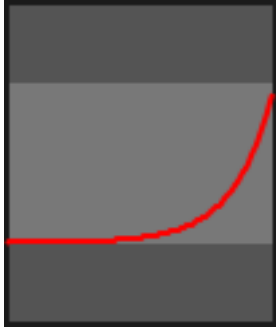
`static in_cubic(progress)`



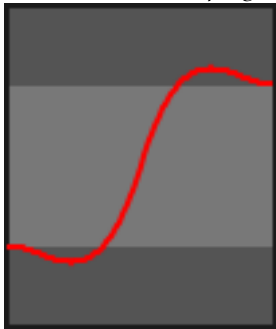
`static in_elastic(progress)`



`static in_expo(progress)`



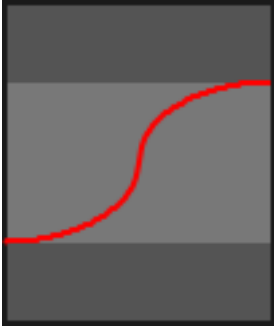
`static in_out_back(progress)`



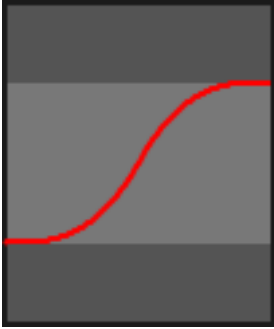
`static in_out_bounce(progress)`



`static in_out_circ(progress)`



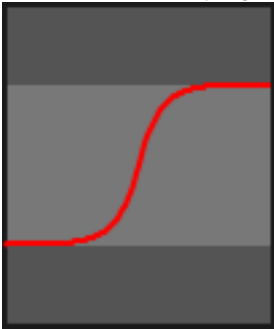
static **in\_out\_cubic**(*progress*)



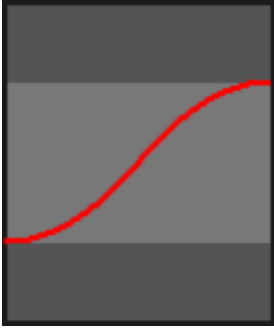
static **in\_out\_elastic**(*progress*)



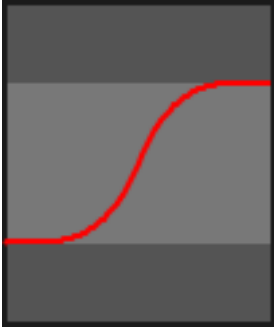
static **in\_out\_expo**(*progress*)



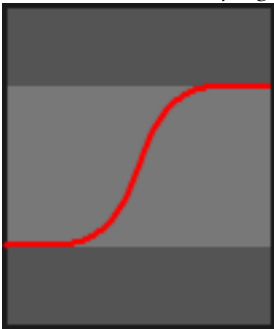
static **in\_out\_quad**(*progress*)



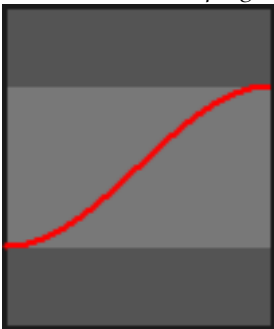
static **in\_out\_quart**(*progress*)



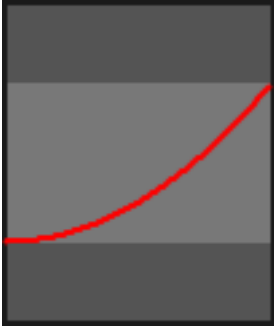
static **in\_out\_quint**(*progress*)



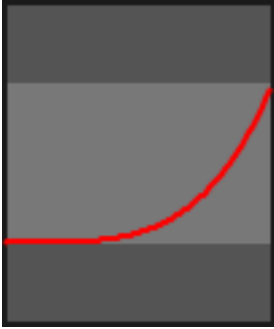
static **in\_out\_sine**(*progress*)



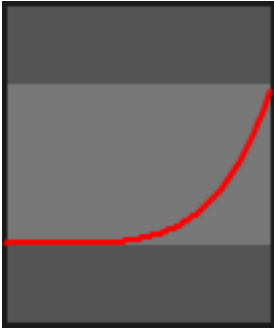
static **in\_quad**(*progress*)



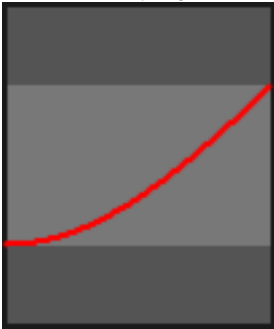
`static in_quart(progress)`



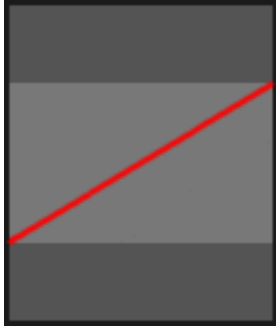
`static in_quint(progress)`



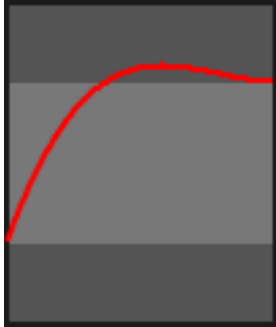
`static in_sine(progress)`



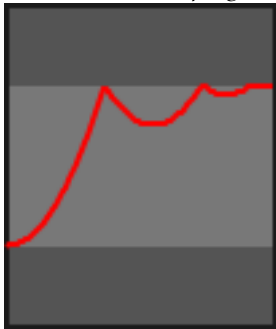
`static linear(progress)`



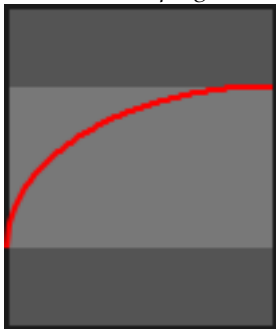
**static out\_back**(*progress*)



**static out\_bounce**(*progress*)

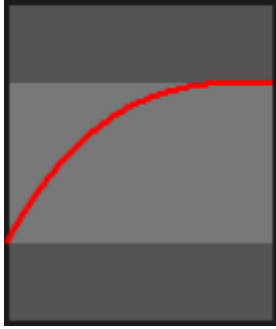


**static out\_circ**(*progress*)

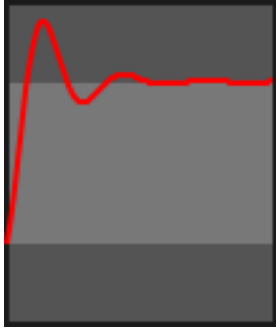


**static out\_cubic**(*progress*)

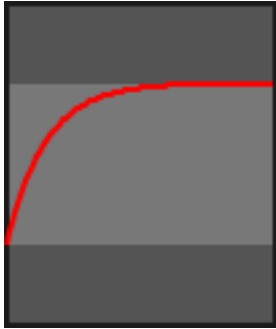




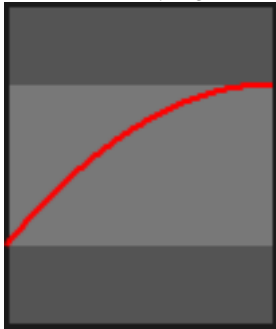
`static out_elastic(progress)`



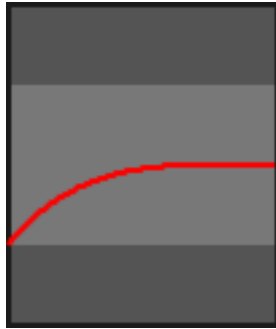
`static out_expo(progress)`



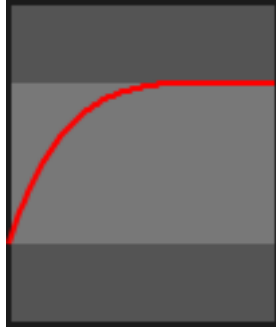
`static out_quad(progress)`



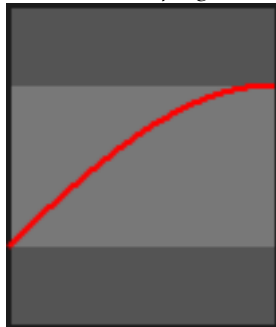
`static out_quart(progress)`



`static out_quint(progress)`



`static out_sine(progress)`



## 21.2 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.

### 21.2.1 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 world'
```

Contents of 'main.py':

```
'''  
Application from a .kv  
=====  
The root application is created from the corresponding .kv. Check the test.kv  
file to see what will be the 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 relation between `main.py` and `test.kv` is explained in `App.load_kv()`.

## 21.2.2 Application configuration

New in version 1.0.7.

Use the configuration file

Your application might want to have its own configuration file. The `App` is able to handle an INI file automatically. You add your section/key/value in the `App.build_config()` method by using the `config` parameter (which is 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 in the config, a file is created on the disk and named from the mangled name of your class. “TestApp” will give a config file-name “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",
    "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 have been changed to', value)
            elif token == ('section1', 'key2'):
                print('Our key2 have 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
```

### 21.2.3 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.

### 21.2.4 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.

### 21.2.5 Pause mode

New in version 1.1.0.

**Warning:** This mode is experimental, and designed for phones/tablets. There are some cases where your application could crash on resume.

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 (default case), then `App.on_stop()` is called.
4. Otherwise 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.

```

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`.

#### **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(\*args)**

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/%s.ini')**

New in version 1.0.7.

Changed in version 1.4.0: Customized the default path for iOS and Android platforms. Added a defaultpath parameter for desktop OS's (not applicable to iOS and Android.)

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: /sdcard/.<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 tilda '~' 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**

#### **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'
```

---

#### **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.

## 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 (the default).

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 False.

New in version 1.1.0.

### **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(\*largs)**

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 1.8.1.

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 to 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 Android, `/sdcard/<app_name>` is returned.

On Windows, `%APPDATA%/<app_name>` is returned.

On Mac OSX, `~/Library/Application Support/<app_name>` is returned.

On Linux, `$XDG_CONFIG_HOME/<app_name>` is returned.

## 21.3 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')
```

## 21.3.1 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: `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.

## 21.4 Atlas

New in version 1.1.0.

Atlas is a class for managing texture atlases: packing multiple textures into one. With it, you reduce the number of images loaded and speedup the application loading.

An Atlas is composed of:

- a json file (.atlas) that contains all the information about the images contained inside the atlas.

- one or multiple atlas images associated with the atlas definition.

### 21.4.1 Definition of .atlas

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 of the Kivy `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 ...
  }
}
```

### 21.4.2 How to create an Atlas

**Warning:** The atlas creation requires Imaging/PIL. This will be removed in the future when the Kivy core Image is able to support loading / blitting / 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 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 256 *.png
Atlas created at myatlas.atlas
1 image have 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`.

**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`

---

### 21.4.3 How to use an Atlas

Usually, you would use the atlas as follows:

```
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:

```
atlas://path/to/myatlas/id
# will search for the "path/to/myatlas.atlas" and get the image "id"
```

In our case, it would be:

```
atlas://images/myatlas/button
```

---

**Note:** In the atlas url, there is no need to add the `.atlas` extension. It will be automatically append to the filename.

---

### 21.4.4 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.

**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  $\geq 2$ , 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.

### textures

List of available textures within the atlas.

**textures** is a **DictProperty** and defaults to {}.

## 21.5 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
text = 'objectid'
instance = Label(text=text)
Cache.append('mycache', text, instance)

# retrieve the cached object
instance = Cache.get('mycache', label)
```

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: 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.

## 21.6 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 removed.

---

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()
```

---

## 21.6.1 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
```

## 21.6.2 Triggered Events

New in version 1.0.5.

A triggered event is a way to defer a callback exactly like `schedule_once()`, but with some added convenience. The callback will only be scheduled once per frame even if you call the trigger twice (or more). 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
t = Clock.create_trigger(my_callback)
t()
t()
```

Before triggered events, you may have used this approach in a widget:

```
def trigger_callback(self, *largs):
    Clock.unschedule(self.callback)
    Clock.schedule_once(self.callback)
```

As soon as you call `trigger_callback()`, it will correctly schedule the callback once in the next frame. 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.

---

**Note:** `ClockBase.create_trigger()` also has a timeout parameter that behaves exactly like `ClockBase.schedule_once()`.

---

### 21.6.3 Threading

New in version 1.8.1.

Often, other threads are used to schedule callbacks with kivy's main thread using `ClockBase`. Therefore, it's important to know what is thread safe and what isn't.

All the `ClockBase` and `ClockEvent` methods are safe with respect to kivy's thread. That is, it's always safe to call these methods from a single thread that is not the kivy thread. However, there are no guarantees as to the order in which these callbacks will be executed.

Calling a previously created trigger from two different threads (even if one of them is the kivy thread), or calling the trigger and its `ClockEvent.cancel()` method from two different threads at the same time is not safe. That is, although no exception will be raised, there are no guarantees that calling the trigger from two different threads will not result in the callback being executed twice, or not executed at all. Similarly, such issues might arise when calling the trigger and canceling it with `ClockBase.unschedule()` or `ClockEvent.cancel()` from two threads simultaneously.

Therefore, it is safe to call `ClockBase.create_trigger()`, `ClockBase.schedule_once()`, `ClockBase.schedule_interval()`, or call or cancel a previously created trigger from an external thread. The following code, though, is not safe because it calls or cancels from two threads simultaneously without any locking mechanism:

```

event = Clock.create_trigger(func)

# in thread 1
event()
# in thread 2
event()
# now, the event may be scheduled twice or once

# the following is also unsafe
# in thread 1
event()
# in thread 2
event.cancel()
# now, the event may or may not be scheduled and a subsequent call
# may schedule it twice

```

Note, in the code above, thread 1 or thread 2 could be the kivy thread, not just an external thread.

`kivy.clock.Clock = None`  
 Instance of `ClockBase`.

`class kivy.clock.ClockBase`  
 Bases: `kivy.clock._ClockBase`  
 A clock object with event support.

**create\_trigger**(*callback, timeout=0*)

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.

**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.

**max\_iteration**

New in version 1.0.5: When a `schedule_once` is used with -1, you can add a limit on how iteration will be allowed. That is here to prevent too much relayout.

**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()`).

**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.

**unschedule**(*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 is scheduled.

**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 1.8.1: The *all* parameter was added. Before, it behaved as if *all* was *True*.

**class** `kivy.clock.ClockEvent` (*clock, loop, callback, timeout, starttime, cid, trigger=False*)

Bases: `object`

A class that describes a callback scheduled with kivy's **Clock**. This class is never created by the user; instead, kivy creates and returns an instance of this class when scheduling a callback.

**Warning:** Most of the methods of this class are internal and can change without notice. The only exception are the `cancel()` and `__call__()` methods.

## `cancel()`

Cancels the callback if it was scheduled to be called.

**class** `kivy.clock.mainthread` (*func*)

Decorator that will schedule the call of the function 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 call in the main thread')

self.req = UrlRequest(url='http://...', on_success=callback)
```

New in version 1.8.0.

## 21.7 Compatibility module for Python 2.7 and > 3.3

`kivy.compat.PY2 = True`

True if Python 2 interpreter is used

`kivy.compat.string_types`

String types that can be used for checking if a object is a string

alias of `basestring`

## 21.8 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.



Note: To avoid instances where the config settings do not work or they are not applied before window creation (like setting an initial window size), `Config.set` should be used before importing any modules that affect the application window (ie. importing `Window`). Ideally, these settings should be declared right at the start of your `main.py` script.

### 21.8.1 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()
```

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.

### 21.8.2 Available configuration tokens

#### kivy

**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.

**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.

**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 '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.

**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

**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` and 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.

**'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** Show the cursor on the screen.

**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`.

**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.

**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 1.8.0: `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**

Kivy configuration object. Its `name` is `'kivy'`

**class kivy.config.ConfigParser(name='')**

Bases: `ConfigParser.ConfigParser`, `object`

Enhanced `ConfigParser` class that supports the addition of default sections and default values.

By default, the kivy `ConfigParser` instance, `Config`, is given the name `'kivy'` and the `ConfigParser` instance used by `App`, `build_settings()`, is given the name `'app'`.

**Parameters**

**name: string**The name of the instance. See **name**. Defaults to "".

**..versionchanged:: 1.8.1**Each ConfigParser can now be named, **name**. 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/key changed. If you don't specify a section or a key, it will call the callback for all section/keys 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.

**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 an option. If not found, it will return the default value.

**getdefaultint**(*section*, *option*, *defaultvalue*)

Get an option. If not found, it will return the default value. The return value will be always converted 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 dynamically changed 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 1.8.1: **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 1.8.1.

**set**(*section*, *option*, *value*)

Functions similarly to PythonConfigParser's set method, except that the value is implicitly converted to a string.

**setall**(*section*, *keyvalues*)

Set a lot of keys/values in one section at the same time.

**setdefault** (*section, option, value*)

Set the default value of a particular option.

**setdefaults** (*section, keyvalues*)

Set a lot of keys/value defaults in one section at the same time.

**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.

## 21.9 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.

`kivy.context.register_context` (*name, cls, \*args, \*\*kwargs*)

Register a new context.

`kivy.context.get_current_context` ()

Return the current context.

## 21.10 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`

Bases: `kivy.event.ObjectWithUid`

Generic event dispatcher interface.

See the module docstring for usage.

**bind**()

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):
        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()

```

### create\_property()

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 1.8.1: In the past, if *value* was of type *bool*, a *NumericProperty* would be created, now a *BooleanProperty* is created.

**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.

```

>>> mywidget = Widget()
>>> mywidget.create_property('custom')
>>> mywidget.custom = True
>>> print(mywidget.custom)
True

```

### dispatch()

Dispatch an event across all the handlers added in `bind()`. 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*: **str**the event name to dispatch.

Changed in version 1.8.1: Keyword arguments collection and forwarding was added. Before, only positional arguments would be collected and forwarded.

### **events()**

Return all the events in the class. Can be used for introspection.

New in version 1.8.0.

### **get\_property\_observers()**

Returns a list of methods that are bound to the property/event passed as the *name* argument:

```
widget_instance.get_property_observers('on_release')
```

New in version 1.8.0.

### **getter()**

Return the getter of a property.

New in version 1.0.9.

### **is\_event\_type()**

Return True if the event\_type is already registered.

New in version 1.0.4.

### **properties()**

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()**

Get a property instance from the name.

New in version 1.0.9.

Returns A **Property** derived instance corresponding to the name.

### **register\_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()**

Return the setter of a property. Use: instance.setter('name'). The setter is a convenient call-back 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()**

Unbind properties from callback functions.

Same usage as `bind()`.

### **unregister\_event\_types()**

Unregister an event type in the dispatcher.

### **class kivy.event.ObjectWithUid**

Bases: `object`

(internal) This class assists in providing unique identifiers for class instances. It is not intended for direct usage.

## 21.11 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 at 0x905c88c>  
Factory instance to use for getting new classes

## 21.12 Garden

New in version 1.7.0.

Changed in version 1.8.0.

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.

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.

---

### 21.12.1 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

## 21.13 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](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)

## 21.14 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)

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](mailto: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: 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**

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.

## 21.15 Interactive launcher

New in version 1.3.0.

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.

---

### 21.15.1 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.

---

### 21.15.2 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

# 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.

---

### 21.15.3 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.

---

### 21.15.4 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: 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.

## 21.16 Kivy Base

This module contains core Kivy functionality and is not intended for end users. Feel free to look though it, but calling any of these methods directly may well result in unpredictable behavior.

### 21.16.1 Event loop management

```
kivy.base.EventLoop = <kivy.base.EventLoopBase object at 0x9d2277c>
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).

**close**()

Exit from the main loop and stop all configured input providers.

**dispatch\_input**()

Called by `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 `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\_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 `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: 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.STOP`.

**class kivy.base.ExceptionManagerBase**

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 instance at 0x9f5c78c>`

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:

**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 usefull for embedding Kivy in another toolkit. (like Qt, check kivy-designed)

`kivy.base.stopTouchApp()`

Stop the current application by leaving the main loop

## 21.17 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`.

### 21.17.1 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.

### 21.17.2 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 instance.

*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).
- 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.

### 21.17.3 Value Expressions, on\_property Expressions, 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):

```
<Widget>:
    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")
```

Furthermore, if a class definition contains an `id`, you can use it as a keyword:

```
<Widget>:
    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).

## 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 multi-line statements 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 an invalid statement:

```
on_state:
    if self.state == 'normal':
        print('normal')
```

#### 21.17.4 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!'.

#### 21.17.5 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.

### 21.17.6 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.

---

### 21.17.7 Templates

Changed in version 1.7.0: Template usage is now deprecated. Please use Dynamic classes instead.

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.
```

### 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. Theses 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 :)

Template limitations

When you are creating a context:

1. you cannot use references other than "root":

```

<MyRule>:
    Widget:
        id: mywidget
        value: 'bleh'
    Template:
        ctxkey: mywidget.value # << fail, this reference mywidget id

```

1. 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, the context will not update the
        # context

```

## 21.17.8 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.

### 21.17.9 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>
```

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)
```

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
```

include <file>

New in version 1.8.1.

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: object

A lightweight class allowing to get an object be bound to action in kv, without using as much resources as EventDispatcher

New in version 1.8.1.

**bind**(\*\*kwargs)

This method is to be overridden by your subclass

kwargs will contains callables to call when your observables are updated, so you can trigger a reevaluation of the expression when you need it, just calling all the callbacks that are relevant.

`kivy.lang.Builder` = <kivy.lang.BuilderBase object at 0x99236ac>  
Main instance of a `BuilderBase`.

**class** `kivy.lang.BuilderBase`  
Bases: `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*)  
Search all the rules that match the widget and apply them.

**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.

**match**(*widget*)  
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. .. versionadded:: 1.0.5  
With templates, you can construct custom widgets from a kv lang definition by giving them a context. Check [Template usage](#).

**unbind\_widget**(*uid*)  
(internal) Unbind all the handlers created by the rules of the widget. The `kivy.uix.widget.Widget.uid` is passed here instead of the widget itself, because we are using it in the widget destructor.  
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.

**class** `kivy.lang.BuilderException`(*context*, *line*, *message*, *cause=None*)  
Bases: `kivy.lang.ParserException`  
Exception raised when the Builder failed to apply a rule on a widget.

```
class kivy.lang.Parser(**kwargs)
```

Bases: 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 #.

```
class kivy.lang.ParserException(context, line, message, cause=None)
```

Bases: exceptions.Exception

Exception raised when something wrong happened in a kv file.

## 21.18 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
```

### 21.18.1 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
```

More information about the allowed values are described in the [kivy.config](#) module.

## 21.18.2 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 = <logging.Logger object at 0x90992cc>
    Kivy default logger instance

class kivy.logger.LoggerHistory(level=0)
    Bases: logging.Handler
    Kivy history handler
```

## 21.19 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.

### 21.19.1 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.

### 21.19.2 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)
```

### 21.19.3 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_FONTSCALE`: 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_FONTSCALE=1.2 python main.py
```

`kivy.metrics.Metrics = <kivy.metrics.MetricsBase object at 0x9879eac>`  
 Default instance of `MetricsBase`, used everywhere in the code .. versionadded:: 1.7.0

**class kivy.metrics.MetricsBase**  
 Bases: `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

`kivy.metrics.metrics = <kivy.metrics.MetricsBase object at 0x9879eac>`  
default instance of `MetricsBase`, used everywhere in the code (deprecated, use `Metrics` instead.)

## 21.20 Multistroke gesture recognizer

New in version 1.8.1.

**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.

### 21.20.1 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()`.

### 21.20.2 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
    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

```

### 21.20.3 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/ndc-protractor.pdf>

Some of the code is derived from the JavaScript implementation here: <http://depts.washington.edu/aimgroup/proj/dollar/ndc-protractor.pdf>

```
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 captial 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])

# 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: 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: “\$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: 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 Multi-strokeGesture 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: 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`)

## 21.21 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)`
- `aaa`
- `rrggbb`

For hexadecimal values, you can also use:

- `#aaa`
- `#rrggbb`

`kivy.parser.parse_int`

alias of `int`

`kivy.parser.parse_float`

alias of `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).

## 21.22 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.

## 21.22.1 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)
```

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. An `errorvalue` is a substitute for the invalid value. An `errorhandler` is a callable (single argument function or lambda) which can return a valid substitute.

errorvalue parameter:

```
# simply returns 0 if the value exceeds the bounds
bnp = BoundedNumericProperty(0, min=-500, max=500, errorvalue=0)
```

errorhandler parameter:

```
# returns the boundary value when exceeded
bnp = BoundedNumericProperty(0, min=-500, max=500,
    errorhandler=lambda x: 500 if x > 500 else -500)
```

## Conclusion

Kivy properties are easier to use than the standard ones. See the next chapter for examples of how to use them :)

### 21.22.2 Observe Properties 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
```

Observe using 'on\_<propname>'

If you created 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.

### 21.22.3 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 1.8.1, 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

Bases: object

Base class for building more complex properties.

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*).

Changed in version 1.4.2: Parameters *errorhandler* and *errorvalue* added

### bind()

Add a new observer to be called only when the value is changed.

### dispatch()

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)

```

### get()

Return the value of the property.

### link()

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.

### set()

Set a new value for the property.

### unbind()

Remove the observer from our widget observer list.

**class** kivy.properties.NumericProperty

Bases: [kivy.properties.Property](#)

Property that represents a numeric value.

**Parameters**

*default*: 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()**

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

Bases: [kivy.properties.Property](#)

Property that represents a string value.

**Parameters**

*default*: string, defaults to "" Specifies the default value of the property.

**class** kivy.properties.ListProperty

Bases: [kivy.properties.Property](#)

Property that represents a list.

**Parameters**

*default*: 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 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 = widget.my_list = [1, 5, 7]
>>> print my_list is widget.my_list
False
>>> my_list.append(10)
>>> print(my_list, widget.my_list)
[1, 5, 7, 10], [1, 5, 7]
```

**class** kivy.properties.ObjectProperty

Bases: [kivy.properties.Property](#)

Property that represents a Python object.

**Parameters**

*default*: 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 1.8.1: `rebind` has been introduced.

Changed in version 1.7.0: `baseclass` parameter added.

**class kivy.properties.BooleanProperty**

Bases: `kivy.properties.Property`

Property that represents only a boolean value.

**Parameters**

**default: boolean** Specifies the default value of the property.

**class kivy.properties.BoundedNumericProperty**

Bases: `kivy.properties.Property`

Property that represents a numeric value within a minimum bound and/or 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()**

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()**

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()

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()

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)

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

Bases: `kivy.properties.Property`

Property that represents a string from a predefined list of valid 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.

### options

Return the options available.

New in version 1.0.9.

## class kivy.properties.ReferenceListProperty

Bases: `kivy.properties.Property`

Property that allows the creation of a tuple of other properties.

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*.

## class kivy.properties.**AliasProperty**

Bases: [kivy.properties.Property](#)

Create a property with a custom getter and setter.

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:

```
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'))
```

### Parameters

**getter: function** Function to use as a property getter

**setter: function** Function to use as a property setter. Properties listening to the alias property won't be updated 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

**cache: boolean** If True, the value will be cached, until one of the binded elements will changes

**rebind: bool, defaults to False** See [ObjectProperty](#) for details.

Changed in version 1.8.1: *rebind* has been introduced.

Changed in version 1.4.0: Parameter *cache* added.

## class kivy.properties.**DictProperty**

Bases: [kivy.properties.Property](#)

Property that represents a dict.

### Parameters

**default: dict, defaults to None** Specifies the default value of the property.

**rebind: bool, defaults to False** See [ObjectProperty](#) for details.

Changed in version 1.8.1: *rebind* has been introduced.

**Warning:** Similar to [ListProperty](#), when assigning a dict to a [DictProperty](#), the dict stored in the property is a copy of the dict and not the original dict. See [ListProperty](#) for details.

## class kivy.properties.**VariableListProperty**

Bases: [kivy.properties.Property](#)

A ListProperty that allows you to work with a variable amount of 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.

### class `kivy.properties.ConfigParserProperty`

Bases: `kivy.properties.Property`

Property that allows one to bind to changes in the configuration values 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 1.8.1.

## set\_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())
```

## 21.23 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.

`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.

## 21.24 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 1.8.1.

`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.

## 21.25 Utils

Changed in version 1.6.0: The `OrderedDict` class has been removed. Use the `collections.OrderedDict`.

`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)`

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: `list`

List with a `clear()` method.

**Warning:** Usage of the `iterate()` function will decrease your performance.

`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: `dict`

`QueryDict` is a `dict()` that can be queried with `dot`.

New in version 1.0.4.

```
d = QueryDict()
# create a key named toto, with the value 1
d.toto = 1
# it's the same as
d['toto'] = 1
```

`kivy.utils.platform` = platform name: 'linux' from: <kivy.utils.Platform object at 0x90ae7ac>

New in version 1.3.0.

Deprecated since 1.8.0: Use `platform` as variable instead of a function.

Calling `platform()` will return one of: *win, linux, android, macosx, ios* or *unknown*.

Changed in version 1.8.0.

`platform` also behaves like a regular variable in comparisons like so:

```
from kivy import platform
if platform == 'linux':
    do_linux_things()
if platform() == 'linux': # triggers deprecation warning
    do_more_linux_things()
foo = {'linux' : do_linux_things}
foo[platform]() # calls do_linux_things
p = platform # assigns to a module object
if p is 'android':
    do_android_things()
p += 'some string' # error!
```

`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: `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](#).

## 21.26 Vector

The **Vector** represents a 2D vector (x, y). Our implementation is made on top of a Python list.

Example for 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
>>> v[1]
45
>>> v.y
45
```

### 21.26.1 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))
```

### 21.26.2 Vector operators

The **Vector** supports some numeric operators like +, -, /:

```
>>> 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 do in-place operations:

```
>>> 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: 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](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)
>>> v
[70.710678118654755, 70.710678118654741]
```

### **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
```

## 21.27 Weak Method

The **WeakMethod** is used in the Clock class to allow a reference to a bound method that permits the associated object to be garbage collected. Check [examples/core/clock\\_method.py](http://code.activestate.com/recipes/81253/) for more information.

This WeakMethod class is taken from the recipe <http://code.activestate.com/recipes/81253/>, based on the nicodemus version. (thanks to him !)

```
class kivy.weakmethod.WeakMethod(method)
```

Bases: `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.



# ADAPTERS

New in version 1.5.0.

An adapter is an intermediating controller-type class that builds views for top-level widgets, interacting with data as prescribed by parameters. Kivy adapters are modelled on the [Adapter design pattern](#). On the view side is an [AbstractView](#), which is the base view for a [ListView](#).

- **Adapters:** The base `Adapter` is subclassed by `SimpleListAdapter` and by `ListAdapter`. Further, `DictAdapter` is a subclass of `ListAdapter`.

*[Adapter](#), [SimpleListAdapter](#), [ListAdapter](#), [DictAdapter](#).*

- **Models:** The data for which an adapter serves as a bridge to views can be any sort of data. However, for convenience, model mixin classes can ease the preparation or shaping of data for use in the system. For selection operations, the `SelectableDataItem` can optionally prepare data items to provide and receive selection information (data items are not required to be “selection-aware”, but in some cases it may be desired).

*[SelectableDataItem](#).*

- **Args Converters:** Argument converters are made by the application programmer to do the work of converting data items to argument dictionaries suitable for instantiating views.

*[List Item View Argument Converters](#).*

---

## 22.1 Adapter

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

An `Adapter` is a bridge between data and an `AbstractView` or one of its subclasses, such as a `ListView`.

Arguments:

- *data*: for any sort of data to be used in a view. For an `Adapter`, data can be an object as well as a list, dict, etc. For a `ListAdapter`, data should be a list. For a `DictAdapter`, data should be a dict.
- *cls*: for a list key class used to instantiate list item view instances (Use this or the template argument).
- *template*: a kv template to use to instantiate list item view instances (Use this or the cls argument).

- *args\_converter*: a function to transform the data argument sets, in preparation for either a cls instantiation or a kv template invocation. If no *args\_converter* is provided, a default one, that assumes that the data items are strings, is used.

**class** `kivy.adapters.adapter.Adapter` (*\*\*kwargs*)

Bases: `kivy.event.EventDispatcher`

An **Adapter** is a bridge between data and an **AbstractView** or one of its subclasses, such as a **ListView**.

#### **args\_converter**

A function that prepares an args dict for the cls or kv template to build a view from a data item.

If an *args\_converter* is not provided, a default one is set that assumes simple content in the form of a list of strings.

*args\_converter* is an **ObjectProperty** and defaults to None.

#### **cls**

A class for instantiating a given view item (Use this or template).

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

#### **data**

The data for which a view is to be constructed using either the *cls* or *template* provided, together with the *args\_converter* provided or the default *args\_converter*.

In this base class, *data* is an **ObjectProperty**, so it could be used for a wide variety of single-view needs.

Subclasses may override it in order to use another data type, such as a **ListProperty** or **DictProperty** as appropriate. For example, in a **ListAdapter**, *data* is a **ListProperty**.

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

#### **template**

A kv template for instantiating a given view item (Use this or *cls*).

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

## 22.2 DictAdapter

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

A **DictAdapter** is an adapter around a python dictionary of records. It extends the list-like capabilities of the **ListAdapter**.

If you wish to have a bare-bones list adapter, without selection, use the **SimpleListAdapter**.

**class** `kivy.adapters.dictadapter.DictAdapter` (*\*\*kwargs*)

Bases: `kivy.adapters.listadapter.ListAdapter`

A **DictAdapter** is an adapter around a python dictionary of records. It extends the list-like capabilities of the **ListAdapter**.

#### **cut\_to\_sel** (*\*args*)

Same as *trim\_to\_sel*, but intervening list items within the selected range are also cut, leaving only list items that are selected.

*sorted\_keys* will be updated by *update\_for\_new\_data()*.

## **data**

A dict that indexes records by keys that are equivalent to the keys in `sorted_keys`, or they are a superset of the keys in `sorted_keys`.

The values can be strings, class instances, dicts, etc.

`data` is a `DictProperty` and defaults to `None`.

## **sorted\_keys**

The `sorted_keys` list property contains a list of hashable objects (can be strings) that will be used directly if no `args_converter` function is provided. If there is an `args_converter`, the record received from a lookup of the data, using keys from `sorted_keys`, will be passed to it for instantiation of list item view class instances.

`sorted_keys` is a `ListProperty` and defaults to `[]`.

## **trim\_left\_of\_sel(\*args)**

Cut list items with indices in `sorted_keys` that are less than the index of the first selected item, if there is a selection.

`sorted_keys` will be updated by `update_for_new_data()`.

## **trim\_right\_of\_sel(\*args)**

Cut list items with indices in `sorted_keys` that are greater than the index of the last selected item, if there is a selection.

`sorted_keys` will be updated by `update_for_new_data()`.

## **trim\_to\_sel(\*args)**

Cut list items with indices in `sorted_keys` that are less than or greater than the index of the last selected item, if there is a selection. This preserves intervening list items within the selected range.

`sorted_keys` will be updated by `update_for_new_data()`.

## 22.3 List Item View Argument Converters

New in version 1.5.

The default list item args converter for list adapters is a function (shown below) that takes a row index and a string. It returns a dict with the string as the `text` item, along with two properties suited for simple text items with a height of 25.

Argument converters may be normal functions or, as in the case of the default args converter, lambdas:

```
list_item_args_converter = lambda row_index, x: {'text': x,
                                                'size_hint_y': None,
                                                'height': 25}
```

## 22.4 ListAdapter

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

A `ListAdapter` is an adapter around a python list.

Selection operations are a main concern for the class.

From an Adapter, a `ListAdapter` gets `cls`, `template`, and `args_converter` properties and adds others that control selection behaviour:

- `selection`, a list of selected items.
- `selection_mode`, 'single', 'multiple', 'none'
- `allow_empty_selection`, a boolean – If False, a selection is forced. If True, and only user or programmatic action will change selection, it can be empty.

If you wish to have a bare-bones list adapter, without selection, use a `SimpleListAdapter`.

A `DictAdapter` is a subclass of a `ListAdapter`. They both dispatch the `on_selection_change` event.

## Events

**`on_selection_change: (view, view list)`** Fired when selection changes

Changed in version 1.6.0: Added `data = ListProperty([])`, which was probably inadvertently deleted at some point. This means that whenever data changes an update will fire, instead of having to reset the data object (Adapter has `data` defined as an `ObjectProperty`, so we need to reset it here to `ListProperty`). See also `DictAdapter` and its set of `data = DictProperty()`.

`class kivy.adapters.listadapter.ListAdapter(**kwargs)`

Bases: `kivy.adapters.adapter.Adapter`, `kivy.event.EventDispatcher`

A base class for adapters interfacing with lists, dictionaries or other collection type data, adding selection, view creation and management functionality.

### **allow\_empty\_selection**

The `allow_empty_selection` may be used for cascading selection between several list views, or between a list view and an observing view. Such automatic maintenance of the selection is important for all but simple list displays. Set `allow_empty_selection` to False and the selection is auto-initialized and always maintained, so any observing views may likewise be updated to stay in sync.

`allow_empty_selection` is a `BooleanProperty` and defaults to True.

### **cached\_views**

View instances for data items are instantiated and managed by the adapter. Here we maintain a dictionary containing the view instances keyed to the indices in the data.

This dictionary works as a cache. `get_view()` only asks for a view from the adapter if one is not already stored for the requested index.

`cached_views` is a `DictProperty` and defaults to {}.

### **create\_view(index)**

This method is more complicated than the one in `kivy.adapters.adapter.Adapter` and `kivy.adapters.simplelistadapter.SimpleListAdapter`, because here we create bindings for the data item and its children back to `self.handle_selection()`, and do other selection-related tasks to keep item views in sync with the data.

### **cut\_to\_sel(\*args)**

Same as `trim_to_sel`, but intervening list items within the selected range are also cut, leaving only list items that are selected.

### **data**

The data list property is redefined here, overriding its definition as an `ObjectProperty` in the Adapter class. We bind to `data` so that any changes will trigger updates. See also how the `DictAdapter` redefines `data` as a `DictProperty`.

`data` is a `ListProperty` and defaults to [].

### **on\_selection\_change(\*args)**

`on_selection_change()` is the default handler for the `on_selection_change` event.

## propagate\_selection\_to\_data

Normally, data items are not selected/deselected because the data items might not have an `is_selected` boolean property – only the item view for a given data item is selected/deselected as part of the maintained selection list. However, if the data items do have an `is_selected` property, or if they mix in `SelectableDataItem`, the selection machinery can propagate selection to data items. This can be useful for storing selection state in a local database or backend database for maintaining state in game play or other similar scenarios. It is a convenience function.

To propagate selection or not?

Consider a shopping list application for shopping for fruits at the market. The app allows for the selection of fruits to buy for each day of the week, presenting seven lists: one for each day of the week. Each list is loaded with all the available fruits, but the selection for each is a subset. There is only one set of fruit data shared between the lists, so it would not make sense to propagate selection to the data because selection in any of the seven lists would clash and mix with that of the others.

However, consider a game that uses the same fruits data for selecting fruits available for fruit-tossing. A given round of play could have a full fruits list, with fruits available for tossing shown selected. If the game is saved and rerun, the full fruits list, with selection marked on each item, would be reloaded correctly if selection is always propagated to the data. You could accomplish the same functionality by writing code to operate on list selection, but having selection stored in the data `ListProperty` might prove convenient in some cases.

`propagate_selection_to_data` is a `BooleanProperty` and defaults to `False`.

## select\_list(*view\_list*, *extend=True*)

The select call is made for the items in the provided `view_list`.

Arguments:

`view_list`: the list of item views to become the new selection, or to add to the existing selection

`extend`: boolean for whether or not to extend the existing list

## selection

The selection list property is the container for selected items.

`selection` is a `ListProperty` and defaults to `[]`.

## selection\_limit

When the `selection_mode` is `multiple` and the `selection_limit` is non-negative, this number will limit the number of selected items. It can be set to 1, which is equivalent to single selection. If `selection_limit` is not set, the default value is -1, meaning that no limit will be enforced.

`selection_limit` is a `NumericProperty` and defaults to -1 (no limit).

## selection\_mode

Selection modes:

- *none*, use the list as a simple list (no select action). This option is here so that selection can be turned off, momentarily or permanently, for an existing list adapter. A `ListAdapter` is not meant to be used as a primary no-selection list adapter. Use a `SimpleListAdapter` for that.
- *single*, multi-touch/click ignored. Single item selection only.
- *multiple*, multi-touch / incremental addition to selection allowed; may be limited to a count by `selection_limit`

`selection_mode` is an `OptionProperty` and defaults to 'single'.

**trim\_left\_of\_sel(\*args)**

Cut list items with indices in sorted\_keys that are less than the index of the first selected item if there is a selection.

**trim\_right\_of\_sel(\*args)**

Cut list items with indices in sorted\_keys that are greater than the index of the last selected item if there is a selection.

**trim\_to\_sel(\*args)**

Cut list items with indices in sorted\_keys that are less than or greater than the index of the last selected item if there is a selection. This preserves intervening list items within the selected range.

## 22.5 SelectableDataItem

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

### 22.5.1 Data Models

Kivy is open about the type of data used in applications built with the system. However, base classes are sometimes needed to ensure data conforms to the requirements of some parts of the system.

A **SelectableDataItem** is a basic Python data model class that can be used as a mixin to build data objects that are compatible with Kivy's **Adapter** and selection system, which works with views such as a **Listview**. The boolean property `is_selected` is a requirement.

The default operation of the selection system is to not propagate selection in views such as **Listview** to the underlying data – selection is by default a view-only operation. However, in some cases, it is useful to propagate selection to the actual data items.

You may, of course, build your own Python data model system as the backend for a Kivy application. For instance, to use the Google App Engine datamodeling system with Kivy, this class could be redefined as:

```
from google.appengine.ext import db

class MySelectableDataItem(db.Model):
    ... other properties
    is_selected = db.BooleanProperty()
```

It is easy to build such a class with plain Python.

```
class kivy.adapters.models.SelectableDataItem(**kwargs)
    Bases: object
```

A mixin class containing requirements for selection operations.

**is\_selected**

A boolean property indicating whether the data item is selected or not.

## 22.6 SimpleListAdapter

New in version 1.5.



**Warning:** This code is still experimental, and its API is subject to change in a future version.

The `SimpleListAdapter` is used for basic lists. For example, it can be used for displaying a list of read-only strings that do not require user interaction.

```
class kivy.adapters.simplelistadapter.SimpleListAdapter(**kwargs)
```

Bases: `kivy.adapters.adapter.Adapter`

A `SimpleListAdapter` is an adapter around a Python list.

From `Adapter`, the `ListAdapter` gets `cls`, `template`, and `args_converter` properties.

#### **data**

The `data` list property contains a list of objects (which can be strings) that will be used directly if no `args_converter` function is provided. If there is an `args_converter`, the data objects will be passed to it for instantiating the item view class instances.

`data` is a `ListProperty` and defaults to [].



---

# ADAPTER

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

An **Adapter** is a bridge between data and an **AbstractView** or one of its subclasses, such as a **ListView**.

Arguments:

- *data*: for any sort of data to be used in a view. For an **Adapter**, data can be an object as well as a list, dict, etc. For a **ListAdapter**, data should be a list. For a **DictAdapter**, data should be a dict.
- *cls*: for a list key class used to instantiate list item view instances (Use this or the template argument).
- *template*: a kv template to use to instantiate list item view instances (Use this or the cls argument).
- *args\_converter*: a function to transform the data argument sets, in preparation for either a cls instantiation or a kv template invocation. If no *args\_converter* is provided, a default one, that assumes that the data items are strings, is used.

**class** `kivy.adapters.adapter.Adapter` (*\*\*kwargs*)

Bases: `kivy.event.EventDispatcher`

An **Adapter** is a bridge between data and an **AbstractView** or one of its subclasses, such as a **ListView**.

### **args\_converter**

A function that prepares an args dict for the cls or kv template to build a view from a data item.

If an *args\_converter* is not provided, a default one is set that assumes simple content in the form of a list of strings.

*args\_converter* is an **ObjectProperty** and defaults to None.

### **cls**

A class for instantiating a given view item (Use this or template).

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

### **data**

The data for which a view is to be constructed using either the cls or template provided, together with the *args\_converter* provided or the default *args\_converter*.

In this base class, data is an **ObjectProperty**, so it could be used for a wide variety of single-view needs.

Subclasses may override it in order to use another data type, such as a `ListProperty` or `DictProperty` as appropriate. For example, in a `ListAdapter`, data is a `ListProperty`. `data` is an `ObjectProperty` and defaults to `None`.

**template**

A kv template for instantiating a given view item (Use this or `cls`).

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

# LIST ITEM VIEW ARGUMENT CONVERTERS

New in version 1.5.

The default list item args converter for list adapters is a function (shown below) that takes a row index and a string. It returns a dict with the string as the *text* item, along with two properties suited for simple text items with a height of 25.

Argument converters may be normal functions or, as in the case of the default args converter, lambdas:

```
list_item_args_converter = lambda row_index, x: {'text': x,  
                                                'size_hint_y': None,  
                                                'height': 25}
```



# DICTADAPTER

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

A `DictAdapter` is an adapter around a python dictionary of records. It extends the list-like capabilities of the `ListAdapter`.

If you wish to have a bare-bones list adapter, without selection, use the `SimpleListAdapter`.

```
class kivy.adapters.dictadapter.DictAdapter(**kwargs)
    Bases: kivy.adapters.listadapter.ListAdapter
```

A `DictAdapter` is an adapter around a python dictionary of records. It extends the list-like capabilities of the `ListAdapter`.

**cut\_to\_sel**(\*args)

Same as `trim_to_sel`, but intervening list items within the selected range are also cut, leaving only list items that are selected.

`sorted_keys` will be updated by `update_for_new_data()`.

**data**

A dict that indexes records by keys that are equivalent to the keys in `sorted_keys`, or they are a superset of the keys in `sorted_keys`.

The values can be strings, class instances, dicts, etc.

`data` is a `DictProperty` and defaults to `None`.

**sorted\_keys**

The `sorted_keys` list property contains a list of hashable objects (can be strings) that will be used directly if no `args_converter` function is provided. If there is an `args_converter`, the record received from a lookup of the data, using keys from `sorted_keys`, will be passed to it for instantiation of list item view class instances.

`sorted_keys` is a `ListProperty` and defaults to `[]`.

**trim\_left\_of\_sel**(\*args)

Cut list items with indices in `sorted_keys` that are less than the index of the first selected item, if there is a selection.

`sorted_keys` will be updated by `update_for_new_data()`.

**trim\_right\_of\_sel**(\*args)

Cut list items with indices in `sorted_keys` that are greater than the index of the last selected item, if there is a selection.

`sorted_keys` will be updated by `update_for_new_data()`.

**trim\_to\_sel**(\*args)

Cut list items with indices in `sorted_keys` that are less than or greater than the index of the last selected item, if there is a selection. This preserves intervening list items within the selected range.

`sorted_keys` will be updated by `update_for_new_data()`.



# LISTADAPTER

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

A `ListAdapter` is an adapter around a python list.

Selection operations are a main concern for the class.

From an `Adapter`, a `ListAdapter` gets `cls`, `template`, and `args_converter` properties and adds others that control selection behaviour:

- `selection`, a list of selected items.
- `selection_mode`, 'single', 'multiple', 'none'
- `allow_empty_selection`, a boolean – If False, a selection is forced. If True, and only user or programmatic action will change selection, it can be empty.

If you wish to have a bare-bones list adapter, without selection, use a `SimpleListAdapter`.

A `DictAdapter` is a subclass of a `ListAdapter`. They both dispatch the `on_selection_change` event.

## Events

**`on_selection_change: (view, view list)`** Fired when selection changes

Changed in version 1.6.0: Added `data = ListProperty([])`, which was probably inadvertently deleted at some point. This means that whenever data changes an update will fire, instead of having to reset the data object (Adapter has `data` defined as an `ObjectProperty`, so we need to reset it here to `ListProperty`). See also `DictAdapter` and its set of `data = DictProperty()`.

```
class kivy.adapters.listadapter.ListAdapter(**kwargs)
```

Bases: `kivy.adapters.adapter.Adapter`, `kivy.event.EventDispatcher`

A base class for adapters interfacing with lists, dictionaries or other collection type data, adding selection, view creation and management functionality.

## **allow\_empty\_selection**

The `allow_empty_selection` may be used for cascading selection between several list views, or between a list view and an observing view. Such automatic maintenance of the selection is important for all but simple list displays. Set `allow_empty_selection` to False and the selection is auto-initialized and always maintained, so any observing views may likewise be updated to stay in sync.

`allow_empty_selection` is a `BooleanProperty` and defaults to True.

## **cached\_views**

View instances for data items are instantiated and managed by the adapter. Here we maintain a dictionary containing the view instances keyed to the indices in the data.

This dictionary works as a cache. `get_view()` only asks for a view from the adapter if one is not already stored for the requested index.

`cached_views` is a `DictProperty` and defaults to {}.

#### **create\_view(*index*)**

This method is more complicated than the one in `kivy.adapters.adapter.Adapter` and `kivy.adapters.simplelistadapter.SimpleListAdapter`, because here we create bindings for the data item and its children back to `self.handle_selection()`, and do other selection-related tasks to keep item views in sync with the data.

#### **cut\_to\_sel(\*args)**

Same as `trim_to_sel`, but intervening list items within the selected range are also cut, leaving only list items that are selected.

#### **data**

The data list property is redefined here, overriding its definition as an `ObjectProperty` in the `Adapter` class. We bind to data so that any changes will trigger updates. See also how the `DictAdapter` redefines data as a `DictProperty`.

`data` is a `ListProperty` and defaults to [].

#### **on\_selection\_change(\*args)**

`on_selection_change()` is the default handler for the `on_selection_change` event.

#### **propagate\_selection\_to\_data**

Normally, data items are not selected/deselected because the data items might not have an `is_selected` boolean property – only the item view for a given data item is selected/deselected as part of the maintained selection list. However, if the data items do have an `is_selected` property, or if they mix in `SelectableDataItem`, the selection machinery can propagate selection to data items. This can be useful for storing selection state in a local database or backend database for maintaining state in game play or other similar scenarios. It is a convenience function.

To propagate selection or not?

Consider a shopping list application for shopping for fruits at the market. The app allows for the selection of fruits to buy for each day of the week, presenting seven lists: one for each day of the week. Each list is loaded with all the available fruits, but the selection for each is a subset. There is only one set of fruit data shared between the lists, so it would not make sense to propagate selection to the data because selection in any of the seven lists would clash and mix with that of the others.

However, consider a game that uses the same fruits data for selecting fruits available for fruit-tossing. A given round of play could have a full fruits list, with fruits available for tossing shown selected. If the game is saved and rerun, the full fruits list, with selection marked on each item, would be reloaded correctly if selection is always propagated to the data. You could accomplish the same functionality by writing code to operate on list selection, but having selection stored in the data `ListProperty` might prove convenient in some cases.

`propagate_selection_to_data` is a `BooleanProperty` and defaults to `False`.

#### **select\_list(*view\_list*, *extend=True*)**

The `select` call is made for the items in the provided `view_list`.

Arguments:

`view_list`: the list of item views to become the new selection, or to add to the existing selection

`extend`: boolean for whether or not to extend the existing list

## selection

The selection list property is the container for selected items.

`selection` is a `ListProperty` and defaults to [].

## selection\_limit

When the `selection_mode` is multiple and the `selection_limit` is non-negative, this number will limit the number of selected items. It can be set to 1, which is equivalent to single selection. If `selection_limit` is not set, the default value is -1, meaning that no limit will be enforced.

`selection_limit` is a `NumericProperty` and defaults to -1 (no limit).

## selection\_mode

Selection modes:

- *none*, use the list as a simple list (no select action). This option is here so that selection can be turned off, momentarily or permanently, for an existing list adapter. A `ListAdapter` is not meant to be used as a primary no-selection list adapter. Use a `SimpleListAdapter` for that.
- *single*, multi-touch/click ignored. Single item selection only.
- *multiple*, multi-touch / incremental addition to selection allowed; may be limited to a count by `selection_limit`

`selection_mode` is an `OptionProperty` and defaults to 'single'.

## trim\_left\_of\_sel(\*args)

Cut list items with indices in `sorted_keys` that are less than the index of the first selected item if there is a selection.

## trim\_right\_of\_sel(\*args)

Cut list items with indices in `sorted_keys` that are greater than the index of the last selected item if there is a selection.

## trim\_to\_sel(\*args)

Cut list items with indices in `sorted_keys` that are less than or greater than the index of the last selected item if there is a selection. This preserves intervening list items within the selected range.



# SELECTABLEDATAITEM

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

## 27.1 Data Models

Kivy is open about the type of data used in applications built with the system. However, base classes are sometimes needed to ensure data conforms to the requirements of some parts of the system.

A `SelectableDataItem` is a basic Python data model class that can be used as a mixin to build data objects that are compatible with Kivy's `Adapter` and selection system, which works with views such as a `ListView`. The boolean property `is_selected` is a requirement.

The default operation of the selection system is to not propagate selection in views such as `ListView` to the underlying data – selection is by default a view-only operation. However, in some cases, it is useful to propagate selection to the actual data items.

You may, of course, build your own Python data model system as the backend for a Kivy application. For instance, to use the Google App Engine datamodeling system with Kivy, this class could be redefined as:

```
from google.appengine.ext import db

class MySelectableDataItem(db.Model):
    ... other properties
    is_selected = db.BooleanProperty()
```

It is easy to build such a class with plain Python.

```
class kivy.adapters.models.SelectableDataItem(**kwargs)
    Bases: object
```

A mixin class containing requirements for selection operations.

**is\_selected**

A boolean property indicating whether the data item is selected or not.



# SIMPLELISTADAPTER

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

The `SimpleListAdapter` is used for basic lists. For example, it can be used for displaying a list of read-only strings that do not require user interaction.

```
class kivy.adapters.simplelistadapter.SimpleListAdapter(**kwargs)
    Bases: kivy.adapters.adapter.Adapter
```

A `SimpleListAdapter` is an adapter around a Python list.

From `Adapter`, the `ListAdapter` gets `cls`, `template`, and `args_converter` properties.

## **data**

The `data` list property contains a list of objects (which can be strings) that will be used directly if no `args_converter` function is provided. If there is an `args_converter`, the data objects will be passed to it for instantiating the item view class instances.

`data` is a `ListProperty` and defaults to [].





# 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`

## 29.1 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).

## 29.2 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.

## 29.3 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)
```

## 29.4 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)
```

## 29.5 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(**kwargs)
    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 1 / 60.

### Events

**on\_start:** widget Fired when the animation is started on a widget.

**on\_complete:** widget Fired when the animation is completed or stopped on a widget.

**on\_progress:** widget, progression Fired when the progression of the animation is changing.

Changed in version 1.4.0: Added s/step parameter.

### 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, \*larges)

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.

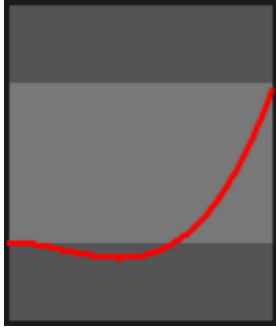
## **class kivy.animation.AnimationTransition**

Bases: object

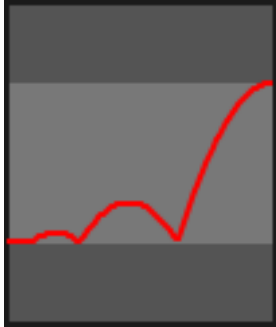
Collection of animation functions to be used with the Animation object. Easing Functions ported to Kivy from the Clutter Project <http://www.clutter-project.org/docs/clutter/stable/ClutterAlpha.html>

The *progress* parameter in each animation function is in the range 0-1.

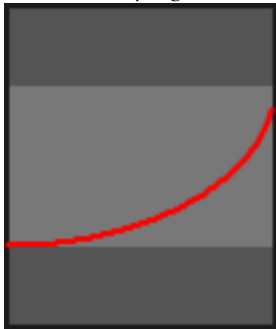
### **static in\_back**(*progress*)



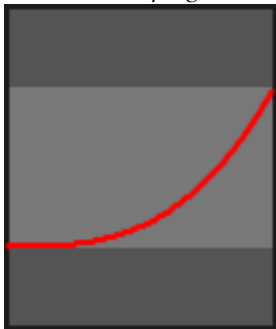
`static in_bounce(progress)`



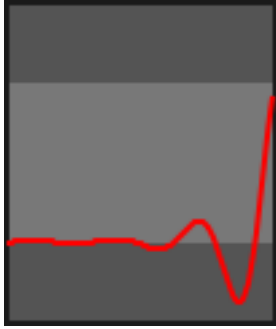
`static in_circ(progress)`



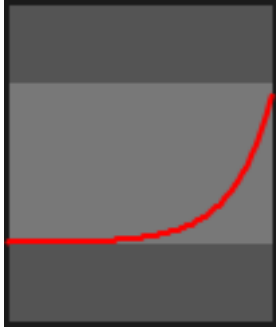
`static in_cubic(progress)`



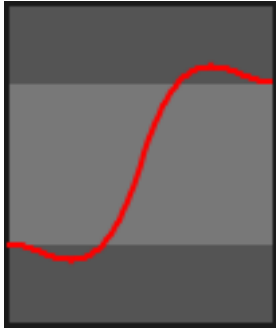
`static in_elastic(progress)`



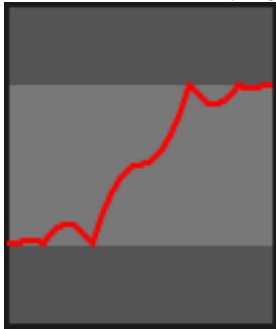
`static in_expo(progress)`



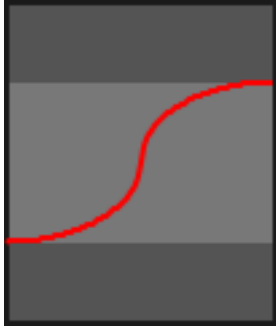
`static in_out_back(progress)`



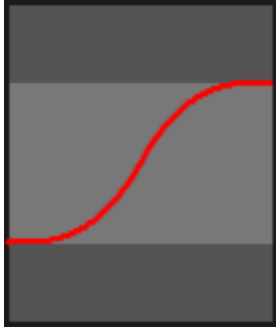
`static in_out_bounce(progress)`



`static in_out_circ(progress)`



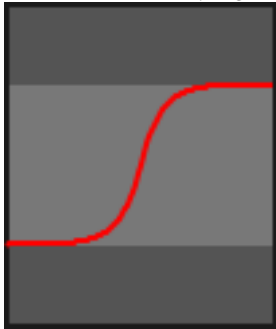
static **in\_out\_cubic**(*progress*)



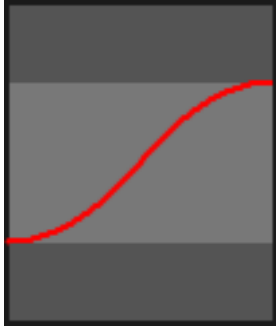
static **in\_out\_elastic**(*progress*)



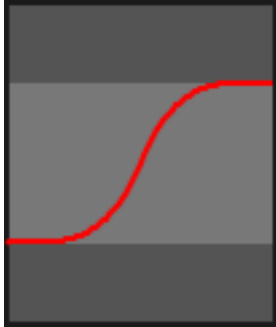
static **in\_out\_expo**(*progress*)



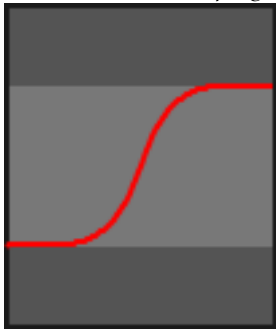
static **in\_out\_quad**(*progress*)



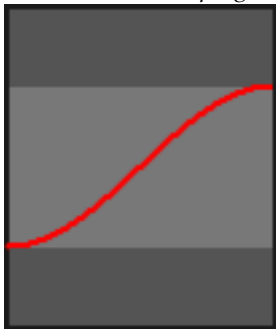
static **in\_out\_quart**(*progress*)



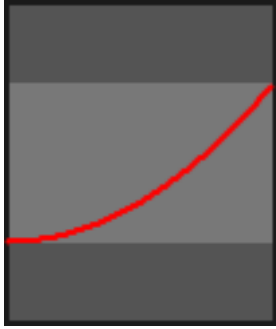
static **in\_out\_quint**(*progress*)



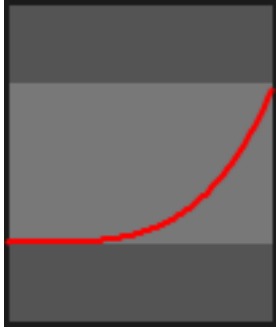
static **in\_out\_sine**(*progress*)



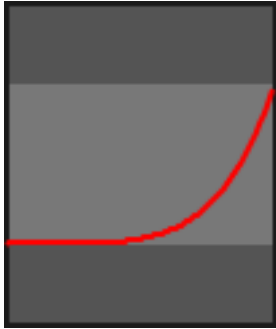
static **in\_quad**(*progress*)



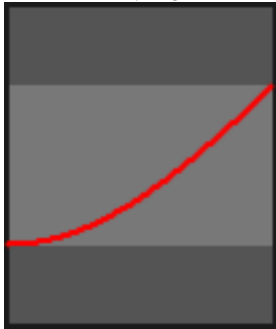
`static in_quart(progress)`



`static in_quint(progress)`

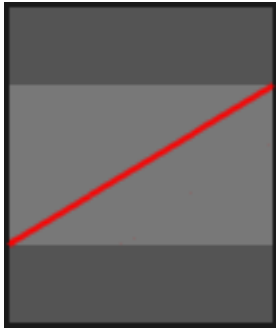


`static in_sine(progress)`

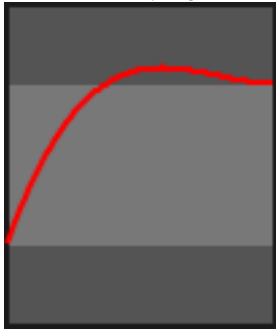


`static linear(progress)`

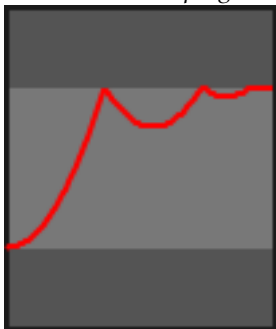




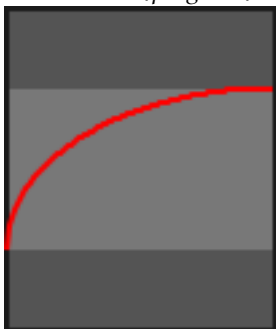
`static out_back(progress)`



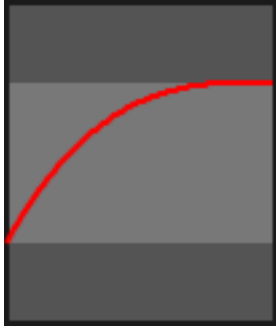
`static out_bounce(progress)`



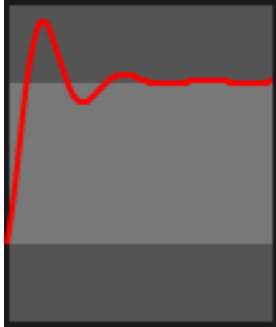
`static out_circ(progress)`



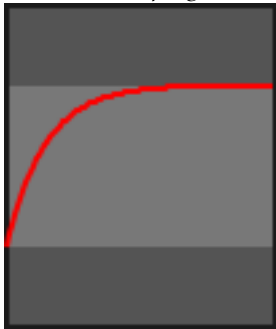
`static out_cubic(progress)`



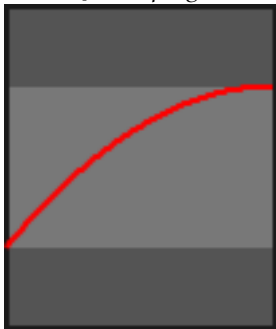
`static out_elastic(progress)`



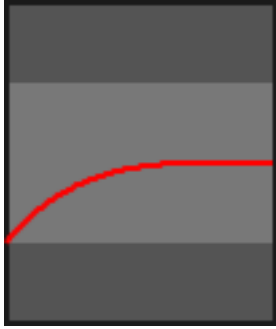
`static out_expo(progress)`



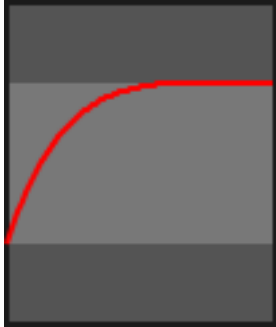
`static out_quad(progress)`



`static out_quart(progress)`



`static out Quint(progress)`



`static out Sine(progress)`





# 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.

## 30.1 Creating an Application

### 30.1.1 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).

### 30.1.2 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 world'
```

Contents of 'main.py':

```
'''
Application from a .kv
=====

The root application is created from the corresponding .kv. Check the test.kv
file to see what will be the 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 relation between main.py and test.kv is explained in `App.load_kv()`.

## 30.2 Application configuration

New in version 1.0.7.

### 30.2.1 Use the configuration file

Your application might want to have its own configuration file. The `App` is able to handle an INI file automatically. You add your section/key/value in the `App.build_config()` method by using the `config` parameter (which is 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 in the config, a file is created on the disk and named from the mangled name of your class. "TestApp" will give a config file-name "test.ini" with the content:

```
[section1]
key1 = value1
key2 = 42
```

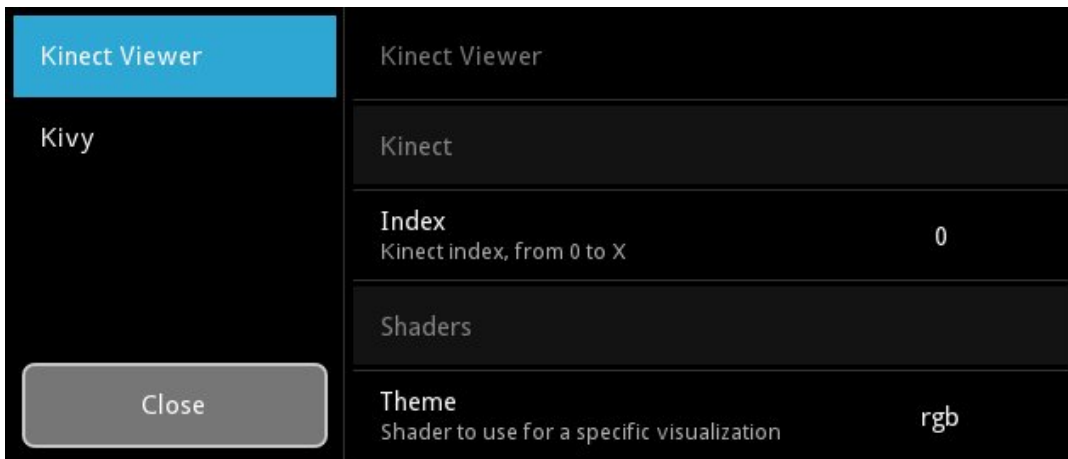
The “test.ini” will be automatically loaded at runtime and you can access the configuration in your `App.build()` method:

```
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')))
```

### 30.2.2 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",
    "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 have been changed to', value)
            elif token == ('section1', 'key2'):
                print('Our key2 have 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

```



## 30.3 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.

## 30.4 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.

## 30.5 Pause mode

New in version 1.1.0.

**Warning:** This mode is experimental, and designed for phones/tablets. There are some cases where your application could crash on resume.

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 (default case), then `App.on_stop()` is called.
4. Otherwise 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.

```
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.

### **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='%(appdir)s/%(appname)s.ini')

New in version 1.0.7.

Changed in version 1.4.0: Customized the default path for iOS and Android platforms. Added a `defaultpath` parameter for desktop OS's (not applicable to iOS and Android.)

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: `/sdcard/.<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 tilda `'~'` 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**

## 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'
```

---

### 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.

#### **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 (the default).

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 False.

New in version 1.1.0.

#### **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** (*\*largs*)

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 1.8.1.

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 to 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 Android, `/sdcard/<app_name>` is returned.

On Windows, `%APPDATA%/<app_name>` is returned.

On Mac OSX, `~/Library/Application Support/<app_name>` is returned.

On Linux, `$XDG_CONFIG_HOME/<app_name>` is returned.



# ATLAS

New in version 1.1.0.

Atlas is a class for managing texture atlases: packing multiple textures into one. With it, you reduce the number of images loaded and speedup the application loading.

An Atlas is composed of:

- a json file (.atlas) that contains all the information about the images contained inside the atlas.
- one or multiple atlas images associated with the atlas definition.

## 31.1 Definition of .atlas

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 of the Kivy `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 ...
  }
}
```

## 31.2 How to create an Atlas

**Warning:** The atlas creation requires Imaging/PIL. This will be removed in the future when the Kivy core Image is able to support loading / blitting / 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 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 256 *.png
Atlas created at myatlas.atlas
1 image have 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`.

---

**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`

---

## 31.3 How to use an Atlas

Usually, you would use the atlas as follows:

```
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:

```
atlas://path/to/myatlas/id
# will search for the "path/to/myatlas.atlas" and get the image "id"
```

In our case, it would be:

```
atlas://images/myatlas/button
```

---

**Note:** In the atlas url, there is no need to add the `.atlas` extension. It will be automatically append to the filename.

---

## 31.4 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.

**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`.

### textures

List of available textures within the atlas.

`textures` is a `DictProperty` and defaults to `{}`.



# KIVY BASE

This module contains core Kivy functionality and is not intended for end users. Feel free to look though it, but calling any of these methods directly may well result in unpredictable behavior.

## 32.1 Event loop management

`kivy.base.EventLoop = <kivy.base.EventLoopBase object at 0x9d2277c>`  
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).

**close**()

Exit from the main loop and stop all configured input providers.

**dispatch\_input**()

Called by `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 `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\_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 `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: `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.STOP`.

**class kivy.base.ExceptionManagerBase**

`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 instance at 0x9f5c78c>`

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:

**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 usefull for embedding Kivy in another toolkit. (like Qt, check kivy-designed)

`kivy.base.stopTouchApp()`

Stop the current application by leaving the main loop





## 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
text = 'objectid'
instance = Label(text=text)
Cache.append('mycache', text, instance)

# retrieve the cached object
instance = Cache.get('mycache', label)
```

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: 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.

## 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 removed.

---

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 orig-

---

inal 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()
```

---

## 34.1 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
```

## 34.2 Triggered Events

New in version 1.0.5.

A triggered event is a way to defer a callback exactly like `schedule_once()`, but with some added convenience. The callback will only be scheduled once per frame even if you call the trigger twice (or more). 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
t = Clock.create_trigger(my_callback)
t()
t()

```

Before triggered events, you may have used this approach in a widget:

```

def trigger_callback(self, *largs):
    Clock.unschedule(self.callback)
    Clock.schedule_once(self.callback)

```

As soon as you call `trigger_callback()`, it will correctly schedule the callback once in the next frame. 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.

---

**Note:** `ClockBase.create_trigger()` also has a `timeout` parameter that behaves exactly like `ClockBase.schedule_once()`.

---

## 34.3 Threading

New in version 1.8.1.

Often, other threads are used to schedule callbacks with kivy's main thread using `ClockBase`. Therefore, it's important to know what is thread safe and what isn't.

All the `ClockBase` and `ClockEvent` methods are safe with respect to kivy's thread. That is, it's always safe to call these methods from a single thread that is not the kivy thread. However, there are no guarantees as to the order in which these callbacks will be executed.

Calling a previously created trigger from two different threads (even if one of them is the kivy thread), or calling the trigger and its `ClockEvent.cancel()` method from two different threads at the same time is not safe. That is, although no exception will be raised, there are no guarantees that calling the trigger from two different threads will not result in the callback being executed twice, or not executed at all. Similarly, such issues might arise when calling the trigger and canceling it with `ClockBase.unschedule()` or `ClockEvent.cancel()` from two threads simultaneously.

Therefore, it is safe to call `ClockBase.create_trigger()`, `ClockBase.schedule_once()`, `ClockBase.schedule_interval()`, or call or cancel a previously created trigger from an external

thread. The following code, though, is not safe because it calls or cancels from two threads simultaneously without any locking mechanism:

```
event = Clock.create_trigger(func)

# in thread 1
event()
# in thread 2
event()
# now, the event may be scheduled twice or once

# the following is also unsafe
# in thread 1
event()
# in thread 2
event.cancel()
# now, the event may or may not be scheduled and a subsequent call
# may schedule it twice
```

Note, in the code above, thread 1 or thread 2 could be the kivy thread, not just an external thread.

`kivy.clock.Clock = None`  
Instance of `ClockBase`.

`class kivy.clock.ClockBase`  
Bases: `kivy.clock._ClockBase`

A clock object with event support.

**create\_trigger**(*callback, timeout=0*)

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.

**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.

## **max\_iteration**

New in version 1.0.5: When a `schedule_once` is used with -1, you can add a limit on how iteration will be allowed. That is here to prevent too much relayout.

## **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()`).

## **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.

## **unschedule**(*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 is scheduled.

**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 1.8.1: The *all* parameter was added. Before, it behaved as if *all* was `True`.

## **class kivy.clock.ClockEvent**(*clock, loop, callback, timeout, starttime, cid, trigger=False*)

Bases: `object`

A class that describes a callback scheduled with kivy's `Clock`. This class is never created by the user; instead, kivy creates and returns an instance of this class when scheduling a callback.

**Warning:** Most of the methods of this class are internal and can change without notice. The only exception are the `cancel()` and `__call__()` methods.

## **cancel()**

Cancels the callback if it was scheduled to be called.

## **kivy.clock.mainthread**(*func*)

Decorator that will schedule the call of the function 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 call in the main thread')
```

```
self.req = URLRequest(url='http://...', on_success=callback)
```

New in version 1.8.0.



# COMPATIBILITY MODULE FOR PYTHON 2.7 AND > 3.3

`kivy.compat.PY2 = True`  
True if Python 2 interpreter is used

`kivy.compat.string_types`  
String types that can be used for checking if a object is a string  
alias of `basestring`



# 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.

Note: To avoid instances where the config settings do not work or they are not applied before window creation (like setting an initial window size), `Config.set` should be used before importing any modules that affect the application window (ie. importing `Window`). Ideally, these settings should be declared right at the start of your `main.py` script.

## 36.1 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()
```

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.

## 36.2 Available configuration tokens

### kivy

**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.

**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.

**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 '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.

**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

**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* and 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.

**'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** Show the cursor on the screen.

**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*.

**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.

**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 1.8.0: *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`

Kivy configuration object. Its `name` is 'kivy'

### `class kivy.config.ConfigParser(name='')`

Bases: `ConfigParser.ConfigParser`, `object`

Enhanced `ConfigParser` class that supports the addition of default sections and default values.

By default, the kivy `ConfigParser` instance, `Config`, is given the name 'kivy' and the `ConfigParser` instance used by App, `build_settings()`, is given the name 'app'.

#### Parameters

`name: string` The name of the instance. See `name`. Defaults to ''.

**..versionchanged:: 1.8.1** Each `ConfigParser` can now be named, `name`. 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/key changed. If you don't specify a section or a key, it will call the callback for all section/keys 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.

### `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 an option. If not found, it will return the default value.

### `getdefaultint(section, option, defaultvalue)`

Get an option. If not found, it will return the default value. The return value will be always converted 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 dynamically changed 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 1.8.1: `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 1.8.1.

**set(section, option, value)**

Functions similarly to `PythonConfigParser`'s `set` method, except that the value is implicitly converted to a string.

**setall(section, keyvalues)**

Set a lot of keys/values in one section at the same time.

**setdefault(section, option, value)**

Set the default value of a particular option.

**setdefaults(section, keyvalues)**

Set a lot of keys/value defaults in one section at the same time.

**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.





# 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.

`kivy.context.register_context(name, cls, *args, **kwargs)`

Register a new context.

`kivy.context.get_current_context()`

Return the current context.



# 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.

## 38.1 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 on the file type.

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. If you have issue with GStreamer, have a look at *GStreamer compatibility*

**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.

**play()**

Play the file.

**seek(position)**

Go to the <position> (in seconds).

**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**

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.

## 38.2 Camera

Core class for acquiring the camera and converting its input into a [Texture](#).

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. If you have issue with GStreamer, have a look at [GStreamer compatibility](#)

```
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 doesnt 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\_frame*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

## 38.3 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:

```
>>> from kivy.core.clipboard import Clipboard
>>> Clipboard.get_types()
['TIMESTAMP', 'TARGETS', 'MULTIPLE', 'SAVE_TARGETS', 'UTF8_STRING',
'COMPOUND_TEXT', 'TEXT', 'STRING', 'text/plain;charset=utf-8',
'text/plain']
>>> Clipboard.get('TEXT')
'Hello World'
>>> Clipboard.put('Great', 'UTF8_STRING')
>>> Clipboard.get_types()
```

```
[ 'UTF8_STRING' ]
>>> Clipboard.get('UTF8_STRING')
'Great'
```

---

**Note:** The main implementation relies on Pygame and works well with text/strings. Anything else might not work the same on all platforms.

---

## 38.4 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.

## 38.5 Image

Core classes for loading images and converting them to a **Texture**. The raw image data can be kept in memory for further access.

**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 or Image object.] A string is interpreted as a path to the image to be loaded. You can also provide a texture object or an already existing image object. In the latter 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.

*scale*[float, defaults to 1.0] Scale of the image.

*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.

### **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.

### 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*)  
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 True.

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(...)
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.

**size**  
Image size (width, height)

**texture**  
Texture of the image

**width**  
Image width

**class** `kivy.core.image.ImageData`(*width*, *height*, *fmt*, *data*, *source=None*, *flip\_vertical=True*,  
*source\_image=None*)

Bases: `object`

Container for images and mipmap images. The container will always have at least the mipmap level 0.

**add\_mipmap**(*level*, *width*, *height*, *data*)  
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

**fmt**  
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.

**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)

## 38.6 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('heLo')
[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: `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'.

**class** `kivy.core.spelling.NoSuchLangError`

Bases: `exceptions.Exception`

Exception to be raised when a specific language could not be found.

**class** `kivy.core.spelling.NoLanguageSelectedError`

Bases: `exceptions.Exception`

Exception to be raised when a language-using method is called but no language was selected prior to the call.

## 38.7 Text

An abstraction of text creation. Depending of the selected backend, the accuracy of text rendering may vary.

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  $\leq 1$ .

This is the backend layer for getting text out of different text providers, you should only be using this directly if your needs aren't fulfilled by the `Label`.

Usage example:

```
from kivy.core.label 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
```

```
class kivy.core.text.LabelBase(text="", font_size=12, font_name='DroidSans', bold=False,  
italic=False, halign='left', valign='bottom', shorten=False,  
text_size=None, mipmap=False, color=None, line_height=1.0,  
strip=False, shorten_from='center', split_str='', **kwargs)
```

Bases: `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\_name: str, defaults to DEFAULT\_FONT**Font name of the text

**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.

Changed in version 1.8.1: *strip*, *shorten\_from*, and *split\_str* were added.

Changed in version 1.8.1: *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.

### **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 1.8.1.

**get\_extents**(*text*)

Return a tuple (width, height) indicating the size of the specified text

**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`.

**Retruns** 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.

### 38.7.1 Text Markup

New in version 1.1.0.

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 might have a closed tag associated: `[/tag]`. Example of a markup text:

```
[b>Hello [color=ff0000]world[/b]][/color]
```

The following tags are availables:

**[b][/b]** Activate bold text

**[i][/i]** Activate italic text

**[font=<str>][/font]** Change the font

**[size=<integer>][/size]** Change the font size

**[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.

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)), ... }
```

#### **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.

## 38.7.2 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()`

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 `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: 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.

#### class kivy.core.text.text\_layout.LayoutLine

Bases: 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.

## 38.8 Video

Core class for reading video files and managing the `kivy.graphics.texture.Texture` video.

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. If you have issue with GStreamer, have a look at [GStreamer compatibility](#)

---

**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 unknown: 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*)**  
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%)

## 38.9 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()

        # 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**

**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.

**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.

**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** 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.

Changed in version 1.8.1: `on_request_close` has been added.

**add\_widget(*widget*)**

Add a widget to a window

**center**

Center of the rotated window.

`center` is a `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).

**close()**

Close the window

### **create\_window(\*args)**

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

### **fullscreen**

This property sets the fullscreen mode of the window. Available options are: `True`, `False`, `'auto'`, `'fake'`. Check the `config` documentation for a more detailed explanation on the values.

New in version 1.2.0.

### **height**

Rotated window height.

`height` is a read-only `AliasProperty`.

### **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.

..versionadded:: 1.8.1

`keyboard_height` is a read-only `AliasProperty` defaults to 0.

### **modifiers**

List of keyboard modifiers currently active.

### **mouse\_pos**

2d position of the mouse within the window.

New in version 1.2.0.

### **on\_close(\*args)**

Event called when the window is closed

### **on\_dropfile(filename)**

Event called when a file is dropped on the application.

**Warning:** This event is currently used only on MacOSX with a patched version of pygame, but is left in place for further evolution (ios, android etc.)

New in version 1.2.0.

**on\_flip()**

Flip between buffers (event)

**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\_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\_rotate**(*rotation*)

Event called when the screen has been rotated.

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

Event called when a touch down event is initiated.

Changed in version 1.8.1: 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 1.8.1: 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 1.8.1: 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.

### **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.

### **rotation**

Get/set the window content rotation. Can be one of 0, 90, 180, 270 degrees.

### **screenshot(name='screenshot{:04d}.png')**

Save the actual displayed image in a file

### **set\_icon(filename)**

Set the icon of the window.

New in version 1.0.5.

**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`.

**size**

Get the rotated size of the window. If `rotation` is set, then the size will change to reflect the rotation.

**softinput\_mode**

This specifies the behavior of window contents on display of soft keyboard on mobile platform. Can be one of `''`, `'pan'`, `'scale'`, `'resize'`.

When `''` The main window is left as it is allowing the user to use `keyboard_height` to manage the window contents the way they want.

when `'pan'` The main window pans moving the bottom part of the window to be always on top of the keyboard.

when `'resize'` The window is resized and the contents scaled to fit the remaining space.

..versionadded::1.8.1

`softinput_mode` is a `OptionProperty` defaults to None.

**system\_size**

Real size of the window ignoring rotation.

**toggle\_fullscreen**()

Toggle fullscreen on window

**width**

Rotated window width.

`width` is a read-only `AliasProperty`.

`kivy.core.window.Window = None`

Instance of a `WindowBase` implementation





# 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.

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. If you have issue with GStreamer, have a look at [GStreamer compatibility](#)

---

**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`.

**play()**

Play the file.

**seek(*position*)**

Go to the <position> (in seconds).

**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**

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.

---

# CAMERA

Core class for acquiring the camera and converting its input into a [Texture](#).

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. If you have issue with GStreamer, have a look at [GStreamer compatibility](#)

```
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 doesnt 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\_frame* 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



# 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:

```
>>> from kivy.core.clipboard import Clipboard
>>> Clipboard.get_types()
['TIMESTAMP', 'TARGETS', 'MULTIPLE', 'SAVE_TARGETS', 'UTF8_STRING',
 'COMPOUND_TEXT', 'TEXT', 'STRING', 'text/plain;charset=utf-8',
 'text/plain']
>>> Clipboard.get('TEXT')
'Hello World'
>>> Clipboard.put('Great', 'UTF8_STRING')
>>> Clipboard.get_types()
['UTF8_STRING']
>>> Clipboard.get('UTF8_STRING')
'Great'
```

---

**Note:** The main implementation relies on Pygame and works well with text/strings. Anything else might not work the same on all platforms.

---



# 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.





# IMAGE

Core classes for loading images and converting them to a **Texture**. The raw image data can be kept in memory for further access.

```
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 or Image object.] A string is interpreted as a path to the image to be loaded. You can also provide a texture object or an already existing image object. In the latter 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.

*scale*[float, defaults to 1.0] Scale of the image.

*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.

## **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.

### 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)

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 True.

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(...)
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.

**size**

Image size (width, height)

**texture**

Texture of the image

**width**

Image width

```
class kivy.core.image.ImageData(width, height, fmt, data, source=None, flip_vertical=True,
                                source_image=None)
```

Bases: object

Container for images and mipmap images. The container will always have at least the mipmap level 0.

**add\_mipmap**(level, width, height, data)

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

**fmt**

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.

**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)

# 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('heLo')
[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: `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'.

**class** kivy.core.spelling.NoSuchLangError

Bases: exceptions.Exception

Exception to be raised when a specific language could not be found.

**class** kivy.core.spelling.NoLanguageSelectedError

Bases: exceptions.Exception

Exception to be raised when a language-using method is called but no language was selected prior to the call.

# TEXT

An abstraction of text creation. Depending of the selected backend, the accuracy of text rendering may vary.

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  $\leq 1$ .

This is the backend layer for getting text out of different text providers, you should only be using this directly if your needs aren't fulfilled by the `Label`.

Usage example:

```
from kivy.core.label 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
```

```
class kivy.core.text.LabelBase(text='', font_size=12, font_name='DroidSans', bold=False,
                               italic=False, halign='left', valign='bottom', shorten=False,
                               text_size=None, mipmap=False, color=None, line_height=1.0,
                               strip=False, shorten_from='center', split_str='', **kwargs)
```

Bases: `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\_name:** `str`, defaults to `DEFAULT_FONT`Font name of the text

**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.

Changed in version 1.8.1: *strip*, *shorten\_from*, and *split\_str* were added.

Changed in version 1.8.1: *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.

### **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 1.8.1.

### **get\_extents(text)**

Return a tuple (width, height) indicating the size of the specified text



## 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`.

**Retruns** 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.

## 45.1 Text Markup

New in version 1.1.0.

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 might have a closed tag associated: `[/tag]`. Example of a markup text:

```
[b>Hello [color=ff0000]world[/b][color]
```

The following tags are available:

**[b][/b]** Activate bold text

**[i][/i]** Activate italic text

**[font=<str>][font]** Change the font

**[size=<integer>][size]** Change the font size

**[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.

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)), ... }
```

**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.

## 45.2 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()`

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 it's 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 `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: `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.

**class** `kivy.core.text.text_layout.LayoutLine`

Bases: `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 than 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.



## TEXT MARKUP

New in version 1.1.0.

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 might have a closed tag associated: `[/tag]`. Example of a markup text:

```
[b>Hello [color=ff0000]world[/b][/color]
```

The following tags are availables:

**[b][/b]** Activate bold text

**[i][/i]** Activate italic text

**[font=<str>][/font]** Change the font

**[size=<integer>][/size]** Change the font size

**[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.

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)), ... }
```

## 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.



## 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()`

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 it's 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. Its 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: 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.

`class kivy.core.text.text_layout.LayoutLine`

Bases: `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.



# VIDEO

Core class for reading video files and managing the `kivy.graphics.texture.Texture` video.

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. If you have issue with GStreamer, have a look at [GStreamer compatibility](#)

---

**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 unknown: 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)

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%)

# 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()

    # 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**

**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.

**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.

**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**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 be 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.

Changed in version 1.8.1: *on\_request\_close* has been added.

### **add\_widget**(*widget*)

Add a widget to a window

### **center**

Center of the rotated window.

**center** is a **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).

### **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

### **fullscreen**

This property sets the fullscreen mode of the window. Available options are: True, False, 'auto', 'fake'. Check the [config](#) documentation for a more detailed explanation on the values.

New in version 1.2.0.

### **height**

Rotated window height.

**height** is a read-only [AliasProperty](#).

### **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.

..versionadded:: 1.8.1

**keyboard\_height** is a read-only [AliasProperty](#) defaults to 0.

### **modifiers**

List of keyboard modifiers currently active.

### **mouse\_pos**

2d position of the mouse within the window.

New in version 1.2.0.

### **on\_close(\*largs)**

Event called when the window is closed

### **on\_dropfile(filename)**

Event called when a file is dropped on the application.

**Warning:** This event is currently used only on MacOSX with a patched version of pygame, but is left in place for further evolution (ios, android etc.)

New in version 1.2.0.

### **on\_flip()**

Flip between buffers (event)

### **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\_motion(etype, me)**

Event called when a Motion Event is received.

#### **Parameters**

**etype:** strOne 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\_rotate** (*rotation*)

Event called when the screen has been rotated.

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

Event called when a touch down event is initiated.

Changed in version 1.8.1: 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 1.8.1: 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 1.8.1: 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.

**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.

**rotation**

Get/set the window content rotation. Can be one of 0, 90, 180, 270 degrees.

**screenshot** (*name='screenshot{:04d}.png'*)

Save the actual displayed image in a file

**set\_icon** (*filename*)

Set the icon of the window.

New in version 1.0.5.

**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**.

**size**

Get the rotated size of the window. If **rotation** is set, then the size will change to reflect the rotation.

**softinput\_mode**

This specifies the behavior of window contents on display of soft keyboard on mobile platform. Can be one of "", 'pan', 'scale', 'resize'.

When "" The main window is left as it is allowing the user to use **keyboard\_height** to manage the window contents the way they want.

when 'pan' The main window pans moving the bottom part of the window to be always on top of the keyboard.

when 'resize' The window is resized and the contents scaled to fit the remaining space.

..versionadded::1.8.1

`softinput_mode` is a `OptionProperty` defaults to `None`.

**system\_size**

Real size of the window ignoring rotation.

**toggle\_fullscreen()**

Toggle fullscreen on window

**width**

Rotated window width.

`width` is a read-only `AliasProperty`.

`kivy.core.window.Window = None`

Instance of a `WindowBase` implementation



---

# 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.

## 50.1 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

## 50.2 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.

`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.

## 50.3 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.

## 50.4 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.

### **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.

## 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



## 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.

*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 frametime 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.

# 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.





# 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.

**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`.

# 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`  
Bases: `kivy.event.ObjectWithUid`  
Generic event dispatcher interface.  
See the module docstring for usage.

**bind()**  
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):
    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()

```

### create\_property()

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 1.8.1: In the past, if *value* was of type *bool*, a *NumericProperty* would be created, now a *BooleanProperty* is created.

**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.

```
>>> mywidget = Widget()
>>> mywidget.create_property('custom')
>>> mywidget.custom = True
>>> print(mywidget.custom)
True
```

#### **dispatch()**

Dispatch an event across all the handlers added in `bind()`. 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*: **str**the event name to dispatch.

Changed in version 1.8.1: Keyword arguments collection and forwarding was added. Before, only positional arguments would be collected and forwarded.

#### **events()**

Return all the events in the class. Can be used for introspection.

New in version 1.8.0.

#### **get\_property\_observers()**

Returns a list of methods that are bound to the property/event passed as the *name* argument:

```
widget_instance.get_property_observers('on_release')
```

New in version 1.8.0.

#### **getter()**

Return the getter of a property.

New in version 1.0.9.

#### **is\_event\_type()**

Return True if the *event\_type* is already registered.

New in version 1.0.4.

#### **properties()**

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()**

Get a property instance from the name.

New in version 1.0.9.

Returns A **Property** derived instance corresponding to the name.

### **register\_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()**

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()**

Unbind properties from callback functions.

Same usage as `bind()`.

### **unregister\_event\_types()**

Unregister an event type in the dispatcher.

**class** kivy.event.**ObjectWithUid**

Bases: object

(internal) This class assists in providing unique identifiers for class instances. It is not intended for direct usage.





## EXTENSION SUPPORT

Sometimes your application requires functionality that is beyond the scope of what Kivy can deliver. In those cases it is necessary to resort to external software libraries. Given the richness of the Python ecosystem, there is already a great number of software libraries that you can simply import and use right away.

For some third-party libraries, it's not as easy as that though. Some libraries require special *wrappers* to be written for them in order to be compatible with Kivy. Some libraries might even need to be patched so that they can be used (e.g. if they open their own OpenGL context to draw in and don't support proper offscreen rendering). On those occasions it is often possible to patch the library in question and to provide a Python wrapper around it that is compatible with Kivy. Sticking with this example, you can't just use the wrapper with a 'normal' installation of the library because the patch would be missing.

That is where Kivy extensions come in handy. A Kivy extension represents a single third-party library that is provided in a way so that it can simply be downloaded as a single file, put in a special directory and then offers the functionality of the wrapped library to Kivy applications. These extensions will not pollute the global Python environment (as they might be unusable on their own after potential patches have been applied) because they reside in special directories for Kivy that are not accessed by Python by default.

Kivy extensions are provided as `*.kex` files. They are really just zip files, but you must not unzip them yourself. Kivy will do that for you as soon as it's appropriate to do so.

**Warning:** Again, do not try to unzip `*.kex` files on your own. While unzipping will work, Kivy will not be able to load the extension and will simply ignore it.

With Kivy's extension system, your application can use specially packaged third-party libraries in a backwards compatible way (by specifying the version that you require) even if the actual third-party library does not guarantee backwards-compatibility. There will be no breakage if newer versions are installed (as a properly suited old version will still be used). For more information about that behaviour, consider the documentation of the `load()` function.

If you want to provide an extension on your own, there is a helper script that sets up the initial extension folder structure that Kivy requires for extensions. It can be found at `kivy/tools/extensions/make-kivyext.py`

`kivy.ext.load(extname, version)`

Use this function to tell Kivy to load a specific version of the given Extension. This is different from `kivy.require()` in that it will always use the exact same major version you specify even if a newer (major) version is available. This is because we cannot make the same backwards-compatibility guarantee that we make with Kivy for third-party extensions. You will still get fixes and optimizations that don't break backwards compatibility via minor version upgrades of the extension.

The function will then return the loaded module as a Python module object and you can bind it to a name of your choosing. This prevents clashes with modules with the same name that might

be installed in a system directory.

Usage example for this function:

```
from kivy.ext import load
myextension = load('myextension', (2, 1))
# You can now use myextension as if you had done 'import myextension',
# but with the added benefit of using the proper version.
```

### Parameters

**extname:** **str**The exact name of the extension that you want to use.

**version:** **two-tuple of ints**A tuple of the form (major, minor), where major and minor are ints that specify the major and minor version number for the extension, e.g. (1, 2) would be akin to 1.2. It is important to note that between minor versions, backwards compatibility is guaranteed, but between major versions it is not. I.e. if you change your extension in a backwards incompatible way, increase the major version number (and reset the minor to 0). If you just do a bug fix or add an optional, backwards-compatible feature, you can just increase the minor version number. If the application then requires version (1, 2), every version starting with that version number will be ok and by default the latest version will be chosen. The two ints major and minor can both be in range(0, infinity).

### kivy.ext.unzip\_extensions()

Unzips Kivy extensions. Internal usage only: don't use it yourself unless you know what you're doing and really want to trigger installation of new extensions.

For your file to be recognized as an extension, it has to fulfil a few requirements:

- We require that the file has the \*.kex extension to make the distinction between a Kivy extension and an ordinary zip file clear.
- We require that the \*.kex extension files be put into any of the directories listed in EXTENSION\_PATHS which is normally ~/.kivy/extensions and extensions/ inside kivy's base directory. We do not look for extensions on sys.path or elsewhere in the system.
- We require that the Kivy extension is zipped in a way so that Python's zipfile module can extract it properly.
- We require that the extension internally obeys the common Kivy extension format, which looks like this:

```
|-- myextension/
    |-- __init__.py
    |-- data/
```

The `__init__.py` file is the main entrypoint to the extension. All names that should be usable when the extension is loaded need to be exported (i.e. made available) in the namespace of that file.

How the extension accesses the code of the library that it wraps (be it pure Python or binary code) is up to the extension. For example there could be another Python module adjacent to the `__init__.py` file from which the `__init__.py` file imports the usable names that it wants to expose.

- We require that the version of the extension be specified in the `setup.py` file that is created by the Kivy extension wizard and that the version specification format as explained in `load()` be used.

## 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 at 0x905c88c>
    Factory instance to use for getting new classes
```



# GARDEN

New in version 1.7.0.

Changed in version 1.8.0.

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.

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.

---

## 58.1 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
```

## 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](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)





---

# 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*)

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](mailto: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: 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**

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.

# 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.

## 61.1 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 widgets position or size. You therefore you 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.

---

## 61.2 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 MacOSX 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**

Bases: *kivy.graphics.instructions.VertexInstruction*

A 2d Bezier curve.

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**

Bases: *kivy.graphics.instructions.ContextInstruction*

BindTexture Graphic instruction. 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`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

A 2d border image. The behavior of the border image is similar to the concept of a CSS3 border-image.

**Parameters**

*border*: **list** Border information in the format (top, right, bottom, left). Each value is in pixels.

**border**

Property for getting/setting the border of the class.

**class** `kivy.graphics.Callback`

Bases: `kivy.graphics.instructions.Instruction`

New in version 1.0.4.

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.

### **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**

Bases: `kivy.graphics.instructions.CanvasBase`

The important Canvas class. Use this class to add graphics or context 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))
```

### **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);
```

**class** kivy.graphics.**CanvasBase**

Bases: `kivy.graphics.instructions.InstructionGroup`

CanvasBase provides the context manager methods for the `Canvas`.

**class** kivy.graphics.**Color**

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction to set the color state for any vertices being drawn after it. All the values passed are between 0 and 1, not 0 and 255.

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')
```

In kv lang:

```
<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.ContextInstruction**

Bases: [kivy.graphics.instructions.Instruction](#)

The ContextInstruction class is the base for the creation of instructions 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**

Bases: [kivy.graphics.vertex\\_instructions.Rectangle](#)

A 2D ellipse.

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**

Bases: [kivy.graphics.instructions.RenderContext](#)

Fbo class for wrapping the OpenGL Framebuffer extension. The Fbo support "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 1.8.1.

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 1.8.1

---

### **add\_reload\_observer()**

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()**

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()**

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

**class kivy.graphics.GraphicException**

Bases: `exceptions.Exception`

Exception raised when a graphics error is fired.

**class kivy.graphics.Instruction**

Bases: `kivy.event.ObjectWithUid`

Represents the smallest instruction available. This class is for internal usage only, don't use it directly.

**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**

Bases: `kivy.graphics.instructions.Instruction`

Group of `Instructions`. Allows for the adding and 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()**

Add a new `Instruction` to our list.

**clear()**

Remove all the `Instructions`.

**get\_group()**

Return an iterable for all the `Instructions` with a specific group name.

**insert()**

Insert a new `Instruction` into our list at index.

**remove()**

Remove an existing `Instruction` from our list.

**remove\_group()**

Remove all `Instructions` with a specific group name.

**class kivy.graphics.Line**

Bases: `kivy.graphics.instructions.VertexInstruction`

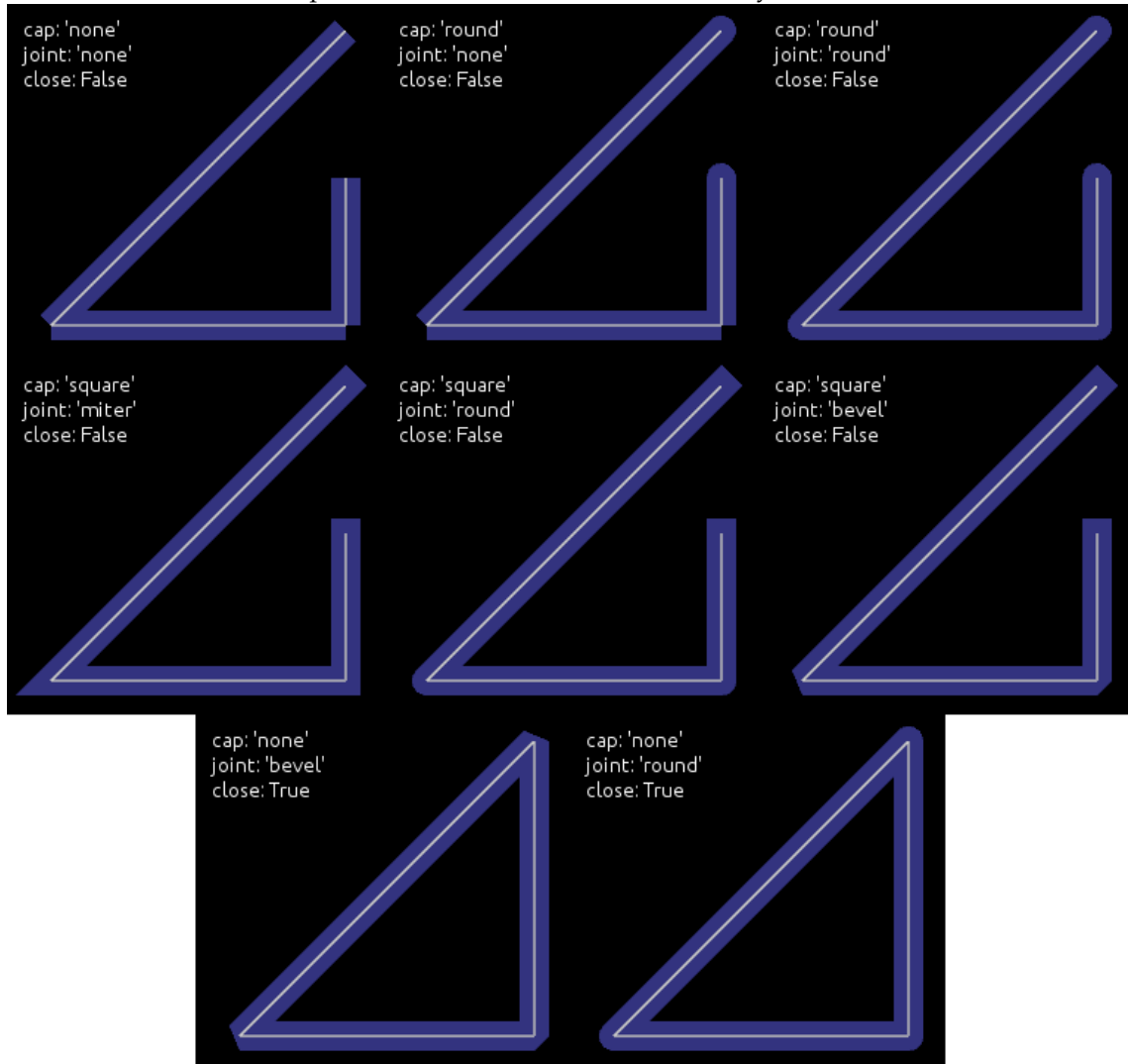
A 2d line.

Drawing a line can be done easily:

```
with self.canvas:
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

Actually, the line have 3 internal drawing mode that you should know about if you want to get the best performance of it:

- 1.If the **width** is 1.0, then we will use standard GL\_LINE drawing from OpenGL. **dash\_length** and **dash\_offset** works, while properties for cap and joint have no sense for this.
- 2.If the **width** is > 1.0, then we will use a custom drawing method, based on triangles. **dash\_length** and **dash\_offset** is not working on that mode. Additionally, if the current color have an alpha < 1.0, 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), default 1
- dash\_offset:** int Offset between the end of a segments and the begining of the next one, default 0, changing this makes it dashed.
- width:** float Width of the line, default 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
- close:** bool, defaults to False If True, the line will be closed.

**circle:** listIf set, the **points** will be set to build a circle. Check **circle** for more information.

**ellipse:** listIf set, the **points** will be set to build an ellipse. Check **ellipse** for more information.

**rectangle:** listIf set, the **points** will be set to build a rectangle. Check **rectangle** for more information.

**bezier:** listIf set, the **points** will be set to build a bezier line. Check **bezier** for more information.

**bezier\_precision:** int, defaults to 180 Precision of the Bezier drawing.

New in version 1.0.8: *dash\_offset* and *dash\_length* have been added

New in version 1.4.1: *width*, *cap*, *joint*, *cap\_precision*, *joint\_precision*, *close*, *ellipse*, *rectangle* have been added.

New in version 1.4.1: *bezier*, *bezier\_precision* 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 calculate 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.

### ellipse

Use this property to build an ellipse, without calculate 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 defaultvalue is 0 and 360.**
- (optional) **segments is the precision of the ellipse. The defaultvalue 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) angle\_end, segments):

- 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 (default to 30)

The line is automatically closed.

Usage:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40, 100))
```

New in version 1.8.1.

## width

Determine the width of the line, defaults to 1.0.

New in version 1.4.1.

## class `kivy.graphics.MatrixInstruction`

Bases: `kivy.graphics.instructions.ContextInstruction`

Base class for Matrix Instruction on the canvas.

## 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' or 'projection\_mat'.

New in version 1.6.0.

## class kivy.graphics.Mesh

Bases: [kivy.graphics.instructions.VertexInstruction](#)

A 2d mesh.

The format for vertices is currently fixed but this might change in a future release. Right now, each vertex is described with 2D coordinates (x, y) and a 2D texture coordinate (u, v).

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:** list List of vertices in the format (x1, y1, u1, v1, x2, y2, u2, v2...).

**indices:** list List of indices in the format (i1, i2, i3...).

**mode:** str Mode of the vbo. Check [mode](#) for more information. Defaults to 'points'.

## 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

Bases: [kivy.graphics.instructions.VertexInstruction](#)

A 2d line.

### Parameters

**points:** list List of points in the format (x1, y1, x2, y2...).

**pointsize:** float, defaults to 1. Size of the point (1. means the real size will be 2).

**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()

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 points of the triangle.

**pointsize**

Property for getting/setting point size.

**class kivy.graphics.PopMatrix**

Bases: [kivy.graphics.instructions.ContextInstruction](#)

Pop the matrix from the context's matrix stack onto the model view.

**stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

**class kivy.graphics.PushMatrix**

Bases: [kivy.graphics.instructions.ContextInstruction](#)

Push the matrix onto the context's matrix stack.

**stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

**class kivy.graphics.Quad**

Bases: [kivy.graphics.instructions.VertexInstruction](#)

A 2d quad.

**Parameters**

*points*: listList 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**

Bases: [kivy.graphics.instructions.VertexInstruction](#)

A 2d rectangle.

**Parameters**

*pos*: listPosition of the rectangle, in the format (x, y).

*size*: listSize 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**

Bases: [kivy.graphics.instructions.Canvas](#)

The render context stores all the necessary information for drawing, i.e.:

- 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\_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**

Bases: `kivy.graphics.context_instructions.Transform`

Rotate the coordinate space by applying a rotation transformation 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()**

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**

Bases: `kivy.graphics.context_instructions.Transform`

Instruction to create a non uniform scale transformation.

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.

## 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

Bases: [kivy.graphics.instructions.Instruction](#)

Use current stencil buffer as a mask. Check the module documentation for 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

Bases: [kivy.graphics.context\\_instructions.Transform](#)

Instruction to create a translation of the model view coordinate space.

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**

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d triangle.

**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**

Bases: `kivy.graphics.instructions.Instruction`

The VertexInstruction class is the base for all graphics instructions 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, y + h, u, y + 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

Bases: `kivy.graphics.instructions.Instruction`

ClearColor Graphics 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

Bases: `kivy.graphics.instructions.Instruction`

Clearbuffer Graphics Instruction.

New in version 1.3.0.

Clear the buffers specified by the instructions buffer mask property. By default, only the color 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.

**class** `kivy.graphics.PushState`

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction that pushes arbitrary states/uniforms onto the context state stack.

New in version 1.6.0.

**class** `kivy.graphics.ChangeState`

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction that changes the values of arbitrary states/uniforms on the current render context.

New in version 1.6.0.

**class** `kivy.graphics.PopState`

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction that pops arbitrary states/uniforms off the context state stack.

New in version 1.6.0.

**class** `kivy.graphics.ApplyContextMatrix`

Bases: `kivy.graphics.instructions.ContextInstruction`

Pre-multiply the matrix at the top of the stack specified by `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' or 'projection\_mat'.

New in version 1.6.0.

**target\_stack**

Name of the matrix stack to use as a target. Can be 'modelview\_mat' or 'projection\_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:  $inverse(transpose(mat3(mvm)))$

New in version 1.6.0.

**class** `kivy.graphics.LoadIdentity`

Bases: `kivy.graphics.instructions.ContextInstruction`

Load the identity Matrix into the matrix stack specified by 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' or 'projection\_mat'.

## 61.3 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`

Bases: `kivy.event.ObjectWithUid`

Represents the smallest instruction available. This class is for internal usage only, don't use it directly.

### 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

Bases: [kivy.graphics.instructions.Instruction](#)

Group of [Instructions](#). Allows for the adding and 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()

Add a new [Instruction](#) to our list.

### clear()

Remove all the [Instructions](#).

### get\_group()

Return an iterable for all the [Instructions](#) with a specific group name.

### insert()

Insert a new [Instruction](#) into our list at index.

### remove()

Remove an existing [Instruction](#) from our list.

### remove\_group()

Remove all [Instructions](#) with a specific group name.

### class kivy.graphics.instructions.ContextInstruction

Bases: [kivy.graphics.instructions.Instruction](#)

The ContextInstruction class is the base for the creation of instructions 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

Bases: [kivy.graphics.instructions.Instruction](#)

The VertexInstruction class is the base for all graphics instructions 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, y + h, u, y + 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

Bases: `kivy.graphics.instructions.CanvasBase`

The important Canvas class. Use this class to add graphics or context 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))
```

### **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);
```

### **class kivy.graphics.instructions.CanvasBase**

Bases: **kivy.graphics.instructions.InstructionGroup**

CanvasBase provides the context manager methods for the **Canvas**.

### **class kivy.graphics.instructions.RenderContext**

Bases: **kivy.graphics.instructions.Canvas**

The render context stores all the necessary information for drawing, i.e.:

- 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\_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**

Bases: [kivy.graphics.instructions.Instruction](#)

New in version 1.0.4.

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.

### **reset\_context**

Set this to True if you want to reset the OpenGL context for Kivy after the callback has been called.

## 61.4 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**

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction to set the color state for any vertices being drawn after it. All the values passed are between 0 and 1, not 0 and 255.

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')
```

In kv lang:

```
<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`

Bases: `kivy.graphics.instructions.ContextInstruction`

`BindTexture` Graphic instruction. 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`

Bases: `kivy.graphics.instructions.ContextInstruction`

Push the matrix onto the context's matrix stack.

**stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

**class** `kivy.graphics.context_instructions.PopMatrix`

Bases: `kivy.graphics.instructions.ContextInstruction`

Pop the matrix from the context's matrix stack onto the model view.

### **stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

### **class kivy.graphics.context\_instructions.Rotate**

Bases: `kivy.graphics.context_instructions.Transform`

Rotate the coordinate space by applying a rotation transformation 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()**

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**

Bases: `kivy.graphics.context_instructions.Transform`

Instruction to create a non uniform scale transformation.

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.

### **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`  
Bases: `kivy.graphics.context_instructions.Transform`  
Instruction to create a translation of the model view coordinate space.  
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`  
Bases: `kivy.graphics.instructions.ContextInstruction`  
Base class for Matrix Instruction on the canvas.

**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' or 'projection\_mat'.  
New in version 1.6.0.

## 61.5 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: `object`

The Context class manages groups of graphics instructions. It can also be used to manage observer callbacks. See `add_reload_observer()` and `remove_reload_observer()` for more information.

### `add_reload_observer()`

(internal) Add a callback to be called after the whole graphics context has been reloaded. This is where you can reload 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.

### `remove_reload_observer()`

(internal) Remove a callback from the observer list previously added by `add_reload_observer()`.

## 61.6 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

        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.

## 61.6.1 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`

Bases: `kivy.graphics.instructions.RenderContext`

Fbo class for wrapping the OpenGL Framebuffer extension. The Fbo support "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 1.8.1.

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 1.8.1

---

### **add\_reload\_observer()**

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()**

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()**

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

## 61.7 GL instructions

New in version 1.3.0.

### 61.7.1 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`

Bases: `kivy.graphics.instructions.Instruction`

ClearColor Graphics 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`

Bases: `kivy.graphics.instructions.Instruction`

Clearbuffer Graphics Instruction.

New in version 1.3.0.

Clear the buffers specified by the instructions buffer mask property. By default, only the color 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.

## 61.8 Graphics compiler

Before rendering an `InstructionGroup`, we compile the group in order to reduce the number of instructions executed at rendering time.

### 61.8.1 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.

## 61.9 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()`

See: [glActiveTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glAttachShader()`

See: [glAttachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindAttribLocation()`  
See: [glBindAttribLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindBuffer()`  
See: [glBindBuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindFramebuffer()`  
See: [glBindFramebuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindRenderbuffer()`  
See: [glBindRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindTexture()`  
See: [glBindTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendColor()`  
See: [glBlendColor\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendEquation()`  
See: [glBlendEquation\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendEquationSeparate()`  
See: [glBlendEquationSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendFunc()`  
See: [glBlendFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendFuncSeparate()`  
See: [glBlendFuncSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glBufferData()`  
See: [glBufferData\(\) on Kronos website](#)

`kivy.graphics.opengl.glBufferSubData()`  
See: [glBufferSubData\(\) on Kronos website](#)

`kivy.graphics.opengl.glCheckFramebufferStatus()`  
See: [glCheckFramebufferStatus\(\) on Kronos website](#)

`kivy.graphics.opengl.glClear()`  
See: [glClear\(\) on Kronos website](#)

`kivy.graphics.opengl.glClearColor()`  
See: [glClearColor\(\) on Kronos website](#)

`kivy.graphics.opengl.glClearStencil()`  
See: [glClearStencil\(\) on Kronos website](#)

`kivy.graphics.opengl.glColorMask()`  
See: [glColorMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompileShader()`  
See: [glCompileShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompressedTexImage2D()`  
See: [glCompressedTexImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompressedTexSubImage2D()`  
See: [glCompressedTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCopyTexImage2D()`  
See: [glCopyTexImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCopyTexSubImage2D()`  
See: [glCopyTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCreateProgram()`  
See: [glCreateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glCreateShader()`  
See: [glCreateShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glCullFace()`  
See: [glCullFace\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteBuffers()`  
See: [glDeleteBuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteFramebuffers()`  
See: [glDeleteFramebuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteProgram()`  
See: [glDeleteProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteRenderbuffers()`  
See: [glDeleteRenderbuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteShader()`  
See: [glDeleteShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteTextures()`  
See: [glDeleteTextures\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthFunc()`  
See: [glDepthFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthMask()`  
See: [glDepthMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glDetachShader()`  
See: [glDetachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisable()`  
See: [glDisable\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisableVertexAttribArray()`  
See: [glDisableVertexAttribArray\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawArrays()`  
See: [glDrawArrays\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawElements()`  
See: [glDrawElements\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnable()`  
See: [glEnable\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnableVertexAttribArray()`  
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()`  
See: [glFramebufferRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glFramebufferTexture2D()`  
See: [glFramebufferTexture2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glFrontFace()`

See: [glFrontFace\(\) on Kronos website](#)

`kivy.graphics.opengl.glGenBuffers()`

See: [glGenBuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenFramebuffers()`

See: [glGenFramebuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenRenderbuffers()`

See: [glGenRenderbuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenTextures()`

See: [glGenTextures\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenerateMipmap()`

See: [glGenerateMipmap\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetActiveAttrib()`

See: [glGetActiveAttrib\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetActiveUniform()`

See: [glGetActiveUniform\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttachedShaders()`

See: [glGetAttachedShaders\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttribLocation()`

See: [glGetAttribLocation\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBooleanv()`

See: [glGetBooleanv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBufferParameteriv()`

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()`

See: [glGetFloatv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetFramebufferAttachmentParameteriv()`

See: [glGetFramebufferAttachmentParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetIntegerv()`

See: [glGetIntegerv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetProgramInfoLog()`

See: [glGetProgramInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetProgramiv()`

See: [glGetProgramiv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetRenderbufferParameteriv()`

See: [glGetRenderbufferParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetShaderInfoLog()`

See: [glGetShaderInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderPrecisionFormat()`

See: [glGetShaderPrecisionFormat\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glGetShaderSource()`

See: [glGetShaderSource\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderiv()`

See: [glGetShaderiv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetString()`

See: [glGetString\(\) on Kronos website](#)

Unlike the C specification, the value will be returned as a string.

`kivy.graphics.opengl.glGetTexParameterfv()`

See: [glGetTexParameterfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetTexParameteriv()`

See: [glGetTexParameteriv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformLocation()`

See: [glGetUniformLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformfv()`

See: [glGetUniformfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformiv()`

See: [glGetUniformiv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetVertexAttribPointerv()`

See: [glGetVertexAttribPointerv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glGetVertexAttribfv()`

See: [glGetVertexAttribfv\(\)](#) on Kronos website

`kivy.graphics.opengl.glGetVertexAttribiv()`

See: [glGetVertexAttribiv\(\)](#) on Kronos website

`kivy.graphics.opengl.glHint()`

See: [glHint\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsBuffer()`

See: [glIsBuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsEnabled()`

See: [glIsEnabled\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsFramebuffer()`

See: [glIsFramebuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsProgram()`

See: [glIsProgram\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsRenderbuffer()`

See: [glIsRenderbuffer\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsShader()`

See: [glIsShader\(\)](#) on Kronos website

`kivy.graphics.opengl.glIsTexture()`

See: [glIsTexture\(\)](#) on Kronos website

`kivy.graphics.opengl.glLineWidth()`

See: [glLineWidth\(\)](#) on Kronos website

`kivy.graphics.opengl.glLinkProgram()`

See: [glLinkProgram\(\)](#) on Kronos website

`kivy.graphics.opengl.glPixelStorei()`

See: [glPixelStorei\(\)](#) on Kronos website

`kivy.graphics.opengl.glPolygonOffset()`

See: [glPolygonOffset\(\)](#) on Kronos website

`kivy.graphics.opengl.glReadPixels()`

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()`

See: [glRenderbufferStorage\(\)](#) on Kronos website

`kivy.graphics.opengl.glSampleCoverage()`

See: [glSampleCoverage\(\)](#) on Kronos website

`kivy.graphics.opengl.glScissor()`

See: [glScissor\(\)](#) on Kronos website

`kivy.graphics.opengl.glShaderBinary()`

See: [glShaderBinary\(\)](#) on Kronos website

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glShaderSource()`

See: [glShaderSource\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilFunc()`

See: [glStencilFunc\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilFuncSeparate()`

See: [glStencilFuncSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilMask()`

See: [glStencilMask\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilMaskSeparate()`

See: [glStencilMaskSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilOp()`

See: [glStencilOp\(\)](#) on Kronos website

`kivy.graphics.opengl.glStencilOpSeparate()`

See: [glStencilOpSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glTexImage2D()`

See: [glTexImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glTexParameterf()`

See: [glTexParameterf\(\)](#) on Kronos website

`kivy.graphics.opengl.glTexParameterfv()`

See: [glTexParameterfv\(\)](#) on Kronos website

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glTexParameteri()`

See: [glTexParameteri\(\)](#) on Kronos website

`kivy.graphics.opengl.glTexParameteriv()`

See: [glTexParameteriv\(\)](#) on Kronos website

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glTexSubImage2D()`

See: [glTexSubImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glUniform1f()`

See: [glUniform1f\(\)](#) on Kronos website

`kivy.graphics.opengl.glUniform1fv()`

See: [glUniform1fv\(\)](#) on Kronos website

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform1i()`

See: [glUniform1i\(\)](#) on Kronos website



`kivy.graphics.opengl.glUniform1iv()`

See: [glUniform1iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform2f()`

See: [glUniform2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2fv()`

See: [glUniform2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform2i()`

See: [glUniform2i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2iv()`

See: [glUniform2iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform3f()`

See: [glUniform3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3fv()`

See: [glUniform3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform3i()`

See: [glUniform3i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3iv()`

See: [glUniform3iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4f()`

See: [glUniform4f\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4fv()`

See: [glUniform4fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4i()`

See: [glUniform4i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform4iv()`

See: [glUniform4iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix2fv()`

See: [glUniformMatrix2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix3fv()`

See: [glUniformMatrix3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix4fv()`

See: [glUniformMatrix4fv\(\) on Kronos website](#)

`kivy.graphics.opengl.glUseProgram()`

See: [glUseProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glValidateProgram()`

See: [glValidateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1f()`

See: [glVertexAttrib1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1fv()`

See: [glVertexAttrib1fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib2f()`

See: [glVertexAttrib2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib2fv()`

See: [glVertexAttrib2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib3f()`

See: [glVertexAttrib3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib3fv()`

See: [glVertexAttrib3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib4f()`

See: [glVertexAttrib4f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib4fv()`

See: [glVertexAttrib4fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttribPointer()`

See: [glVertexAttribPointer\(\) on Kronos website](#)

`kivy.graphics.opengl.glViewport()`

See: [glViewport\(\) on Kronos website](#)

## 61.10 OpenGL utilities

New in version 1.0.7.

`kivy.graphics.opengl_utils.gl_get_extensions()`

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()`

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()`

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()`

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()`

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()`

Return 1 if the texture can be converted to a native format.

`kivy.graphics.opengl_utils.gl_has_texture_native_format()`

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()`

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()`

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()`

Return the minor component of the OpenGL version.

New in version 1.2.0.

`kivy.graphics.opengl_utils.gl_get_version_major()`

Return the major component of the OpenGL version.

New in version 1.2.0.

## 61.11 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.

---

### 61.11.1 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;
```

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;

```

### 61.11.2 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
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`

Bases: `object`

Create a vertex or fragment shader.

**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.

## 61.12 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: drawing instructions will now be drawn without clipping but the  
# mask will still be on the stack. You can return to PHASE 2 at any  
# time by issuing another *StencilUse* command.
```

### StencilPop

```
# PHASE 4: the stencil is now removed from the stack and unloaded.
```

### 61.12.1 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)

### 61.12.2 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:
    # new in kivy 1.3.0, remove the mask previously added
    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`

Bases: `kivy.graphics.instructions.Instruction`

Use current stencil buffer as a mask. Check the module documentation for 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.

## 61.13 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.VertexIntruction.texture` parameter. If you want to use your texture in kv lang, you can save it in an `ObjectProperty` inside your widget.

### 61.13.1 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 rgb / 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)]

# 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 1.8.1, 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')

```

### 61.13.2 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.

### 61.13.3 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.

### 61.13.4 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(0, 64, 64, 64)
topright = texture.get_region(64, 64, 64, 64)
```

### 61.13.5 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.

---

### 61.13.6 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 = '
```

**class** `kivy.graphics.texture.Texture`

Bases: `object`

Handle an OpenGL texture. This class can be used to create simple textures or complex textures based on `ImageData`.

**add\_reload\_observer()**

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()

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()

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 False**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 1.8.1: `pbuffer` can now be any class instance that implements the python buffer interface and / or memoryviews thereof.

### blit\_data()

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.

### create()

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, default 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

### **create\_from\_data()**

Create a texture from an `ImageData` class.

### **flip\_horizontal()**

Flip `tex_coords` for horizontal display.

New in version 1.8.1.

### **flip\_vertical()**

Flip `tex_coords` for vertical display.

### **get\_region()**

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()**

Remove a callback from the observer list, previously added by `add_reload_observer()`.

New in version 1.2.0.

### **save()**

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, default 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.

### **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**

Bases: [kivy.graphics.texture.Texture](#)

Handle a region of a Texture class. Useful for non power-of-2 texture handling.

## 61.14 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: object

Optimized matrix class for OpenGL:

```
>>> 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]
```

**identity()**

Reset the matrix to the identity matrix (inplace).

**inverse()**

Return the inverse of the matrix as a new Matrix.

**look\_at()**

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()**

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()**

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()**

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()**

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()**

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()**

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

### **translate()**

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()**

Return the transposed matrix as a new Matrix.

New in version 1.6.0.

### **view\_clip()**

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.

## 61.15 Vertex Instructions

This module includes all the classes for drawing simple vertex objects.

**Note:** 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 when the button is pressed the triangle coordinates will be changed, the graphics will not be updated because the list itself has not been 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`

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d triangle.

**Parameters**

*points*: listList 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`

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d quad.

**Parameters**

*points*: listList 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`

Bases: `kivy.graphics.instructions.VertexInstruction`



A 2d rectangle.

**Parameters**

*pos*: listPosition of the rectangle, in the format (x, y).

*size*: listSize 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.BorderImage`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

A 2d border image. The behavior of the border image is similar to the concept of a CSS3 border-image.

**Parameters**

*border*: listBorder information in the format (top, right, bottom, left). Each value is in pixels.

**border**

Property for getting/setting the border of the class.

class `kivy.graphics.vertex_instructions.Ellipse`

Bases: `kivy.graphics.vertex_instructions.Rectangle`

A 2D ellipse.

Changed in version 1.0.7: Added `angle_start` and `angle_end`.

**Parameters**

*segments*: int, defaults to 180Define how many segments are needed for drawing the ellipse. The drawing will be smoother if you have many segments.

*angle\_start*: int, defaults to 0Specifies the starting angle, in degrees, of the disk portion.

*angle\_end*: int, defaults to 360Specifies 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`

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d line.

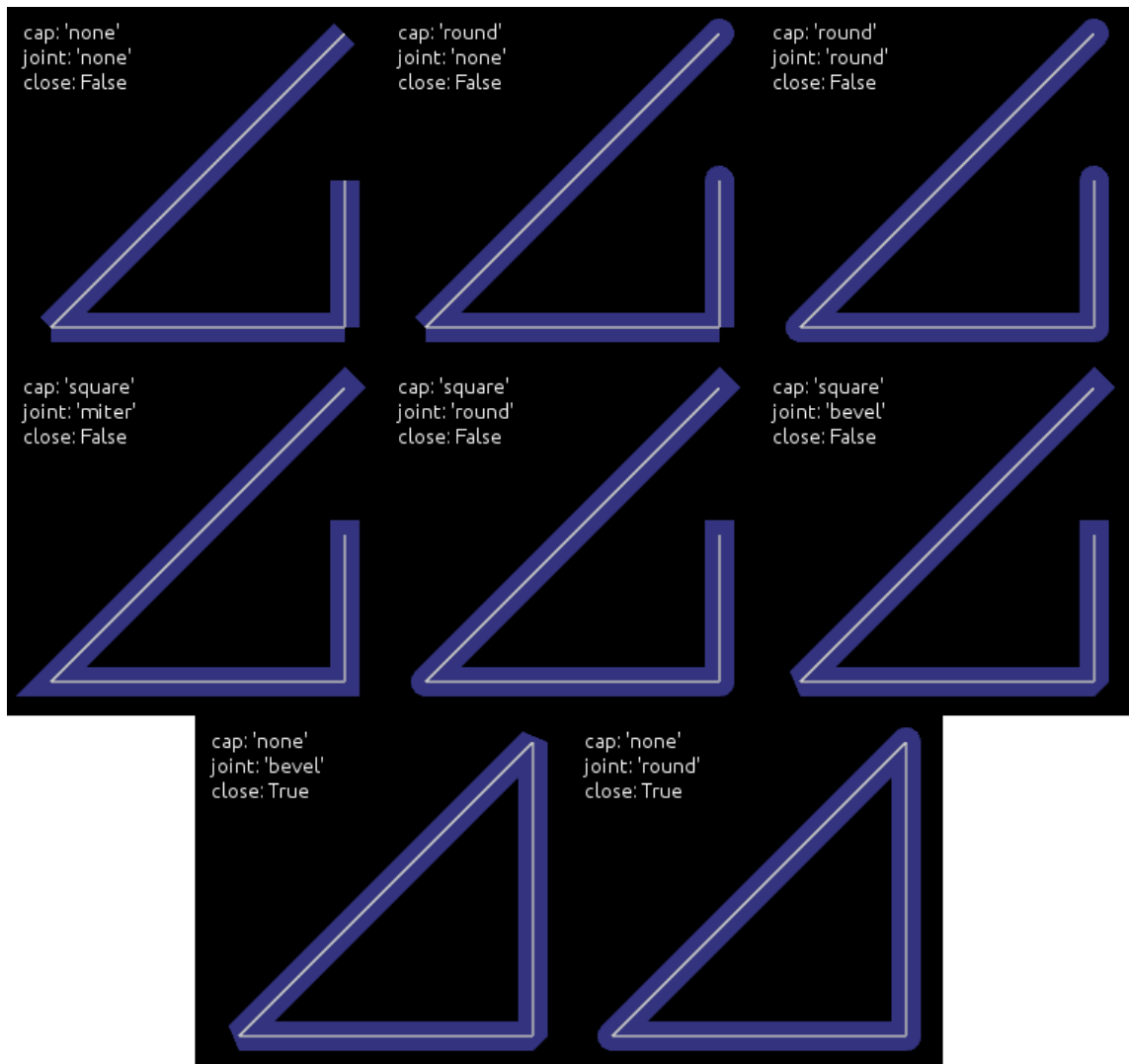
Drawing a line can be done easily:

```
with self.canvas:
```

```
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

Actually, the line have 3 internal drawing mode that you should know about if you want to get the best performance of it:

- 1.If the `width` is 1.0, then we will use standard `GL_LINE` drawing from OpenGL. `dash_length` and `dash_offset` works, while properties for cap and joint have no sense for this.
- 2.If the `width` is > 1.0, then we will use a custom drawing method, based on triangles. `dash_length` and `dash_offset` is not working on that mode. Additionally, if the current color have an alpha < 1.0, 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), default 1

**dash\_offset:** int Offset between the end of a segments and the beginning of the next one, default 0, changing this makes it dashed.

**width:** float Width of the line, default 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

**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. Check [circle](#) for more information.

**ellipse:** list If set, the [points](#) will be set to build an ellipse. Check [ellipse](#) for more information.

**rectangle:** list If set, the [points](#) will be set to build a rectangle. Check [rectangle](#) for more information.

**bezier:** list If set, the [points](#) will be set to build a bezier line. Check [bezier](#) for more information.

**bezier\_precision:** int, defaults to 180 Precision of the Bezier drawing.

New in version 1.0.8: `dash_offset` and `dash_length` have been added

New in version 1.4.1: `width`, `cap`, `joint`, `cap_precision`, `joint_precision`, `close`, `ellipse`, `rectangle` have been added.

New in version 1.4.1: *bezier*, *bezier\_precision* 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 calculate 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.

### **ellipse**

Use this property to build an ellipse, without calculate 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) angle\_end, segments):

- 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 (default to 30)

The line is automatically closed.

Usage:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40, 100))
```

New in version 1.8.1.

### **width**

Determine the width of the line, defaults to 1.0.

New in version 1.4.1.

## **class kivy.graphics.vertex\_instructions.Point**

Bases: **kivy.graphics.instructions.VertexInstruction**

A 2d line.

### **Parameters**

**points:** list List of points in the format (x1, y1, x2, y2...).

**pointsize:** float, defaults to 1. Size of the point (1. means the real size will be 2).

**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()**

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 points of the triangle.

### **pointsize**

Property for getting/setting point size.

**class** `kivy.graphics.vertex_instructions.Mesh`

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d mesh.

The format for vertices is currently fixed but this might change in a future release. Right now, each vertex is described with 2D coordinates (x, y) and a 2D texture coordinate (u, v).

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:** list List of vertices in the format (x1, y1, u1, v1, x2, y2, u2, v2...).

**indices:** list List of indices in the format (i1, i2, i3...).

**mode:** str Mode of the vbo. Check `mode` for more information. Defaults to 'points'.

#### 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.vertex_instructions.GraphicException`

Bases: `exceptions.Exception`

Exception raised when a graphics error is fired.

**class** `kivy.graphics.vertex_instructions.Bezier`

Bases: `kivy.graphics.instructions.VertexInstruction`

A 2d Bezier curve.

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.vertex_instructions.SmoothLine`

Bases: `kivy.graphics.vertex_instructions.Line`

Experimental line using over-draw method to get better antialiasing results. It has few drawbacks:

- drawing line with alpha will unlikely doesn't give the intended result if the line cross itself
- no cap or joint are supported
- it use a custom texture with premultiplied alpha
- dash is not supported
- line under 1px width are not supported, it will look the same

**Warning:** This is an unfinished work, experimental, subject to crash.

New in version 1.8.1.

### **overdraw\_width**

Determine the overdraw width of the line, defaults to 1.2





# GRAPHICS COMPILER

Before rendering an `InstructionGroup`, we compile the group in order to reduce the number of instructions executed at rendering time.

## 62.1 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 `GI_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.

# 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: `object`

The Context class manages groups of graphics instructions. It can also be used to manage observer callbacks. See [add\\_reload\\_observer\(\)](#) and [remove\\_reload\\_observer\(\)](#) for more information.

**add\_reload\_observer()**

(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.

**remove\_reload\_observer()**

(internal) Remove a callback from the observer list previously added by [add\\_reload\\_observer\(\)](#).



# 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`

Bases: `kivy.graphics.instructions.ContextInstruction`

Instruction to set the color state for any vertices being drawn after it. All the values passed are between 0 and 1, not 0 and 255.

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')
```

In kv lang:

```
<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`

Bases: `kivy.graphics.instructions.ContextInstruction`

`BindTexture` Graphic instruction. 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`

Bases: `kivy.graphics.instructions.ContextInstruction`

Push the matrix onto the context's matrix stack.

**stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

**class** `kivy.graphics.context_instructions.PopMatrix`

Bases: `kivy.graphics.instructions.ContextInstruction`

Pop the matrix from the context's matrix stack onto the model view.

**stack**

Name of the matrix stack to use. Can be 'modelview\_mat' or 'projection\_mat'.

New in version 1.6.0.

**class** `kivy.graphics.context_instructions.Rotate`

Bases: `kivy.graphics.context_instructions.Transform`

Rotate the coordinate space by applying a rotation transformation 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()**

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`

Bases: `kivy.graphics.context_instructions.Transform`

Instruction to create a non uniform scale transformation.

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.

**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`

Bases: `kivy.graphics.context_instructions.Transform`

Instruction to create a translation of the model view coordinate space.

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`

Bases: `kivy.graphics.instructions.ContextInstruction`

Base class for Matrix Instruction on the canvas.

**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' or 'projection\_mat'.

New in version 1.6.0.



# 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

        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.

## 65.1 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
```

Bases: `kivy.graphics.instructions.RenderContext`

Fbo class for wrapping the OpenGL Framebuffer extension. The Fbo support "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 1.8.1.

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 1.8.1

---

#### `add_reload_observer()`

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()**

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()**

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



# GL INSTRUCTIONS

New in version 1.3.0.

## 66.1 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`  
Bases: `kivy.graphics.instructions.Instruction`

ClearColor Graphics 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`  
Bases: `kivy.graphics.instructions.Instruction`

Clearbuffer Graphics Instruction.

New in version 1.3.0.

Clear the buffers specified by the instructions buffer mask property. By default, only the color 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.

# 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`

Bases: `kivy.event.ObjectWithUid`

Represents the smallest instruction available. This class is for internal usage only, don't use it directly.

**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`

Bases: `kivy.graphics.instructions.Instruction`

Group of **Instructions**. Allows for the adding and 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()**

Add a new **Instruction** to our list.

**clear()**

Remove all the **Instructions**.

**get\_group()**

Return an iterable for all the **Instructions** with a specific group name.

**insert()**

Insert a new **Instruction** into our list at index.

**remove()**

Remove an existing **Instruction** from our list.

### **remove\_group()**

Remove all **Instructions** with a specific group name.

### **class kivy.graphics.instructions.ContextInstruction**

Bases: **kivy.graphics.instructions.Instruction**

The ContextInstruction class is the base for the creation of instructions 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**

Bases: **kivy.graphics.instructions.Instruction**

The VertexInstruction class is the base for all graphics instructions 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, y + h, u, y + 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`

Bases: `kivy.graphics.instructions.CanvasBase`

The important Canvas class. Use this class to add graphics or context 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))
```

### **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);
```

**class** kivy.graphics.instructions.**CanvasBase**

Bases: [kivy.graphics.instructions.InstructionGroup](#)

CanvasBase provides the context manager methods for the [Canvas](#).

**class** kivy.graphics.instructions.**RenderContext**

Bases: [kivy.graphics.instructions.Canvas](#)

The render context stores all the necessary information for drawing, i.e.:

- 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\_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**

Bases: [kivy.graphics.instructions.Instruction](#)

New in version 1.0.4.

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.

#### **reset\_context**

Set this to True if you want to reset the OpenGL context for Kivy after the callback has been called.



# 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()`  
See: [glActiveTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glAttachShader()`  
See: [glAttachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindAttribLocation()`  
See: [glBindAttribLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindBuffer()`  
See: [glBindBuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindFramebuffer()`  
See: [glBindFramebuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindRenderbuffer()`  
See: [glBindRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindTexture()`  
See: [glBindTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendColor()`  
See: [glBlendColor\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendEquation()`  
See: [glBlendEquation\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendEquationSeparate()`  
See: [glBlendEquationSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendFunc()`  
See: [glBlendFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glBlendFuncSeparate()`  
See: [glBlendFuncSeparate\(\) on Kronos website](#)

`kivy.graphics.opengl.glBufferData()`  
See: [glBufferData\(\) on Kronos website](#)

`kivy.graphics.opengl.glBufferSubData()`  
See: [glBufferSubData\(\) on Kronos website](#)

`kivy.graphics.opengl.glCheckFramebufferStatus()`  
See: [glCheckFramebufferStatus\(\) on Kronos website](#)

`kivy.graphics.opengl.glClear()`  
See: [glClear\(\) on Kronos website](#)

`kivy.graphics.opengl.glClearColor()`  
See: [glClearColor\(\) on Kronos website](#)

`kivy.graphics.opengl.glClearStencil()`  
See: [glClearStencil\(\) on Kronos website](#)

`kivy.graphics.opengl.glColorMask()`  
See: [glColorMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompileShader()`  
See: [glCompileShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompressedTexImage2D()`  
See: [glCompressedTexImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCompressedTexSubImage2D()`  
See: [glCompressedTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCopyTexImage2D()`  
See: [glCopyTexImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCopyTexSubImage2D()`  
See: [glCopyTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glCreateProgram()`  
See: [glCreateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glCreateShader()`  
See: [glCreateShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glCullFace()`  
See: [glCullFace\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteBuffers()`  
See: [glDeleteBuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteFramebuffers()`  
See: [glDeleteFramebuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteProgram()`  
See: [glDeleteProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteRenderbuffers()`  
See: [glDeleteRenderbuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteShader()`  
See: [glDeleteShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteTextures()`  
See: [glDeleteTextures\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthFunc()`  
See: [glDepthFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthMask()`  
See: [glDepthMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glDetachShader()`  
See: [glDetachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisable()`  
See: [glDisable\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisableVertexAttribArray()`  
See: [glDisableVertexAttribArray\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawArrays()`  
See: [glDrawArrays\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawElements()`  
See: [glDrawElements\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnable()`  
See: [glEnable\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnableVertexAttribArray()`  
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()`  
See: [glFramebufferRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glFramebufferTexture2D()`  
See: [glFramebufferTexture2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glFrontFace()`  
See: [glFrontFace\(\) on Kronos website](#)

`kivy.graphics.opengl.glGenBuffers()`  
See: [glGenBuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenFramebuffers()`  
See: [glGenFramebuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenRenderbuffers()`  
See: [glGenRenderbuffers\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenTextures()`  
See: [glGenTextures\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGenerateMipmap()`  
See: [glGenerateMipmap\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetActiveAttrib()`  
See: [glGetActiveAttrib\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetActiveUniform()`  
See: [glGetActiveUniform\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttachedShaders()`

See: [glGetAttachedShaders\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetAttribLocation()`

See: [glGetAttribLocation\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBooleanv()`

See: [glGetBooleanv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetBufferParameteriv()`

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()`

See: [glGetFloatv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetFramebufferAttachmentParameteriv()`

See: [glGetFramebufferAttachmentParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetIntegerv()`

See: [glGetIntegerv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetProgramInfoLog()`

See: [glGetProgramInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetProgramiv()`

See: [glGetProgramiv\(\) on Kronos website](#)

Unlike the C specification, the value(s) will be the result of the call

`kivy.graphics.opengl.glGetRenderbufferParameteriv()`

See: [glGetRenderbufferParameteriv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetShaderInfoLog()`

See: [glGetShaderInfoLog\(\) on Kronos website](#)

Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderPrecisionFormat()`

See: [glGetShaderPrecisionFormat\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glGetShaderSource()`

See: [glGetShaderSource\(\) on Kronos website](#)



Unlike the C specification, the source code will be returned as a string.

`kivy.graphics.opengl.glGetShaderiv()`

See: [glGetShaderiv\(\) on Kronos website](#)

Unlike the C specification, the value will be the result of call.

`kivy.graphics.opengl.glGetString()`

See: [glGetString\(\) on Kronos website](#)

Unlike the C specification, the value will be returned as a string.

`kivy.graphics.opengl.glGetTexParameterfv()`

See: [glGetTexParameterfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetTexParameteriv()`

See: [glGetTexParameteriv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformLocation()`

See: [glGetUniformLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformfv()`

See: [glGetUniformfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetUniformiv()`

See: [glGetUniformiv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetVertexAttribPointerv()`

See: [glGetVertexAttribPointerv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glGetVertexAttribfv()`

See: [glGetVertexAttribfv\(\) on Kronos website](#)

`kivy.graphics.opengl.glGetVertexAttribiv()`

See: [glGetVertexAttribiv\(\) on Kronos website](#)

`kivy.graphics.opengl.glHint()`

See: [glHint\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsBuffer()`

See: [glIsBuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsEnabled()`

See: [glIsEnabled\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsFramebuffer()`

See: [glIsFramebuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsProgram()`

See: [glIsProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsRenderbuffer()`

See: [glIsRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsShader()`

See: [glIsShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glIsTexture()`

See: [glIsTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glLineWidth()`

See: [glLineWidth\(\) on Kronos website](#)

`kivy.graphics.opengl.glLinkProgram()`

See: [glLinkProgram\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glPixelStorei()`

See: [glPixelStorei\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glPolygonOffset()`

See: [glPolygonOffset\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glReadPixels()`

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()`

See: [glRenderbufferStorage\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glSampleCoverage()`

See: [glSampleCoverage\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glScissor()`

See: [glScissor\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glShaderBinary()`

See: [glShaderBinary\(\)](#) on [Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glShaderSource()`

See: [glShaderSource\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilFunc()`

See: [glStencilFunc\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilFuncSeparate()`

See: [glStencilFuncSeparate\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilMask()`

See: [glStencilMask\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilMaskSeparate()`

See: [glStencilMaskSeparate\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilOp()`

See: [glStencilOp\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glStencilOpSeparate()`

See: [glStencilOpSeparate\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glTexImage2D()`

See: [glTexImage2D\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glTexParameterf()`

See: [glTexParameterf\(\)](#) on [Kronos website](#)

`kivy.graphics.opengl.glTexParameterfv()`

See: [glTexParameterfv\(\)](#) on [Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glTexParameterI()`

See: [glTexParameterI\(\) on Kronos website](#)

`kivy.graphics.opengl.glTexParameteriv()`

See: [glTexParameteriv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glTexSubImage2D()`

See: [glTexSubImage2D\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1f()`

See: [glUniform1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1fv()`

See: [glUniform1fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform1i()`

See: [glUniform1i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1iv()`

See: [glUniform1iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform2f()`

See: [glUniform2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2fv()`

See: [glUniform2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform2i()`

See: [glUniform2i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2iv()`

See: [glUniform2iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform3f()`

See: [glUniform3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3fv()`

See: [glUniform3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform3i()`

See: [glUniform3i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3iv()`

See: [glUniform3iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4f()`

See: [glUniform4f\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4fv()`

See: [glUniform4fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniform4i()`

See: [glUniform4i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform4iv()`

See: [glUniform4iv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix2fv()`

See: [glUniformMatrix2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix3fv()`

See: [glUniformMatrix3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glUniformMatrix4fv()`

See: [glUniformMatrix4fv\(\) on Kronos website](#)

`kivy.graphics.opengl.glUseProgram()`

See: [glUseProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glValidateProgram()`

See: [glValidateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1f()`

See: [glVertexAttrib1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1fv()`

See: [glVertexAttrib1fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib2f()`

See: [glVertexAttrib2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib2fv()`

See: [glVertexAttrib2fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib3f()`

See: [glVertexAttrib3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib3fv()`

See: [glVertexAttrib3fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttrib4f()`

See: [glVertexAttrib4f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib4fv()`

See: [glVertexAttrib4fv\(\) on Kronos website](#)

**Warning:** Not implemented yet.

`kivy.graphics.opengl.glVertexAttribPointer()`

See: [glVertexAttribPointer\(\) on Kronos website](#)

`kivy.graphics.opengl.glViewport()`

See: [glViewport\(\) on Kronos website](#)



## OPENGL UTILITIES

New in version 1.0.7.

`kivy.graphics.opengl_utils.gl_get_extensions()`

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()`

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()`

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()`

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()`

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()`

Return 1 if the texture can be converted to a native format.

`kivy.graphics.opengl_utils.gl_has_texture_native_format()`

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()`

Return a list of texture formats recognized by kivy. The texture list is informative but might not be supported by your hardware. If you want a list of supported textures, you must filter that list as follows:

```
supported_fmfs = [gl_has_texture_format(x) for x in gl_get_texture_formats()]
```

`kivy.graphics.opengl_utils.gl_get_version()`

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()`

Return the minor component of the OpenGL version.

New in version 1.2.0.

`kivy.graphics.opengl_utils.gl_get_version_major()`

Return the major component of the OpenGL version.

New in version 1.2.0.



# 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.

---

## 70.1 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;
```

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;
```

## 70.2 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
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`

Bases: `object`

Create a vertex or fragment shader.

**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.

# 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: drawing instructions wil now be drawn without clipping but the  
# mask will still be on the stack. You can return to PHASE 2 at any  
# time by issuing another *StencilUse* command.
```

## StencilPop

```
# PHASE 4: the stencil is now removed from the stack and unloaded.
```

## 71.1 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)

## 71.2 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:
    # new in kivy 1.3.0, remove the mask previously added
    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`

Bases: `kivy.graphics.instructions.Instruction`

Use current stencil buffer as a mask. Check the module documentation for 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.

# 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.VertexIntruction.texture` parameter. If you want to use your texture in kv lang, you can save it in an **ObjectProperty** inside your widget.

## 72.1 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 rgb / 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)]

# 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 1.8.1, 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')

```

## 72.2 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.

## 72.3 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.

## 72.4 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(0, 64, 64, 64)
topright = texture.get_region(64, 64, 64, 64)
```

## 72.5 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.

---

## 72.6 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 = '
```

## class kivy.graphics.texture.Texture

Bases: object

Handle an OpenGL texture. This class can be used to create simple textures or complex textures based on ImageData.

### add\_reload\_observer()

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()

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()

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 False** 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 1.8.1: `pbuffer` can now be any class instance that implements the python buffer interface and / or memoryviews thereof.

### blit\_data()

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.

### **create()**

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, default 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

### **create\_from\_data()**

Create a texture from an ImageData class.

### **flip\_horizontal()**

Flip tex\_coords for horizontal display.

New in version 1.8.1.

### **flip\_vertical()**

Flip tex\_coords for vertical display.

### **get\_region()**

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()**

Remove a callback from the observer list, previously added by [add\\_reload\\_observer\(\)](#).

New in version 1.2.0.

### **save()**

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, default 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.

### **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**

Bases: [kivy.graphics.texture.Texture](#)

Handle a region of a Texture class. Useful for non power-of-2 texture handling.

---

# 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: `object`

Optimized matrix class for OpenGL:

```
>>> 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]
```

**identity()**

Reset the matrix to the identity matrix (inplace).

**inverse()**

Return the inverse of the matrix as a new `Matrix`.

**look\_at()**

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()**

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()**

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()**

Creates a perspective matrix (inplace).

#### **Parameters**

**fovy: float**"Field Of View" angle  
**aspect: float**Aspect ratio  
**zNear: float**Near clipping plane  
**zFar: float**Far clipping plane

New in version 1.6.0.

### **project()**

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()**

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()**

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

### **translate()**

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()**

Return the transposed matrix as a new Matrix.

New in version 1.6.0.

#### **view\_clip()**

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.



# 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)
- MacOSX 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 to represent a touch and non-touch object.

**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

**distance**(*other\_touch*)

Return the distance between the current touch and another touch.

**dpos**

Return delta between last position and current position, in the screen coordinate system (`self.dx, self.dy`)

**grab**(*class\_instance*, *exclusive=False*)

Grab this motion event. You can grab a touch if you absolutely want to receive `on_touch_move()` and `on_touch_up()`, even if the touch is not dispatched by your 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
```

**is\_mouse\_scrolling**

Returns True if the touch is a mousewheel scrolling

New in version 1.6.0.

**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`)

**pop**()

Pop attributes values from the stack

**ppos**

Return the previous position of the touch in the screen coordinate system (`self.px`, `self.py`)

**push**(*attrs=None*)

Push attribute values in *attrs* onto the stack

**scale\_for\_screen**(*w*, *h*, *p=None*, *rotation=0*, *smode='None'*, *kheight=0*)

Scale position for the screen

**spos**

Return the position in the 0-1 coordinate system (`self.sx`, `self.sy`)

**ungrab**(*class\_instance*)

Ungrab a previously grabbed touch

**class** `kivy.input.MotionEventProvider`(*device*, *args*)

Bases: `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

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

## 74.1 Input Postprocessing

### 74.1.1 Calibration

New in version 1.8.1.

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 go 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.

## class kivy.input.postproc.calibration.InputPostprocCalibration

Bases: object

Recalibrate the inputs.

The configuration must go within a section named *postproc:calibration*. Within the section, you must have line like:

```
devicename = 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

## 74.1.2 Dejitte

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.dejitte.InputPostprocDejitte**

Bases: 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 dejitte (because they're very precise already).

## 74.1.3 Double Tap

Search touch for a double tap

**class kivy.input.postproc.doubletap.InputPostprocDoubleTap**

Bases: 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.

## 74.1.4 Ignore list

Ignore touch on some areas of the screen

**class kivy.input.postproc.ignorelist.InputPostprocIgnoreList**

Bases: 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.

## 74.1.5 Retain Touch

Reuse touch to counter lost finger behavior

`class kivy.input.postproc.retaintouch.InputPostprocRetainTouch`

Bases: `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.

## 74.1.6 Triple Tap

New in version 1.7.0.

Search touch for a triple tap

`class kivy.input.postproc.tripletap.InputPostprocTripleTap`

Bases: `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 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.

## 74.2 Providers

### 74.2.1 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>

### 74.2.2 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.

### 74.2.3 Common definitions for a Windows provider

This file provides common definitions for constants used by `WM_Touch` / `WM_Pen`.

### 74.2.4 Leap Motion - finger only

### 74.2.5 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 there are any touches activated 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 1.8.1.

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), `multitouch_sim` will be added to touch's profile, and property `multitouch_sim` to the touch. By default `multitouch_sim` is True and multitouch will be emulated for that touch. However, if `multitouch_on_demand` is added to the config:

```
[input]
mouse = mouse,multitouch_on_demand
```

then `multitouch_sim` defaults to `False`. In that case, if before mouse release (e.g. in `on_touch_down/move`) `multitouch_sim` is set to True, the touch will simulate multi-touch. For example:

```
if 'multitouch_sim' in touch.profile:
    touch.multitouch_sim = True
```

Following is a list of the supported profiles for `MouseEvent`.

Profile name	Description
button	Mouse button (left, right, middle, scrollup, scrolldown) Use property <code>button</code>
pos	2D position. Use properties <code>x</code> , <code>y</code> or <code>pos</code>
multitouch_sim	If multitouch is simulated. Use property <code>multitouch_sim</code> . See documataion above.

## 74.2.6 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

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
```

### 74.2.7 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

### 74.2.8 Native support of MultitouchSupport framework for MacBook (MaxOSX platform)

### 74.2.9 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

#### 74.2.10 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.

#### 74.2.11 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.

#### 74.2.12 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.

**class** `kivy.input.providers.tuio.Tuio2dObjMotionEvent` (*device, id, args*)

Bases: `kivy.input.providers.tuio.TuioMotionEvent`

A 2dObj TUIO object.

## 74.3 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.

### 74.3.1 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.

### 74.3.2 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
```

### 74.3.3 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:ang
```

### 74.3.4 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.

#### 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.

## 74.4 Motion Event

The `MotionEvent` is the base class used for every touch and non-touch event. 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`.

---

### 74.4.1 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()`.

### 74.4.2 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`.

### 74.4.3 Profiles

A capability is the ability of a `MotionEvent` to store new information or a way to indicate what is supported by the `MotionEvent`. 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 check what is currently supported by the `MotionEvent` and how to access it.

This is a tiny list of the supported profiles by default. Check other input providers to see if there are other profiles available.

Profile name	Description
angle	2D angle. Use property <i>a</i>
button	Mouse button (left, right, middle, scrollup, scrolldown) Use property <i>button</i>
markerid	Marker or Fiducial ID. Use property <i>fid</i>
pos	2D position. Use properties <i>x, y</i> or <i>pos'</i>
pos3d	3D position. Use properties <i>x, y, z</i>
pressure	Pressure of the contact. Use property <i>pressure</i>
shape	Contact shape. Use property <i>shape</i>

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 to represent a touch and non-touch object.

#### 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 absolutely want to receive on\_touch\_move() and on\_touch\_up(), even if the touch is not dispatched by your 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

## 74.5 Motion Event Factory

Factory of `MotionEvent` providers.

**class** `kivy.input.factory.MotionEventFactory`

`MotionEventFactory` 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

## 74.6 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: `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.

## 74.7 Motion Event Shape

Represent the shape of the `MotionEvent`

**class** `kivy.input.shape.Shape`

Bases: `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 fo the rect



# MOTION EVENT FACTORY

Factory of **MotionEvent** providers.

**class** `kivy.input.factory.MotionEventFactory`

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



# MOTION EVENT

The `MotionEvent` is the base class used for every touch and non-touch event. 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`.

---

## 76.1 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()`.

## 76.2 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`.

## 76.3 Profiles

A capability is the ability of a `MotionEvent` to store new information or a way to indicate what is supported by the `MotionEvent`. 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 check what is currently supported by the `MotionEvent` and how to access it.

This is a tiny list of the supported profiles by default. Check other input providers to see if there are other profiles available.

Profile name	Description
angle	2D angle. Use property <i>a</i>
button	Mouse button (left, right, middle, scrollup, scrolldown) Use property <i>button</i>
markerid	Marker or Fiducial ID. Use property <i>fid</i>
pos	2D position. Use properties <i>x, y</i> or <i>pos'</i>
pos3d	3D position. Use properties <i>x, y, z</i>
pressure	Pressure of the contact. Use property <i>pressure</i>
shape	Contact shape. Use property <i>shape</i>

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 to represent a touch and non-touch object.

#### 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 absolutely want to receive on\_touch\_move() and on\_touch\_up(), even if the touch is not dispatched by your 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





# INPUT POSTPROCESSING

## 77.1 Calibration

New in version 1.8.1.

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.

**class kivy.input.postproc.calibration.InputPostprocCalibration**

Bases: object

Recalibrate the inputs.

The configuration must go within a section named *postproc:calibration*. Within the section, you must have line like:

```
devicename = param=value,param=value
```

### Parameters

**xratio:** floatValue to multiply X

**yratio:** floatValue to multiply Y

**xoffset:** floatValue to add to X

**yoffset:** floatValue to add to Y

## 77.2 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: 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).

## 77.3 Double Tap

Search touch for a double tap

```
class kivy.input.postproc.doubletap.InputPostprocDoubleTap  
    Bases: 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.

## 77.4 Ignore list

Ignore touch on some areas of the screen

```
class kivy.input.postproc.ignorelist.InputPostprocIgnoreList  
    Bases: 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.

## 77.5 Retain Touch

Reuse touch to counter lost finger behavior

```
class kivy.input.postproc.retaintouch.InputPostprocRetainTouch
```

Bases: 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.

## 77.6 Triple Tap

New in version 1.7.0.

Search touch for a triple tap

```
class kivy.input.postproc.tripletap.InputPostprocTripleTap
```

Bases: 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.



# CALIBRATION

New in version 1.8.1.

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.

**class kivy.input.postproc.calibration.InputPostprocCalibration**

Bases: object

Recalibrate the inputs.

The configuration must go within a section named *postproc:calibration*. Within the section, you must have line like:

```
devicename = 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



# 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: `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).





## DOUBLE TAP

Search touch for a double tap

```
class kivy.input.postproc.doubletap.InputPostprocDoubleTap
```

```
    Bases: 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.



## IGNORE LIST

Ignore touch on some areas of the screen

```
class kivy.input.postproc.ignorelist.InputPostprocIgnoreList
```

```
    Bases: 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.



## RETAIN TOUCH

Reuse touch to counter lost finger behavior

`class kivy.input.postproc.retain_touch.InputPostprocRetainTouch`

Bases: `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.



## TRIPLE TAP

New in version 1.7.0.

Search touch for a triple tap

`class kivy.input.postproc.tripletap.InputPostprocTripleTap`  
Bases: `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 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.





## 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: `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.



# PROVIDERS

## 85.1 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>

## 85.2 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.

## 85.3 Common definitions for a Windows provider

This file provides common definitions for constants used by WM\_Touch / WM\_Pen.

## 85.4 Leap Motion - finger only

## 85.5 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 there are any touches activated by another provider, the mouse event will be discarded. Add this to your configuration:

```
[input]
mouse = mouse,disable_on_activity
```

### 85.5.1 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 1.8.1.

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), `multitouch_sim` will be added to touch’s profile, and property `multitouch_sim` to the touch. By default `multitouch_sim` is True and multitouch will be emulated for that touch. However, if `multitouch_on_demand` is added to the config:

```
[input]
mouse = mouse,multitouch_on_demand
```

then `multitouch_sim` defaults to `False`. In that case, if before mouse release (e.g. in `on_touch_down/move`) `multitouch_sim` is set to True, the touch will simulate multi-touch. For example:

```
if 'multitouch_sim' in touch.profile:
    touch.multitouch_sim = True
```

Following is a list of the supported profiles for `MouseEvent`.

Profile name	Description
button	Mouse button (left, right, middle, scrollup, scrolldown) Use property <code>button</code>
pos	2D position. Use properties <code>x</code> , <code>y</code> or <code>pos</code>
multitouch_sim	If multitouch is simulated. Use property <code>multitouch_sim</code> . See documation above.

## 85.6 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

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
```

## 85.7 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

## 85.8 Native support of MultitouchSupport framework for MacBook (MaxOSX platform)

## 85.9 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

## 85.10 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.

## 85.11 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.

## 85.12 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.

### 85.12.1 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
```

### 85.12.2 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).depack(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.

**class** `kivy.input.providers.tuio.Tuio2dObjMotionEvent`(*device*, *id*, *args*)

Bases: `kivy.input.providers.tuio.TuioMotionEvent`

A 2dObj TUIO object.



# 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>



# 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

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
```



LEAP MOTION - FINGER ONLY



# 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





NATIVE SUPPORT OF  
MULTITOUCHSUPPORT  
FRAMEWORK FOR MACBOOK  
(MAXOSX PLATFORM)



# 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 there are any touches activated by another provider, the mouse event will be discarded. Add this to your configuration:

```
[input]
mouse = mouse,disable_on_activity
```

## 91.1 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 1.8.1.

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), `multitouch_sim` will be added to touch’s profile, and property `multitouch_sim` to the touch. By default `multitouch_sim` is `True` and multitouch will be emulated for that touch. However, if `multitouch_on_demand` is added to the config:

```
[input]
mouse = mouse,multitouch_on_demand
```

then `multitouch_sim` defaults to `False`. In that case, if before mouse release (e.g. in `on_touch_down/move`) `multitouch_sim` is set to `True`, the touch will simulate multi-touch. For example:

```
if 'multitouch_sim' in touch.profile:
    touch.multitouch_sim = True
```

Following is a list of the supported profiles for `MouseEvent`.

Profile name	Description
<code>button</code>	Mouse button (left, right, middle, scrollup, scrolldown) Use property <i>button</i>
<code>pos</code>	2D position. Use properties <i>x</i> , <i>y</i> or <i>pos'</i>
<code>multitouch_sim</code>	If multitouch is simulated. Use property <i>multitouch_sim</i> . See documation above.

# 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



# 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.





# 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.

## 94.1 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
```

## 94.2 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.

**class** `kivy.input.providers.tuio.Tuio2dObjMotionEvent`(*device*, *id*, *args*)

Bases: `kivy.input.providers.tuio.TuioMotionEvent`

A 2dObj TUIO object.

# COMMON DEFINITIONS FOR A WINDOWS PROVIDER

This file provides common definitions for constants used by WM\_Touch / WM\_Pen.



## 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.



## 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.





# 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.

## 98.1 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.

## 98.2 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

```

## 98.3 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:ang

```

## 98.4 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.

### 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.



## MOTION EVENT SHAPE

Represent the shape of the `MotionEvent`

```
class kivy.input.shape.Shape
```

Bases: `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 fo the rect



---

# INTERACTIVE LAUNCHER

---

New in version 1.3.0.

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.

---

## 100.1 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.

---

## 100.2 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

# 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.

---

## 100.3 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.

---

## 100.4 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()
```

#### 100.4.1 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: 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.



---

# 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`.

## 101.1 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.

## 101.2 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 instance.

*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).
- 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.

## 101.3 Value Expressions, `on_property` Expressions, 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):

```
<Widget>:
    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")
```

Furthermore, if a class definition contains an `id`, you can use it as a keyword:

```
<Widget>:
    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).

### 101.3.1 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 multi-line statements 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 an invalid statement:

```
on_state:
    if self.state == 'normal':
        print('normal')
```

## 101.4 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!'.

## 101.5 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.

## 101.6 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 it's 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.

---

## 101.7 Templates

Changed in version 1.7.0: Template usage is now deprecated. Please use Dynamic classes instead.

### 101.7.1 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.
```

## 101.7.2 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 :)

### 101.7.3 Template limitations

When you are creating a context:

1. you cannot use references other than “root”:

```
<MyRule>:
  Widget:
    id: mywidget
    value: 'bleh'
  Template:
    ctxkey: mywidget.value # << fail, this reference mywidget id
```

1. 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, the context will not update the
    # context
```

## 101.8 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.

## 101.9 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>
```

## 101.9.1 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)
```

## 101.9.2 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
```

### 101.9.3 include <file>

New in version 1.8.1.

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: object

A lightweight class allowing to get an object be bound to action in kv, without using as much resources as EventDispatcher

New in version 1.8.1.

**bind**(\*\*kwargs)

This method is to be overridden by your subclass

kwargs will contains callables to call when your observables are updated, so you can trigger a reevaluation of the expression when you need it, just calling all the callbacks that are relevant.

kivy.lang.Builder = <kivy.lang.BuilderBase object at 0x99236ac>

Main instance of a **BuilderBase**.

#### class kivy.lang.BuilderBase

Bases: 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*)

Search all the rules that match the widget and apply them.

**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.

**match**(*widget*)

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. .. versionadded:: 1.0.5

With templates, you can construct custom widgets from a kv lang definition by giving them a context. Check [Template usage](#).

**unbind\_widget**(*uid*)

(internal) Unbind all the handlers created by the rules of the widget. The `kivy.uix.widget.Widget.uid` is passed here instead of the widget itself, because we are using it in the widget destructor.

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.

**class** `kivy.lang.BuilderException`(*context*, *line*, *message*, *cause=None*)

Bases: `kivy.lang.ParserException`

Exception raised when the Builder failed to apply a rule on a widget.

**class** `kivy.lang.Parser`(**\*\*kwargs**)

Bases: `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 #.

**class** `kivy.lang.ParserException`(*context, line, message, cause=None*)

Bases: `exceptions.Exception`

Exception raised when something wrong happened in a kv file.





# EXTERNAL LIBRARIES

Kivy comes with other python/C libraries :

- ddslib
- oscAPI (modified / optimized)
- mtdev

**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.

## 102.1 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.



# 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.



---

# 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')
```

## 104.1 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: `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.





---

# 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
```

## 105.1 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
```

More information about the allowed values are described in the [kivy.config](#) module.

## 105.2 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 = <logging.Logger object at 0x90992cc>  
    Kivy default logger instance
```

```
class kivy.logger.LoggerHistory(level=0)  
    Bases: logging.Handler  
    Kivy history handler
```

---

# 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.

## 106.1 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 other 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.

## 106.2 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)
```

## 106.3 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_FONTSCALE`: 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_FONTSCALE=1.2 python main.py
```

`kivy.metrics.Metrics = <kivy.metrics.MetricsBase object at 0x9879eac>`  
 Default instance of `MetricsBase`, used everywhere in the code .. versionadded:: 1.7.0

**class kivy.metrics.MetricsBase**

Bases: `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

**kivy.metrics.metrics** = <kivy.metrics.MetricsBase object at 0x9879eac>

default instance of **MetricsBase**, used everywhere in the code (deprecated, use *Metrics* instead.)



---

# 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 heirarchy and widget properties.
- **webdebugger**: Realtime examination of your app internals via a web browser.

Modules are automatically loaded from the Kivy path and User path:

- *PATH\_TO\_KIVY/kivy/modules*
- *HOME/.kivy/mods*

## 107.1 Activating a module

There are various ways in which you can activate a kivy module.

### 107.1.1 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.

### 107.1.2 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', '')
```

### 107.1.3 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.

---

## 107.2 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.

### 107.3 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.



## 107.3.1 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 Windows [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.

## 107.4 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.

### 107.4.1 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)
```

## 107.5 Monitor module

The Monitor module is a toolbar that shows the activity of your current application :

- FPS
- Graph of input events

### 107.5.1 Usage

For normal module usage, please see the [modules](#) documentation.

## 107.6 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

### 107.6.1 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

### 107.6.2 Usage

For normal module usage, please see the [modules](#) documentation.

## 107.7 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.

## 107.8 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.

### 107.8.1 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.

**show\_cursor:** boolean, default to False New in version 1.8.0.

**cursor\_image:** str, defaults to 'atlas://data/images/defaulttheme/slider\_cursor'  
Image used to represent the cursor if displayed .. versionadded:: 1.8.0

**cursor\_size:** tuple, defaults to (None, None) Apparent size of the mouse cursor, if displayed, default value will keep its real size. .. versionadded:: 1.8.0

**cursor\_offset:** tuple, defaults to (None, None) Offset of the texture image, default value , will align the top-left corner of the image to the mouse pos. .. version-added:: 1.8.0

## 107.8.2 Example

In your configuration (*~/.kivy/config.ini*), you can add something like this:

```
[modules]
touchring = image=mypointer.png,scale=.3,alpha=.7
```

## 107.9 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/>

---

# 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.

## 108.1 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 Windows `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.

---

# 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.

## 109.1 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)
```





# MONITOR MODULE

The Monitor module is a toolbar that shows the activity of your current application :

- FPS
- Graph of input events

## 110.1 Usage

For normal module usage, please see the [modules](#) documentation.



# 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

## 111.1 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

## 111.2 Usage

For normal module usage, please see the `modules` documentation.



# 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.



---

# 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.

## 113.1 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.

**show\_cursor:** boolean, default to False New in version 1.8.0.

**cursor\_image:** str, defaults to '`atlas://data/images/defaulttheme/slider_cursor`'  
Image used to represent the cursor if displayed .. versionadded:: 1.8.0

**cursor\_size:** tuple, defaults to (None, None) Apparent size of the mouse cursor, if displayed, default value will keep its real size. .. versionadded:: 1.8.0

**cursor\_offset:** tuple, defaults to (None, None) Offset of the texture image, default value , will align the top-left corner of the image to the mouse pos. .. version-added:: 1.8.0

## 113.2 Example

In your configuration (`~/.kivy/config.ini`), you can add something like this:

```
[modules]
touchring = image=mypointer.png,scale=.3,alpha=.7
```





# 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/>



---

# MULTISTROKE GESTURE RECOGNIZER

New in version 1.8.1.

**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.

## 115.1 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()**.

## 115.2 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
    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

```

## 115.3 Algorithm details

For more information about the matching algorithm, see:

“Protractor: A fast and accurate gesture recognizer” by Yang Li <http://yangli.org/pdf/protractor-chi2010.pdf>

“\$N-Protractor” by Lisa Anthony and Jacob O. Wobbrock <http://depts.washington.edu/aimgroup/proj/dollar/ndoprotractor.pdf>

Some of the code is derived from the JavaScript implementation here: <http://depts.washington.edu/aimgroup/proj/dollar/ndoprotractor.pdf>

```

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 captial 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])

# 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: 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: “\$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: object
```

Represents a (uni)stroke path as a list of Vectors. Normally, this class is instantiated by MultistrokeGesture 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: 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*)



---

# NETWORK SUPPORT

Kivy currently supports basic, asynchronous network requests. Please refer to [kivy.network.urlrequest.UrlRequest](#).

## 116.1 Url Request

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 XHR 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)
```

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 weather in Paris:

```
def got_weather(req, results):
    for key, value in results['weather'][0].items():
        print(key, ': ', value)

req = UrlRequest(
    'http://api.openweathermap.org/data/2.5/weather?q=Paris,fr',
    got_weather)
```

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, time-
                                         out=None, method=None, decode=True,
                                         debug=False, file_path=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

#### 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.
- 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.

Changed in version 1.8.0: Parameter `decode` added. Parameter `file_path` added. Parameter `on_redirect` added. Parameter `on_failure` added.

#### **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.

**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.





---

## URL REQUEST

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 XHR 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)
```

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 weather in Paris:

```
def got_weather(req, results):
    for key, value in results['weather'][0].items():
        print(key, ': ', value)

req = UrlRequest(
    'http://api.openweathermap.org/data/2.5/weather?q=Paris,fr',
    got_weather)
```

Example of Posting data (adapted from httplib 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, time-
                                         out=None, method=None, decode=True,
                                         debug=False, file_path=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

#### 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.
- 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.

Changed in version 1.8.0: Parameter `decode` added. Parameter `file_path` added. Parameter `on_redirect` added. Parameter `on_failure` added.

#### **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.

**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.



---

## 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)`
- `aaa`
- `rrggbb`

For hexadecimal values, you can also use:

- `#aaa`
- `#rrggbb`

`kivy.parser.parse_int`

alias of `int`

`kivy.parser.parse_float`

alias of `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).



---

# 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.

## 119.1 Comparison Python vs. Kivy

### 119.1.1 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)
```

### 119.1.2 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!

### 119.1.3 Error Handling

If setting a value would otherwise raise a `ValueError`, you have two options to handle the error gracefully within the property. An `errorvalue` is a substitute for the invalid value. An `errorhandler` is a callable (single argument function or lambda) which can return a valid substitute.

`errorvalue` parameter:

```

# simply returns 0 if the value exceeds the bounds
bnp = BoundedNumericProperty(0, min=-500, max=500, errorvalue=0)

```

`errorhandler` parameter:

```

# returns the boundary value when exceeded
bnp = BoundedNumericProperty(0, min=-500, max=500,
    errorhandler=lambda x: 500 if x > 500 else -500)

```

### 119.1.4 Conclusion

Kivy properties are easier to use than the standard ones. See the next chapter for examples of how to use them :)

## 119.2 Observe Properties 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.



### 119.2.1 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
```

### 119.2.2 Observe using 'on\_<propname>'

If you created 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.

## 119.3 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 1.8.1, 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

Bases: object

Base class for building more complex properties.

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*).

Changed in version 1.4.2: Parameters *errorhandler* and *errorvalue* added

### bind()

Add a new observer to be called only when the value is changed.

### dispatch()

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)
```

### get()

Return the value of the property.

### link()

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.

**set()**

Set a new value for the property.

**unbind()**

Remove the observer from our widget observer list.

**class** `kivy.properties.NumericProperty`

Bases: `kivy.properties.Property`

Property that represents a numeric value.

**Parameters**

*default*: **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()**

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`

Bases: `kivy.properties.Property`

Property that represents a string value.

**Parameters**

*default*: **string, defaults to ""** Specifies the default value of the property.

**class** `kivy.properties.ListProperty`

Bases: `kivy.properties.Property`

Property that represents a list.

**Parameters**

*default*: **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 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 = widget.my_list = [1, 5, 7]
>>> print my_list is widget.my_list
False
>>> my_list.append(10)
>>> print(my_list, widget.my_list)
[1, 5, 7, 10], [1, 5, 7]
```

### class `kivy.properties.ObjectProperty`

Bases: `kivy.properties.Property`

Property that represents a Python object.

#### Parameters

**default:** `object` 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 1.8.1: `rebind` has been introduced.

Changed in version 1.7.0: `baseclass` parameter added.

### class `kivy.properties.BooleanProperty`

Bases: `kivy.properties.Property`

Property that represents only a boolean value.

#### Parameters

**default:** `boolean` Specifies the default value of the property.

### class `kivy.properties.BoundedNumericProperty`

Bases: `kivy.properties.Property`

Property that represents a numeric value within a minimum bound and/or 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()

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()

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()

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()

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)

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

Bases: `kivy.properties.Property`

Property that represents a string from a predefined list of valid 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.

#### options

Return the options available.

New in version 1.0.9.

### class `kivy.properties.ReferenceListProperty`

Bases: `kivy.properties.Property`

Property that allows the creation of a tuple of other properties.

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`.

### class `kivy.properties.AliasProperty`

Bases: `kivy.properties.Property`

Create a property with a custom getter and setter.

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`:

```
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'))
```

#### Parameters

**getter:** function Function to use as a property getter

**setter:** function Function to use as a property setter. Properties listening to the alias property won't be updated 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

**cache:** boolean If `True`, the value will be cached, until one of the binded elements will changes

**rebind:** bool, defaults to `False` See `ObjectProperty` for details.

Changed in version 1.8.1: `rebind` has been introduced.

Changed in version 1.4.0: Parameter `cache` added.

### class `kivy.properties.DictProperty`

Bases: `kivy.properties.Property`

Property that represents a dict.

#### Parameters

**default:** dict, defaults to `None` Specifies the default value of the property.

**rebind:** bool, defaults to `False` See `ObjectProperty` for details.

Changed in version 1.8.1: `rebind` has been introduced.

**Warning:** Similar to `ListProperty`, when assigning a dict to a `DictProperty`, the dict stored in the property is a copy of the dict and not the original dict. See `ListProperty` for details.

### class `kivy.properties.VariableListProperty`

Bases: `kivy.properties.Property`

A `ListProperty` that allows you to work with a variable amount of 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.

### class `kivy.properties.ConfigParserProperty`

Bases: `kivy.properties.Property`

Property that allows one to bind to changes in the configuration values 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 represent 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 1.8.1.

### **set\_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())
```



# 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.

`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.



---

# STORAGE

New in version 1.7.0.

**Warning:** This module is still experimental, and the API is subject to change in a future version.

## 121.1 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](#)

## 121.2 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
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')

```

## 121.3 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, entry):
    print('the key', key, 'have', entry)
mystore.get('plop', callback=my_callback)

```

The callback signature is (for almost all methods) `callback(store, key, result)`:

```

#. 'store' is the 'Store' instance currently used.
#. 'key' is the key to search for.
#. 'entry' is the result of the lookup for the 'key'.

```

## 121.4 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:** `bool` 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) availables
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.

## 121.5 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.

## 121.6 JSON store

Can be used to save/load key-value pairs from a json file.

**class** `kivy.storage.jsonstore.JsonStore(filename, **kwargs)`

Bases: `kivy.storage.AbstractStore`

Store implementation using a json file for storing the keys-value pairs. See the `kivy.storage` module documentation for more information.



## 121.7 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.



## 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.



# JSON STORE

Can be used to save/load key-value pairs from a json file.

**class** `kivy.storage.jsonstore.JsonStore`(*filename*, *\*\*kwargs*)

Bases: `kivy.storage.AbstractStore`

Store implementation using a json file for storing the keys-value pairs. See the `kivy.storage` module documentation for more information.



---

# 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.





## 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 1.8.1.

`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.



---

# WIDGETS

A widget is an element of a graphical user interface. The *kivy.uix* module contains classes for creating and managing Widgets.

First read: *Widget class*

- **UX widgets:** Classical user interface widgets, perfect and 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 *Layout*.

*Grid Layout, Box Layout, Anchor 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, List View, TabbedPanel, Video player, VKeyboard,*

- **Behaviors widgets:** These widgets do no rendering but act on the graphics instructions or interaction (touch) behavior.

*Scatter, Stencil View*

- **Screen manager:** Manages screens and transitions when switching from one to another.

*Screen Manager*

---

## 126.1 Abstract View

New in version 1.5: This code is still experimental, and its API is subject to change in a future version.

The **AbstractView** widget has an `adapter` property for an adapter that mediates to data. The adapter manages an `item_view_instances` dict property that holds views for each data item, operating as a cache.

```
class kivy.uix.abstractview.AbstractView(**kwargs)
```

```
    Bases: kivy.uix.floatlayout.FloatLayout
```

```
    View using an Adapter as a data provider.
```

### **adapter**

The adapter can be one of several kinds of **adapters**. The most common example is the **ListAdapter** used for managing data items in a list.

---

## 126.2 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`.

## 126.2.1 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()
```

## 126.2.2 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.

### **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.

## 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.

**orientation**

Link to the `Accordion.orientation` property.

**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
```

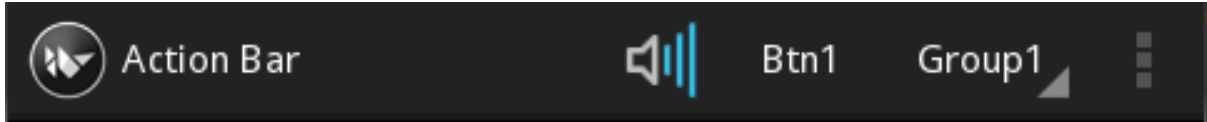
`class kivy.uix.accordion.AccordionException`

Bases: `exceptions.Exception`

AccordionException class.

## 126.3 Action Bar

New in version 1.8.0.



The ActionBar widget is like Android's ActionBar, where items are stacked horizontally.

The `ActionBar` will contain one `ActionView` and many `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` will contain an `ActionOverflow`. A `ContextualActionView` is a subclass of an `ActionView`.

```
class kivy.uix.actionbar.ActionBarException
```

Bases: `exceptions.Exception`

ActionBarException class

```
class kivy.uix.actionbar.ActionItem
```

Bases: `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 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.

`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`.

```
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 it's 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.

```
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.spinner.Spinner`

ActionGroup class, see module documentation for more information.

## 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 a `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.

### **overflow\_image**

Image to be used as an Overflow Image.

`overflow_image` is an `ObjectProperty` 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.

### **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 an `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`.

### **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.actionbar.ActionButton`

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'`.

### **previous\_image**

Image for the 'previous' ActionButtons default graphical representation.

`previous_image` is a `StringProperty` and defaults to `'atlas://data/images/defaulttheme/previous_normal'`.

### **title**

Title for ActionView.

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

### **with\_previous**

Specifies whether clicking on ActionPrevious will load the previous screen or not. If True, the previous\_icon will be shown otherwise it will not.

`with_previous` is a `BooleanProperty` and defaults to True.

`class kivy.uix.actionbar.ActionBar(**kwargs)`

Bases: `kivy.uix.boxlayout.BoxLayout`

ActionBar, see the module documentation for more information.

#### **Events**

`on_previous` Fired when action\_previous of action\_view is pressed.

### **action\_view**

action\_view of ActionBar.

`action_view` is an `ObjectProperty` and defaults to an instance of ActionView.

### **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 ActionBars default graphical representation.

`background_image` is an `StringProperty` and defaults to `'atlas://data/images/defaulttheme/action_bar'`.

### **border**

`border` to be applied to the `background_image`.

## 126.4 Anchor Layout



The **AnchorLayout** aligns 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'.

**padding**

Padding between the widget box and it's 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].

## 126.5 Behaviors

New in version 1.8.0.

This module implements behaviors that can be mixed with existing base widgets. For example, if you want to add a “button” capability to an *Image*, you could do:

```
class IconButton(ButtonBehavior, Image):
    pass
```

---

**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.

---

```
class kivy.uix.behaviors.ButtonBehavior(**kwargs)
```

Bases: object

Button behavior.

### 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).

### 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](#), defaults to None.

### state

State of the button, must be one of ‘normal’ or ‘down’. The state is ‘down’ only when the button is currently touched/clicked, otherwise ‘normal’.

*state* is an [OptionProperty](#).

```
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.ButtonBehavior](#)

ToggleButton behavior, see *ToggleButton* module documentation for more information.

New in version 1.8.0.

```
static get_widgets(groupname)
```

Return the widgets contained in a specific group. If the group doesn’t exist, an empty list will be returned.

---

**Important:** Always release the result of this method! 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. Garbage collector might need more elements before flushing it. The return of this method is informative, you've been warned!

### group

Group of the button. If None, no group will be used (button is independent). If specified, **group** must be a hashable object, like a string. Only one button in a group can be in 'down' state.

**group** is a **ObjectProperty**

```
class kivy.uix.behaviors.DragBehavior(**kwargs)
```

Bases: object

Drag behavior. When combined with a widget, dragging in the rectangle defined by **drag\_rectangle** will drag the widget.

For example, to make a popup which is draggable by its title do:

```
from kivy.uix.behaviors import DragBehavior
from kivy.uix.popup import Popup

class DragPopup(DragBehavior, Popup):
    pass
```

And in .kv do::

```
<DragPopup>:drag_rectangle: self.x, self.y+self._container.height, self.width, self.height -
self._container.height drag_timeout: 10000000 drag_distance: 0
```

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 go to children. It is advisable that you base this value on the dpi of your target device's screen.

**drag\_distance** is a **NumericProperty**, defaults to 20 (pixels), according to the default value of **scroll\_distance** in user configuration.

### drag\_rect\_height

Height of the axis aligned bounding rectangle where dragging is allowed.

**drag\_rect\_height** is a **NumericProperty**, defaults to 100.

### drag\_rect\_width

Width of the axis aligned bounding rectangle where dragging is allowed.

**drag\_rect\_width** is a **NumericProperty**, 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**, 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**, 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 go to the children.

`drag_timeout` is a **NumericProperty**, defaults to 55 (milliseconds), according to the default value of `scroll_timeout` in user configuration.

```
class kivy.uix.behaviors.FocusBehavior(**kwargs)
```

Bases: `object`

Implements 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 than 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.

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 previous or next widget is not specified, `focus_next` and `focus_previous` default to `None`, which 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
```

New in version 1.8.1.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

### **focus\_next**

The `FocusBehavior` instance to acquire focus when tab is pressed when 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 a **ObjectProperty**, 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 `:attr:'focus_previous` is the *StopIteration* class, focus won't move backward, but end here.

`focus_previous` is a **ObjectProperty**, defaults to *None*.

### **focused**

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.

`focused` is a **BooleanProperty**, defaults to *False*.

### **is\_focusable**

Whether the instance can become focused. If focused, it'll lose focus when set to *False*.

`is_focusable` is a **BooleanProperty**, defaults to *True* on a desktop (i.e. desktop is *True* in `config`), *False* otherwise.

### **keyboard**

The keyboard to bind, 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 get 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 information on the keyboard modes.

`keyboard` is a **AliasProperty**, defaults to *None*.

### **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` at the end after it is done if it didn't consume 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` at the end after it is done if it didn't consume escape.

See `on_key_down()`

```
class kivy.uix.behaviors.CompoundSelectionBehavior(**kwargs)
    Bases: object
```

Selection behavior implements the logic behind keyboard and touch selection of selectable widgets managed by the derived widget. For example, it could be combined with a `GridLayout` to add selection to the layout.

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, it 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.

When the controller needs to select a node it calls `select_node()` and `deselect_node()`. Therefore, they must be overwritten in order to affect the selected nodes. By default, the class doesn't listen to keyboard and touch events, therefore, 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.

For example, to add selection to a grid layout which will contain `Button` widgets:

```
class SelectableGrid(CompoundSelectionBehavior, GridLayout):

    def __init__(self, **kwargs):
        super(CompoundSelectionBehavior, self).__init__(**kwargs)
        keyboard = Window.request_keyboard(None, self)
        keyboard.bind(on_key_down=self.select_with_key_down,
                    on_key_up=self.select_with_key_up)

    def select_node(self, node):
        node.background_color = (1, 0, 0, 1)
        return super(CompoundSelectionBehavior, self).select_node(node)

    def deselect_node(self, node):
        node.background_color = (1, 1, 1, 1)
        super(CompoundSelectionBehavior, self).deselect_node(node)
```

Then, for each button added to the layout, bind `on_touch_down` of the button to `select_with_touch()` to pass on the touch events.

New in version 1.8.1.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

```
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\_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 is reversed to make it work by default for most layouts so that the final result is that e.g. home, although it will select the last element on this list, visually it'll 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**

*keystr*, 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**

Whether the keyboard can be used for selection. If False, keyboard inputs will be ignored.

`keyboard_select` is a `BooleanProperty`, 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`, defaults to False.

## page\_count

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when pageup (or pagedown) is pressed.

`page_count` is a `NumericProperty`, defaults to 10.

## right\_count

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the right (or left) arrow on the keyboard is pressed.

`right_count` is a `NumericProperty`, defaults to 1.

## scroll\_count

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the mouse scroll wheel is scrolled.

`right_count` is a `NumericProperty`, 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 to its selected state when this is 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 deslect 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 recieved 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.

*selected\_nodes* is a **ListProperty** and defaults to the empty list, []. It is read-only and should not be modified.

### **touch\_multiselect**

A special touch mode which determines whether touch events, as processed with *select\_with\_touch()*, will add to the selection the currently touched node, 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 than *multiselect*, because when this is True simply touching an unselected node will select it, even if e.g. ctrl is not pressed. If this is False, however, ctrl is required to be held in order to add to selection when *multiselect* is True.

---

**Note:** *multiselect*, when False, will disable *touch\_multiselect*.

---

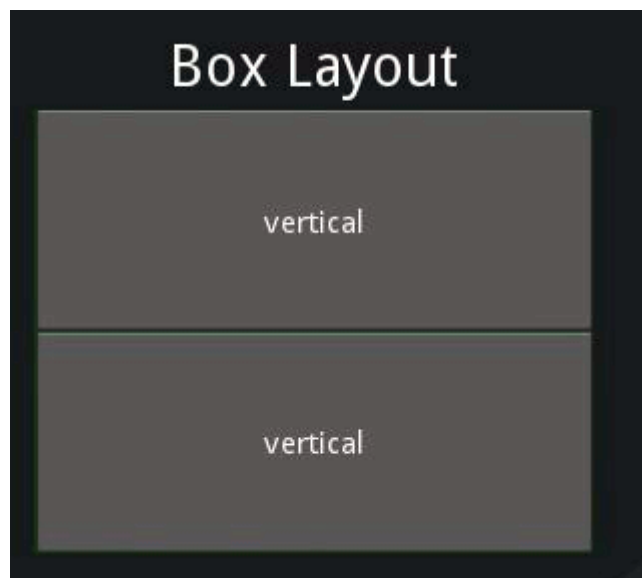
*touch\_multiselect* is a **BooleanProperty**, defaults to False.

### **up\_count**

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the up (or down) arrow on the keyboard is pressed.

*up\_count* is a **NumericProperty**, defaults to 1.

## 126.6 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.

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.

#### **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].

#### **spacing**

Spacing between children, in pixels.

**spacing** is a `NumericProperty` and defaults to 0.

## 126.7 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.

### 126.7.1 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',
```

```

        '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()

```

## 126.7.2 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.
```

### **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).

`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: (top, right, bottom, left). Read the `BorderImage` instructions for more information about how to use it.

`border` is a `ListProperty` and defaults to `(16, 16, 16, 16)`

### **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'`.

### **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)`

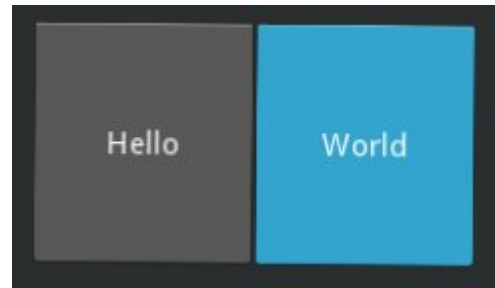
Bases: `kivy.uix.button.Button`

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.



## 126.8 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, you can use the same properties that you can use 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)
```

```
class kivy.uix.button.Button(**kwargs)
```

Bases: `kivy.uix.behaviors.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).

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 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 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: (top, right, bottom, left). Read the `BorderImage` instruction for more information about how to use it.

`border` is a `ListProperty` and defaults to (16, 16, 16, 16)

## 126.9 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].

## 126.10 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 use it horizontally or vertically. The carousel can display pages in loop or not.

Example:

```
class Example1(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 = Factory.AsyncImage(source=src, allow_stretch=True)
            carousel.add_widget(image)
        return carousel
```

```
Example1().run()
```

Changed in version 1.5.0: The carousel now supports active children, like the `ScrollView`. It will detect a swipe gesture according to `Carousel.scroll_timeout` and `Carousel.scroll_distance`.

In addition, the container used for adding a slide is now hidden in the API. We made a mistake by exposing it to the user. 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.

#### **anim\_cancel\_duration**

Defines the duration of the animation when a swipe movement is not accepted. This is generally when the user doesn't swipe enough. 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 in the next/previous slide.

New in version 1.8.0.

#### **current\_slide**

The currently shown slide.

`current_slide` is an `AliasProperty`.

Changed in version 1.5.0: The property doesn't expose the container used for storing the slide. It returns widget you have added.

#### **direction**

Specifies the direction in which the slides are ordered i.e. the direction from which the user swipes to go from one slide to the next. 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 a `OptionProperty` and defaults to `'right'`.

#### **index**

Get/Set the current visible slide based on the index.

`index` is a `AliasProperty` and defaults to 0 (the first item).

#### **load\_next(mode='next')**

Animate to 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 swipe infinitely. When the user reaches the last page, they will return to first page when trying to swipe to the next.

`loop` is a `BooleanProperty` and defaults to `False`.

### **min\_move**

Defines the minimal distance from the edge where the movement is considered a swipe gesture and the Carousel will change its content. This is a percentage of the Carousel width. If the movement doesn't reach this minimal value, then 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. If `orientation` is 'horizontal', the next slide is to the right. If `orientation` is 'vertical', the next slide is towards the bottom.

`next_slide` is a `AliasProperty`.

Changed in version 1.5.0: The property doesn't expose the container used for storing the slide. It returns the widget you have added.

### **previous\_slide**

The previous slide in the Carousel. It is None if the current slide is the first slide in the Carousel. If `orientation` is 'horizontal', the previous slide is to the left. If `orientation` is 'vertical', the previous slide towards the bottom.

`previous_slide` is a `AliasProperty`.

Changed in version 1.5.0: This property doesn't expose the container used for storing the slide. It returns the widget you have added.

### **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, the scrolling will be disabled 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.

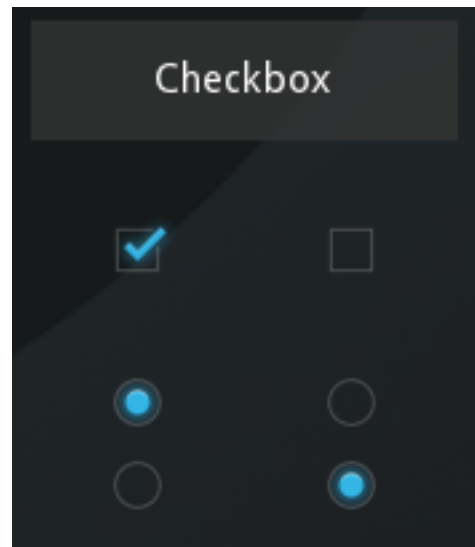
### **slides**

List of slides inside the Carousel. The slides are added when a widget is added to Carousel using `add_widget()`.

`slides` is a `ListProperty` and is read-only.

## 126.11 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.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 **BooleanProperty** and defaults to **False**.

## 126.12 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

```

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.

### 126.12.1 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.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`.

## 126.13 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 selected color. 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/DroidSansMono.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.

**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.

## 126.14 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.

### 126.14.1 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))
```

```
runTouchApp(mainbutton)
```

## 126.14.2 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.

### 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 allow 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.

**container**

(internal) Property that will be set to the container of the dropdown list. It is a **GridLayout** by default.

**dismiss (\*largs)**

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.

**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.

**select (data)**

Call this method to trigger the *on\_select* event with the *data* selection. The *data* can be anything you want.

## 126.15 EffectWidget

New in version 1.8.1: This code is still experimental, and its API is subject to change in a future version.

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 antialiasing, to emulating the appearance of a crt monitor!

The basic usage is as follows:

```
w = EffectWidget()
w.add_widget(Button(text='Hello!'))
w.effects = [InvertEffect(), HorizontalBlurEffect(size=2.0)]
```

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.

---

**Note:** It is not efficient to resize an **EffectWidget**, as each **Fbo** is recreated every time. If you need to resize frequently, consider doing things a different way.

---

**Note:** 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.

---

### 126.15.1 Provided Effects

The module comes with several pre-written effects. Some have adjustable properties (e.g. blur radius), see the individual effect documentation for more details.

- `MonochromeEffect` - makes the widget grayscale.
- `InvertEffect` - inverts the widget colors.
- `ChannelMixEffect` - swaps around color channels.
- `ScanlinesEffect` - displays flickering scanlines.
- `PixelateEffect` - pixelates the image.
- `HorizontalBlurEffect` - Gaussian blurs horizontally.
- `VerticalBlurEffect` - Gaussian blurs vertically.
- `FXAAEffect` - applies a very basic AA.

### 126.15.2 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 colour at each point, then call your `effect` function to transform it. The parameters are:

- **color:** The normal colour 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 module documentation for full information on setting effects and creating your own.

#### 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, 1)

#### effects

List of all the effects to be applied. These should all be instances 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.

#### texture

The output texture of our 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 module documentation for more details.

#### fbo

The fbo currently using this effect. The `EffectBase` automatically handles this.

`fbo` is a `ObjectProperty` and defaults to None.

#### glsl

The glsl string defining your effect function, see 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 if 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 values for the uniform variable with the given name are uploaded to the shader.

You must still manually declare your uniform variables at the top of your glsl string.

#### **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 colours in monochrome.

`class kivy.uix.effectwidget.InvertEffect(*args, **kwargs)`

Bases: `kivy.uix.effectwidget.EffectBase`

Inverts the colours 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 antialiasing via fxaa.

## 126.16 FileChooser

New in version 1.0.5.

Changed in version 1.2.0: In the chooser template, the `controller` is not a direct reference anymore but a weak-reference. You must update all the notation `root.controller.xxx` to `root.controller().xxx`.

### 126.16.1 Simple example

main.py

```
#!/usr/bin/env python
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()

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"
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)

```

```
class kivy.uix.filechooser.FileChooserListView(**kwargs)
```

Bases: `kivy.uix.filechooser.FileChooserController`

Implementation of `FileChooserController` using a list view.

```
class kivy.uix.filechooser.FileChooserIconView(**kwargs)
```

Bases: `kivy.uix.filechooser.FileChooserController`

Implementation of `FileChooserController` using an icon view.

```
class kivy.uix.filechooser.FileChooserController(**kwargs)
```

Bases: `kivy.uix.floatlayout.FloatLayout`

Base for implementing a FileChooser. Don't use this class directly, but prefer using an implementation such as the `FileChooserListView` or `FileChooserIconView`.

#### Events

`on_entry_added`: `entry`, `parent`Fired when a root-level entry is added to the file

list.

**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.

**on\_remove\_subentry: entry, parent** Fired when entries are removed from an entry, usually 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**

**BooleanProperty**, defaults to False. Determines whether directories are valid selections or not.

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 weird filename, undecodable without knowing it's 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.

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\_encodings is a **ListProperty** and defaults to ['utf-8', 'latin1', 'cp1252'],

**file\_system**

Implementation to access the file system. Must be an instance of **FileSystemAbstract**.

New in version 1.8.0.

**ObjectProperty**, defaults to **FileSystemLocal()**

**files**

Read-only **ListProperty**. The list of files in the directory specified by path after applying the filters.

**filter\_dirs**

**BooleanProperty**, defaults to False. Indicates whether filters should also apply to directories.

**filters**

**ListProperty**, defaults to [], equal to `**`. Specifies the filters to be applied to the files in the directory.

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
<code>*</code>	matches everything
<code>?</code>	matches any single character
<code>[seq]</code>	matches any character in seq
<code>[!seq]</code>	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: If the filter is a callable (function or method), it will be called with the path and the file name as arguments for each file in the directory. The callable should return `True` to indicate a match and `False` otherwise.

### **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.).

### **multiselect**

**BooleanProperty**, defaults to `False`. Determines whether the user is able to select multiple files or not.

### **path**

**StringProperty**, 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 question marks (?) instead of their unicode characters.

### **progress\_cls**

Class to use for displaying a progress indicator for filechooser loading.

New in version 1.2.0.

**ObjectProperty**, defaults to `FileChooserProgress`.

Changed in version 1.8.0: If you set a string, the **Factory** will be used to resolve the class.

### **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`.

New in version 1.2.0.

**StringProperty**, defaults to `None`.

---

**Note:** Similar to `path`, if `rootpath` is specified, whether it's a bytes or unicode string determines the type of the filenames and paths read.

---

### **selection**

Read-only **ListProperty**. Contains the list of files that are currently selected.

### **show\_hidden**

**BooleanProperty**, defaults to `False`. Determines whether hidden files and folders should be shown.

### **sort\_func**

**ObjectProperty**. Provides a function to be called with a list of filenames, and the filesystem implementation as the second argument. Returns a list of filenames sorted for display in the view.

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.

### **path**

Current path of the FileChooser, read-only.

### **total**

Total number of entries to load.

**class** `kivy.uix.filechooser.FileSystemAbstract`

Bases: `object`

Class for implementing a File System view that can be used with the `FileChooser`.attr:~*FileChooser.file\_system*.

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.

## 126.17 Float Layout

The `FloatLayout` class honors only the `Widget.pos_hint` and `Widget.size_hint` attributes.



For example, say you create a FloatLayout with a size of (300, 300):

```
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 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.

## 126.18 Gesture Surface

New in version 1.8.1.

**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, ie 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, ie 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.

**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



## 126.19 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.

### 126.19.1 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.

### 126.19.2 Column Width and Row Height

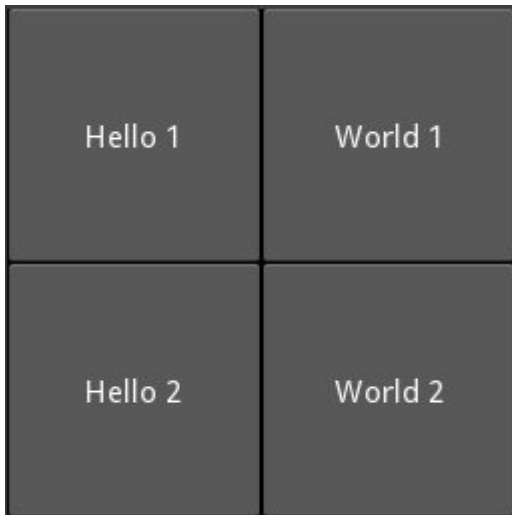
The column width/row height are determined in 3 steps:

- 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.

### 126.19.3 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'))
```

Hello 1	World 1
Hello 2	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 0.

#### **cols\_minimum**

List of minimum sizes for each column.

New in version 1.0.7.

`cols_minimum` is a `DictProperty` and defaults to {}.

#### **minimum\_height**

Minimum height needed to contain all children.

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.

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.

New in version 1.0.8.

`minimum_width` is a `kivy.properties.NumericProperty` and defaults to 0.

#### **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 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 0.

#### **rows\_minimum**

List of minimum sizes for each row.

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].

#### **class kivy.uix.gridlayout.GridLayoutException**

Bases: exceptions.Exception

Exception for errors if the grid layout manipulation fails.

## 126.20 Image

The **Image** widget is used to display an image:

```
wimg = Image(source='mylogo.png')
```

### 126.20.1 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.

## 126.20.2 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:  
            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).

### **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 a `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 a `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.

Depending of the texture creation, the value will be a `Texture` or a `TextureRegion` object.

`texture` is a `ObjectProperty` and defaults to None.

## texture\_size

Texture size of the image.

**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.

---

## 126.21 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')
```

### 126.21.1 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

**[font=<str>][font]** Change the font

**[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.

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&bt;[/size]"
```

## 126.21.2 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]'
```

## 126.21.3 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):
    """ 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]

```

```
    [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`.

---

### **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**

Text color, in the format (r, g, b, a)

New in version 1.8.0.

`disabled_color` is a `ListProperty` and defaults to [1, 1, 1, .5].

### **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 `'DroidSans'`.

### **font\_size**

Font size of the text, in pixels.

`font_size` is a `NumericProperty` and defaults to 12dp.

### **halign**

Horizontal alignment of the text.

`halign` is an `OptionProperty` and defaults to `'left'`. Available options are : left, center, right and justify.

**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 1.6.0: A new option was added to `halign`, namely *justify*.

### **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.

### **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 1.8.1: `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 1.8.1: `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 1.8.1.

`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 `shorten_from` is the empty string, "", we split on every character fitting as much text as possible into the line.

New in version 1.8.1.

`split_str` is a `StringProperty` and defaults to "" (the empty string).

## 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 1.8.1.

`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\_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 constrains. 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 `texture_size` is `[None, None]`, the texture will be the size required to fit the text, otherwise it's clipped to fit `texture_size`.

When `texture_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(\*largs)**

Force texture recreation with the current Label properties.

After this function call, the `texture` and `texture_size` will be updated in this order.

### **valign**

Vertical alignment of the text.

`valign` is an `OptionProperty` and defaults to `'bottom'`. Available options are : `bottom`, `middle` and `top`.

**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.

## 126.22 Layout

Layouts are used to calculate and assign widget positions.

The `Layout` class itself cannot be used directly. You should use one of the concrete 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`
- Stack layout : `kivy.uix.stacklayout.StackLayout`

### 126.22.1 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
```

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.

```
do_layout(*args)
```

This function is called when a layout is needed by a trigger. If you are writing a new `Layout` subclass, don't call this function directly but use `_trigger_layout()` instead.

New in version 1.0.8.

## 126.23 List View

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

The `ListView` widget provides a scrollable/pannable viewport that is clipped to the scrollview's bounding box which contains list item view instances.

The `ListView` implements an `AbstractView` as a vertical, scrollable list. The `AbstractView` has one property: `adapter`. The `ListView` sets an adapter to one of a `SimpleListAdapter`, `ListAdapter` or a `DictAdapter`.

### 126.23.1 Introduction

Lists are central parts of many software projects. Kivy's approach to lists includes providing solutions for simple lists, along with a substantial framework for building lists of moderate to advanced complexity. For a new user, it can be difficult to ramp up from simple to advanced. For this reason, Kivy provides an extensive set of examples that you may wish to run first, to get a taste of the range of functionality offered. You can tell from the names of the examples that they illustrate the "ramping up" from simple to advanced:

- `kivy/examples/widgets/lists/list_simple.py`
- `kivy/examples/widgets/lists/list_simple_in_kv.py`
- `kivy/examples/widgets/lists/list_simple_in_kv_2.py`
- `kivy/examples/widgets/lists/list_master_detail.py`
- `kivy/examples/widgets/lists/list_two_up.py`
- `kivy/examples/widgets/lists/list_kv.py`
- `kivy/examples/widgets/lists/list_composite.py`

- `kivy/examples/widgets/lists/list_cascade.py`
- `kivy/examples/widgets/lists/list_cascade_dict.py`
- `kivy/examples/widgets/lists/list_cascade_images.py`
- `kivy/examples/widgets/lists/list_ops.py`

Many of the examples feature selection, some restricting selection to single selection, where only one item at a time can be selected, and others allowing multiple item selection. Many of the examples illustrate how selection in one list can be connected to actions and selections in another view or another list.

Find your own way of reading the documentation here, examining the source code for the example apps and running the examples. Some may prefer to read the documentation through first, others may want to run the examples and view their code. No matter what you do, going back and forth will likely be needed.

## 126.23.2 Basic Example

In its simplest form, we make a listview with 100 items:

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

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 2
        super(MainView, self).__init__(**kwargs)

        list_view = ListView(
            item_strings=[str(index) for index in range(100)])

        self.add_widget(list_view)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))
```

Or, we could declare the listview using the kv language:

```
from kivy.uix.modalview import ModalView
from kivy.uix.listview import ListView
from kivy.uix.gridlayout import GridLayout
from kivy.lang import Builder

Builder.load_string("""
<ListViewModal>:
    size_hint: None, None
    size: 400, 400
    ListView:
        size_hint: .8, .8
        item_strings: [str(index) for index in range(100)]
""")

class ListViewModal(ModalView):
```



```

def __init__(self, **kwargs):
    super(ListViewModal, self).__init__(**kwargs)

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 1
        super(MainView, self).__init__(**kwargs)

        listview_modal = ListViewModal()

        self.add_widget(listview_modal)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))

```

### 126.23.3 Using an Adapter

Behind the scenes, the basic example above uses the `SimpleListAdapter`. When the constructor for the `ListView` sees that only a list of strings is provided as an argument (called `item_strings`), it creates an instance of `SimpleListAdapter` using the list of strings.

Simple in `SimpleListAdapter` means: *without selection support*. It is a scrollable list of items that does not respond to touch events.

To use a `SimpleListAdapter` explicitly when creating a `ListView` instance, do:

```

simple_list_adapter = SimpleListAdapter(
    data=["Item #{0}".format(i) for i in range(100)],
    cls=Label)

list_view = ListView(adapter=simple_list_adapter)

```

The instance of `SimpleListAdapter` has a required data argument which contains data items to use for instantiating Label views for the list view (note the `cls=Label` argument). The data items are strings. Each item string is set by the `SimpleListAdapter` as the `text` argument for each Label instantiation.

You can declare a `ListView` with an adapter in a kv file with special attention given to the way longer python blocks are indented:

```

from kivy.uix.modalview import ModalView
from kivy.uix.listview import ListView
from kivy.uix.gridlayout import GridLayout
from kivy.lang import Builder
from kivy.factory import Factory

# Note the special nature of indentation in the adapter declaration, where
# the adapter: is on one line, then the value side must be given at one
# level of indentation.

Builder.load_string("""
#:import label kivy.uix.label
#:import sla kivy.adapters.simplelistadapter

<ListViewModal>:

```

```

size_hint: None, None
size: 400, 400
ListView:
    size_hint: .8, .8
    adapter:
        sla.SimpleListAdapter(
            data=["Item #{0}".format(i) for i in range(100)],
            cls=label.Label)
"""

class ListViewModal(ModalView):
    def __init__(self, **kwargs):
        super(ListViewModal, self).__init__(**kwargs)

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 1
        super(MainView, self).__init__(**kwargs)

        listview_modal = ListViewModal()

        self.add_widget(listview_modal)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))

```

#### 126.23.4 ListAdapter and DictAdapter

For many uses of a list, the data is more than a simple list of strings. Selection functionality is also often needed. The [ListAdapter](#) and [DictAdapter](#) cover these more elaborate needs.

The [ListAdapter](#) is the base class for [DictAdapter](#), so we can start with it.

See the [ListAdapter](#) docs for details, but here are synopses of its arguments:

- *data*: strings, class instances, dicts, etc. that form the basis data for instantiating views.
- *cls*: a Kivy view that is to be instantiated for each list item. There are several built-in types available, including [ListItemLabel](#) and [ListItemButton](#), or you can make your own class that mixes in the required [SelectableView](#).
- *template*: the name of a Kivy language (kv) template that defines the Kivy view for each list item.

---

**Note:** Pick only one, *cls* or *template*, to provide as an argument.

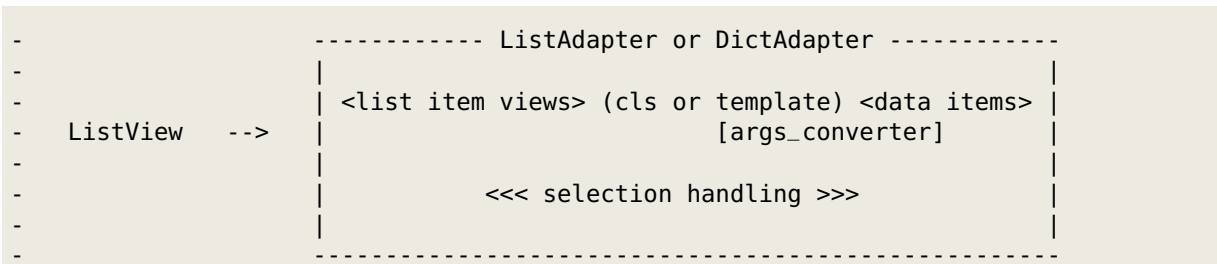
---

- *args\_converter*: a function that takes a data item object as input and uses it to build and return an args dict, ready to be used in a call to instantiate item views using the item view *cls* or *template*. In the case of *cls*, the args dict acts as a kwargs object. For a *template*, it is treated as a context (ctx) but is essentially similar in form to the kwargs usage.
- *selection\_mode*: a string with the value 'single', 'multiple' or others (See [selection\\_mode](#) for details).
- *allow\_empty\_selection*: a boolean, which if False (the default), forces there to always be a selection if there is data available. If True, selection happens only as a result of user action.

In narrative, we can summarize as follows:

A listview's adapter takes data items and uses an `args_converter` function to transform them into arguments for making list item view instances, using either a `cls` or a `kv` template.

In a graphic, a summary of the relationship between a listview and its list adapter, looks like this:



A `DictAdapter` has the same arguments and requirements as `ListAdapter` except for two things:

1. There is an additional argument, `sorted_keys`, which must meet the requirements of normal python dictionary keys.
2. The data argument is, as you would expect, a dict. Keys in the dict must include the keys in the `sorted_keys` argument, but they may form a superset of the keys in `sorted_keys`. Values may be strings, class instances, dicts, etc. (The `args_converter` uses it accordingly).

### 126.23.5 Using an Args Converter

A `ListView` allows use of built-in list item views, such as `ListItemButton`, your own custom item view class or a custom `kv` template. Whichever type of list item view is used, an `args_converter` function is needed to prepare, per list data item, args for the `cls` or template.

---

**Note:** Only the `ListItemLabel`, `ListItemButton` or custom classes like them (and not the simple `Label` or `Button` classes) are to be used in the listview system.

---

**Warning:** `ListItemButton` inherits the `background_normal` and `background_down` properties from the `Button` widget, so the `selected_color` and `deselected_color` are not represented faithfully by default.

Here is an `args_converter` for use with the built-in `ListItemButton` specified as a normal Python function:

```
def args_converter(row_index, an_obj):
    return {'text': an_obj.text,
            'size_hint_y': None,
            'height': 25}
```

and as a lambda:

```
args_converter = lambda row_index, an_obj: {'text': an_obj.text, 'size_hint_y': None,
                                             'height': 25}
```

In the args converter example above, the data item is assumed to be an object (class instance), hence the reference `an_obj.text`.

Here is an example of an args converter that works with list data items that are dicts:

```
args_converter = lambda row_index, obj: {'text': obj['text'],
                                         'size_hint_y': None,
                                         'height': 25}
```

So, it is the responsibility of the developer to code the `args_converter` according to the data at hand. The `row_index` argument can be useful in some cases, such as when custom labels are needed.

### 126.23.6 An Example ListView

Now, to some example code:

```
from kivy.adapters.listadapter import ListAdapter
from kivy.uix.listview import ListItemButton, ListView

data = [{'text': str(i), 'is_selected': False} for i in range(100)]

args_converter = lambda row_index, rec: {'text': rec['text'],
                                          'size_hint_y': None,
                                          'height': 25}

list_adapter = ListAdapter(data=data,
                           args_converter=args_converter,
                           cls=ListItemButton,
                           selection_mode='single',
                           allow_empty_selection=False)

list_view = ListView(adapter=list_adapter)
```

This listview will show 100 buttons with text of 0 to 100. The args converter function works on dict items in the data. `ListItemButton` views will be instantiated from the args converted by `args_converter` for each data item. The listview will only allow single selection: additional touches will be ignored. When the listview is first shown, the first item will already be selected because `allow_empty_selection` is `False`.

The `ListItemLabel` works in much the same way as the `ListItemButton`.

### 126.23.7 Using a Custom Item View Class

The data used in an adapter can be any of the normal Python types, such as strings, class instances and dictionaries. They can also be custom classes, as shown below. It is up to the programmer to assure that the `args_converter` performs the appropriate conversions.

Here we make a simple `DataItem` class that has the required `text` and `is_selected` properties:

```
from kivy.uix.listview import ListItemButton
from kivy.adapters.listadapter import ListAdapter

class DataItem(object):
    def __init__(self, text='', is_selected=False):
        self.text = text
        self.is_selected = is_selected

data_items = []
data_items.append(DataItem(text='cat'))
data_items.append(DataItem(text='dog'))
data_items.append(DataItem(text='frog'))

list_item_args_converter = lambda row_index, obj: {'text': obj.text,
                                                  'size_hint_y': None,
                                                  'height': 25}
```

```
list_adapter = ListAdapter(data=data_items,
                           args_converter=list_item_args_converter,
                           selection_mode='single',
                           propagate_selection_to_data=True,
                           allow_empty_selection=False,
                           cls=ListItemButton)

list_view = ListView(adapter=list_adapter)
```

The data is set in a `ListAdapter` along with a list item `args_converter` function above (lambda) and arguments concerning selection: only single selection is allowed, and selection in the listview will propagate to the data items. The propagation setting means that the `is_selected` property for each data item will be set and kept in sync with the list item views. By having `allow_empty_selection=False`, when the listview first appears, the first item, 'cat', will already be selected. The list adapter will instantiate a `ListItemButton` class instance for each data item, using the assigned `args_converter`.

The `list_view` would be added to a view with `add_widget()` after the last line, where it is created. See the basic example at the top of this documentation for an example of `add_widget()` use in the context of a sample app.

You may also use the provided `SelectableDataItem` mixin to make a custom class. Instead of the "manually-constructed" `DataItem` class above, we could do:

```
from kivy.adapters.models import SelectableDataItem

class DataItem(SelectableDataItem):
    # Add properties here.
    pass
```

`SelectableDataItem` is a simple mixin class that has an `is_selected` property.

## 126.23.8 Using an Item View Template

`SelectableView` is another simple mixin class that has required properties for a list item: `text`, and `is_selected`. To make your own template, mix it in as follows:

```
from kivy.uix.listview import ListItemButton
from kivy.uix.listview import SelectableView

Builder.load_string("""
[CustomListItem@SelectableView+BoxLayout]:
    size_hint_y: ctx.size_hint_y
    height: ctx.height
    ListItemButton:
        text: ctx.text
        is_selected: ctx.is_selected
""")
```

A class called `CustomListItem` will be instantiated for each list item. Note that it is a `BoxLayout` and is thus a kind of container. It contains a `ListItemButton` instance.

Using the power of the Kivy language (kv), you can easily build composite list items – in addition to `ListItemButton`, you could have a `ListItemLabel`, or a custom class you have defined and registered with the system.

An `args_converter` needs to be constructed that goes along with such a kv template. For example, to use the kv template above:

```

list_item_args_converter = \
    lambda row_index, rec: {'text': rec['text'],
                            'is_selected': rec['is_selected'],
                            'size_hint_y': None,
                            'height': 25}

integers_dict = \
    { str(i): {'text': str(i), 'is_selected': False} for i in range(100)}

dict_adapter = DictAdapter(sorted_keys=[str(i) for i in range(100)],
                           data=integers_dict,
                           args_converter=list_item_args_converter,
                           template='CustomListItem')

list_view = ListView(adapter=dict_adapter)

```

A dict adapter is created with 1..100 integer strings as sorted\_keys, and an integers\_dict as data. integers\_dict has the integer strings as keys and dicts with text and is\_selected properties. The CustomListItem defined above in the Builder.load\_string() call is set as the kv template for the list item views. The list\_item\_args\_converter lambda function will take each dict in integers\_dict and will return an args dict, ready for passing as the context (ctx) for the template.

The list\_view would be added to a view with add\_widget() after the last line, where it is created. Again, see the basic example above for add\_widget() use.

### 126.23.9 Using CompositeListItem

The class `CompositeListItem` is another option for building advanced composite list items. The kv language approach has its advantages, but here we build a composite list view using a straight Kivy widget method:

```

args_converter = lambda row_index, rec:
    {'text': rec['text'],
     'size_hint_y': None,
     'height': 25,
     'cls_dicts': [{'cls': ListItemButton,
                     'kwargs': {'text': rec['text']}},
                   {'cls': ListItemLabel,
                     'kwargs': {'text': "Middle-{}".format(rec['text']),
                                'is_representing_cls': True}},
                   {'cls': ListItemButton,
                     'kwargs': {'text': rec['text']}}]}

item_strings = ["{0}".format(index) for index in range(100)]

integers_dict =
    { str(i): {'text': str(i), 'is_selected': False} for i in range(100)}

dict_adapter = DictAdapter(sorted_keys=item_strings,
                           data=integers_dict,
                           args_converter=args_converter,
                           selection_mode='single',
                           allow_empty_selection=False,
                           cls=CompositeListItem)

list_view = ListView(adapter=dict_adapter)

```

The args\_converter is somewhat complicated, so we should go through the details. Observe in the `DictAdapter` instantiation that `CompositeListItem` instance is set as the cls to be instantiated for each list item. The args\_converter will make args dicts for this cls. In the args\_converter, the first

three items, `text`, `size_hint_y`, and `height`, are arguments for `CompositeListItem` itself. After that you see a `cls_dicts` list that contains argument sets for each of the member widgets for this composite: `ListIconButton` and `ListItemLabel`. This is a similar approach to using a kv template described above.

The `sorted_keys` and `data` arguments for the dict adapter are the same as in the previous code example. For details on how `CompositeListItem` works, examine the code, looking for how parsing of the `cls_dicts` list and kwargs processing is done.

### 126.23.10 Uses for Selection

What can we do with selection? Combining selection with the system of bindings in Kivy, we can build a wide range of user interface designs.

We could make data items that contain the names of dog breeds, and connect the selection of dog breed to the display of details in another view, which would update automatically on selection. This is done via a binding to the `on_selection_change` event:

```
list_adapter.bind(on_selection_change=callback_function)
```

where `callback_function()` does whatever is needed for the update. See the example called `list_master_detail.py`, and imagine that the list on the left would be a list of dog breeds, and the detail view on the right would show details for a selected dog breed.

In another example, we could set the `selection_mode` of a listview to 'multiple', and load it with a list of answers to a multiple-choice question. The question could have several correct answers. A color swatch view could be bound to selection change, as above, so that it turns green as soon as the correct choices are made, unless the number of touches exceeds a limit, when the answer session would be terminated. See the examples that feature thumbnail images to get some ideas, e.g., `list_cascade_dict.py`.

In a more involved example, we could chain together three listviews, where selection in the first controls the items shown in the second, and selection in the second controls the items shown in the third. If `allow_empty_selection` were set to `False` for these listviews, a dynamic system of selection "cascading" from one list to the next, would result.

There are so many ways that listviews and Kivy bindings functionality can be used, that we have only scratched the surface here. For on-disk examples, see these:

```
kivy/examples/widgets/lists/list_*.py
```

Several examples show the "cascading" behavior described above. Others demonstrate the use of kv templates and composite list views.

```
class kivy.uix.listview.SelectableView(**kwargs)
    Bases: object
```

The `SelectableView` mixin is used to design list items and other classes that are to be instantiated by an adapter for use in a listview. The `ListAdapter` and `DictAdapter` adapters are selection-enabled. `select()` and `deselect()` are to be overridden with display code to mark items as selected or not, if desired.

```
deselect(*args)
```

The list item is responsible for updating the display for being unselected, if desired.

```
index
```

The index into the underlying data list or the data item this view represents.

`index` is a `NumericProperty`, default to -1.

### **is\_selected**

A `SelectableView` instance carries this property, which should be kept in sync with the equivalent property in the data item it represents.

`is_selected` is a `BooleanProperty`, default to `False`.

### **select(\*args)**

The list item is responsible for updating the display for being selected, if desired.

**class** `kivy.uix.listview.ListItemButton(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.button.Button`

`ListItemButton` mixes `SelectableView` with `Button` to produce a button suitable for use in `ListView`.

### **deselected\_color**

`deselected_color` is a `ListProperty` and defaults to `[0., 1., 0., 1]`.

### **selected\_color**

`selected_color` is a `ListProperty` and defaults to `[1., 0., 0., 1]`.

**class** `kivy.uix.listview.ListItemLabel(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.label.Label`

`ListItemLabel` mixes `SelectableView` with `Label` to produce a label suitable for use in `ListView`.

**class** `kivy.uix.listview.CompositeListItem(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.boxlayout.BoxLayout`

`CompositeListItem` mixes `SelectableView` with `BoxLayout` for a generic container-style list item, to be used in `ListView`.

### **background\_color**

`ListItem` subclasses `Button`, which has `background_color`, but for a composite list item, we must add this property.

`background_color` is a `ListProperty` and defaults to `[1, 1, 1, 1]`.

### **deselected\_color**

`deselected_color` is a `ListProperty` and defaults to  `[.33, .33, .33, 1]`.

### **representing\_cls**

Which component view class, if any, should represent for the composite list item in `__repr__()`?

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

### **selected\_color**

`selected_color` is a `ListProperty` and defaults to `[1., 0., 0., 1]`.

**class** `kivy.uix.listview.ListView(**kwargs)`

Bases: `kivy.uix.abstractview.AbstractView`, `kivy.event.EventDispatcher`

`ListView` is a primary high-level widget, handling the common task of presenting items in a scrolling list. Flexibility is afforded by use of a variety of adapters to interface with data.

The adapter property comes via the mixed in `AbstractView` class.

`ListView` also subclasses `EventDispatcher` for scrolling. The event `on_scroll_complete` is used in refreshing the main view.

For a simple list of string items, without selection, use `SimpleListAdapter`. For list items that respond to selection, ranging from simple items to advanced composites, use `ListAdapter`. For an alternate powerful adapter, use `DictAdapter`, rounding out the choice for designing highly interactive lists.



## Events

**`on_scroll_complete: (boolean, )`**Fired when scrolling completes.

## container

The container is a `GridLayout` widget held within a `ScrollView` widget. (See the associated kv block in the `Builder.load_string()` setup). Item view instances managed and provided by the adapter are added to this container. The container is cleared with a call to `clear_widgets()` when the list is rebuilt by the `populate()` method. A padding `Widget` instance is also added as needed, depending on the row height calculations.

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

## divider

[TODO] Not used.

## divider\_height

[TODO] Not used.

## item\_strings

If `item_strings` is provided, create an instance of `SimpleListAdapter` with this list of strings, and use it to manage a no-selection list.

`item_strings` is a `ListProperty` and defaults to `[]`.

## row\_height

The `row_height` property is calculated on the basis of the height of the container and the count of items.

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

## scrolling

If the `scroll_to()` method is called while scrolling operations are happening, a call recursion error can occur. `scroll_to()` checks to see that scrolling is `False` before calling `populate()`. `scroll_to()` dispatches a `scrolling_complete` event, which sets scrolling back to `False`.

`scrolling` is a `BooleanProperty` and defaults to `False`.

## 126.24 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.

### 126.24.1 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)

# bind the on_press event of the button to the dismiss function
content.bind(on_press=view.dismiss)

# open the view
view.open()
```

## 126.24.2 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_open`: Fired when the `ModalView` is opened.

`on_dismiss`: Fired when the `ModalView` is closed. If the callback returns `True`, the dismiss will be canceled.

### 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).

`background_color` is a `ListProperty` and defaults to `[0, 0, 0, .7]`.

### **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: (top, right, bottom, 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)
```

### **open(\*largs)**

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`.

## 126.25 PageLayout

The `PageLayout` class allow to create a simple multiple page layout, in a way that allows easy flipping of one page to another using borders.

`PageLayout` doesn't honor `size_hint` or `pos_hint` in any way currently.

New in version 1.8.0.

example:

```
PageLayout:
    Button:
        text: 'page1'

    Button:
        text: 'page2'

    Button:
        text: 'page3'
```

`class kivy.uix.pagelayout.PageLayout (**kwargs)`

Bases: `kivy.uix.layout.Layout`

PageLayout class. See module documentation for more information

**border**

Width of the border used around current page to display previous/next page when needed.

`border` is a `NumericProperty`, default to 0.

**page**

Currently displayed page.

`page` is a `NumericProperty`, default to 0.

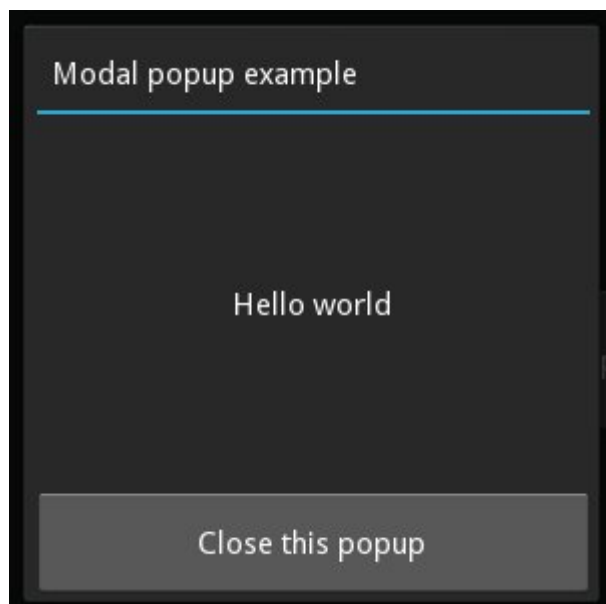
**swipe\_threshold**

Thresold to the swipe action triggering, as percentage of the widget size.

`swipe_threshold` is a `NumericProperty`, default to .5.

## 126.26 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.

### 126.26.1 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 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()
```

## 126.26.2 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.

### content

Content of the popup that is displayed just under the title.

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

### 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\_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\_size**

Represents the font size of the popup title.

New in version 1.6.0.

`title_size` is a `NumericProperty` and defaults to '14sp'.

## **class kivy.uix.popup.PopupException**

Bases: `exceptions.Exception`

Popup exception, fired when multiple content widgets are added to the popup.

New in version 1.4.0.

## 126.27 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` than returns the value of the progressbar. If the value is  $< 0$  or  $> \text{max}$ , 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
>>> slider.value_normalized
0.5
```

`value_normalized` is an `AliasProperty`.

## 126.28 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.

For example, if you create a `RelativeLayout`, add a widget with `position = (0,0)`, the child widget will also move when you change the position of the `RelativeLayout`. The child widgets coordinates remain (0,0) i.e. they are always relative to the containing layout.

### 126.28.1 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.

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.

## 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.

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.
```

## 126.29 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.

```
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.

## 126.30 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 similar to a `RelativeLayout` (see `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 it's 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()`.

### 126.30.1 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)
```

## 126.30.2 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)
```

## 126.30.3 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.

## 126.30.4 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.)
```

## 126.30.5 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 1.8.1: 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.applly_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**.

### **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.

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 1.8.1.

**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.

**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.

**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.

## 126.31 Scatter Layout

New in version 1.6.0.

This layout behaves just like a **RelativeLayout**. For example, if you create a **ScatterLayout**, add a widget with position = (0,0), the child widget will also move when you change the position of the **ScatterLayout**. The child widget's coordinates remain (0,0), i.e. they are relative to the containing layout.

However, since `ScatterLayout` is implemented using a `Scatter` widget, you can also translate, rotate and scale the layout using touches (mouse or fingers) just like 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
    RelativeLayout class, see module documentation for more information.
```

## 126.32 Screen Manager

New in version 1.4.0.

**Warning:** This widget is still experimental, and its API is subject to change in a future version.

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.

### 126.32.1 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'
```

From 1.8.0, you can now switch dynamically to a new screen, change the transition options and remove the previous one by using `ScreenManager.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')
```

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.

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'
            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()

```

## 126.32.2 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
- **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(**kwargs)
```

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.

### **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\_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.

### **previous()**

Return the name of the previous screen from the screen list.

### **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 must not change the list manually. Use `Screen.add_widget()` instead.

`screens` is a `ListProperty` and defaults to `[]`, read-only.

## switch\_to(screen, \*\*options)

Add a new screen to the `ScreenManager` and switch to it. The previous screen will be removed from the children. `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 screen that will be hidden and the screen that will be shown. By default, an instance of `SlideTransition` will be given.

For example, if you want to change to a `WipeTransition`:

```
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'
```

Changed in version 1.8.0: Default transition has been changed from `SwapTransition` to `SlideTransition`.

## class kivy.uix.screenmanager.ScreenManagerException

Bases: `exceptions.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)
```

### **clearcolor**

Sets the color of Fbo ClearColor.

New in version 1.8.1.

`clearcolor` is a `ListProperty` and defaults to `[0, 0, 0, 1]`.

#### **fs**

Fragment shader to use.

`fs` is a `StringProperty` and defaults to `None`.

#### **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**

Bases: `kivy.uix.screenmanager.TransitionBase`

Swap transition that looks like iOS transition when a new window appears on the screen.

### **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.

## 126.33 Scroll View

New in version 1.0.4.

The `ScrollView` widget provides a scrollable/pannable viewport that is clipped at the scrollview's bounding box.

### 126.33.1 Scrolling Behavior

The `ScrollView` accepts only one child and applies a viewport/window to it according to the `ScrollView.scroll_x` and `ScrollView.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:

- **`ScrollView.scroll_distance`**: the minimum distance to travel, defaults to 20 pixels.
- **`ScrollView.scroll_timeout`**: the maximum time period, defaults to 250 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.

### 126.33.2 Limiting to the X or Y Axis

By default, the `ScrollView` allows scrolling in both the X and Y axes. You can explicitly disable scrolling on an axis by setting `ScrollView.do_scroll_x` or `ScrollView.do_scroll_y` to `False`.

### 126.33.3 Managing the Content Size and Position

`ScrollView` manages the position of its children similarly to a `RelativeLayout` (see `relativelayout`) but not the size. You must carefully specify the `size_hint` of your content to get the desired scroll/pan effect.

By default, `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.

To scroll a `GridLayout` on Y-axis/vertically, set the child's width identical to that of the `ScrollView` (`size_hint_x=1`, default), and set the `size_hint_y` property to `None`:

```

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(30):
    btn = Button(text=str(i), size_hint_y=None, height=40)
    layout.add_widget(btn)
root = ScrollView(size_hint=(None, None), size=(400, 400))
root.add_widget(layout)

```

## 126.33.4 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 `ScrollView.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 `ScrollView.effect_x` and/or `ScrollView.effect_y` to an instance of the effect you want to use. This will override the default effect set in `ScrollView.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.

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

### **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 1.8.1.

`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`, default 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`, default 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.

### **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\_type**

Sets the type of scrolling to use for the content of the scrollview. Available options are: ['content'], ['bars'], ['bars', 'content'].

New in version 1.8.0.

`scroll_type` is a `OptionProperty`, 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.



### **update\_from\_scroll** (\*larges)

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.

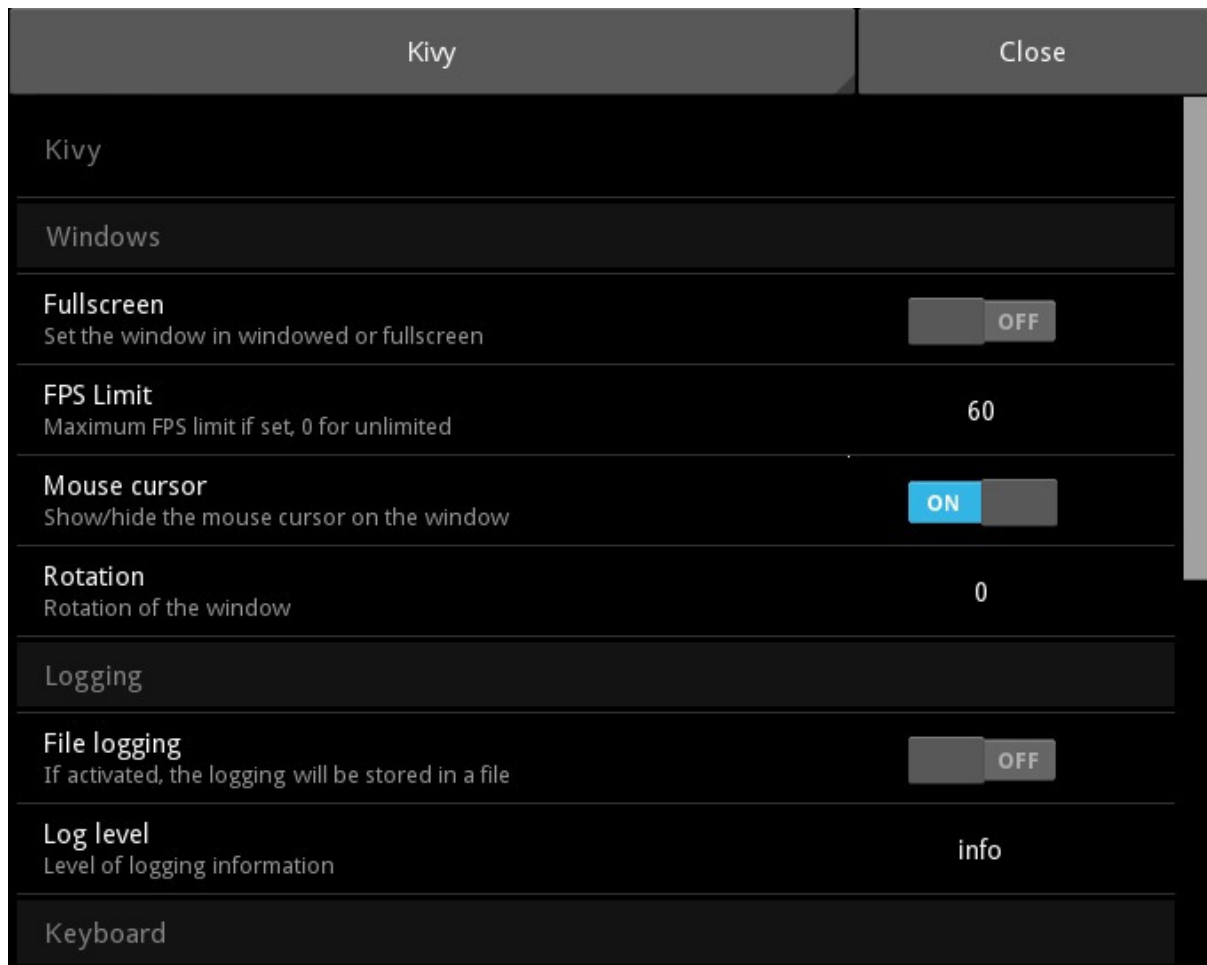
### **viewport\_size**

(internal) Size of the internal viewport. This is the size of your only child in the scrollview.

## 126.34 Settings

New in version 1.0.7.

This module is 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 with the `App` class. Use `Settings.add_kivy_panel()` to configure the Kivy core settings in a panel.

### 126.34.1 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 default values for all 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",
    "true": "auto"
  }
]
```

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	<code>SettingTitle</code>
bool	<code>SettingBoolean</code>
numeric	<code>SettingNumeric</code>
options	<code>SettingOptions</code>
string	<code>SettingString</code>
path	<code>SettingPath</code> (new from 1.1.0)

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 `SettingTitle.title` property 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...

```

## 126.34.2 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.

**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 section/key/value 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`.

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 `:class'InterfaceWithSidebar'`.

Changed in version 1.8.0: If you set a string, the `Factory` will be used to resolve the class.

### **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.

**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`.

**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.

#### popup

(internal) Used to store the current popup when it is 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.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** – A `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** – A `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.

### **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.

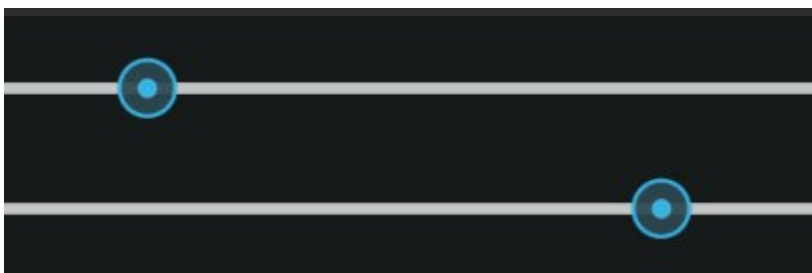
**Parameters**`uid` – A panel uid. It should be used to retrieve and display a settings panel that has previously been added with `add_panel()`.

### **panels**

(internal) Stores a dictionary mapping settings panels to their uids.

`panels` is a `DictProperty` and defaults to `{}`.

## 126.35 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')
```

```
class kivy.uix.slider.Slider(**kwargs)
```

Bases: `kivy.uix.widget.Widget`

Class for creating a Slider widget.

Check module documentation for more details.

#### **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.

#### **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 10. The range of the slider is reduced from padding \*2 on the screen. It allows drawing a cursor of 20px width without having the cursor go out of the widget.

`padding` is a `NumericProperty` and defaults to 10.

#### **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`).

#### **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
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**.

## 126.36 Spinner

New in version 1.4.0.



Spinner is a widget that provide a quick way to select one value from a set. In the default state, a spinner show its currently selected value. Touching the spinner displays a dropdown menu with all

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 showed
    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, 'have text', text)

spinner.bind(text=show_selected_value)

runTouchApp(spinner)
```

```
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 you set a string, the `Factory` will be used to resolve the class.

#### 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 at least:

- a `text` property, used to display the value.
- an `on_release` event, used to trigger the option when pressed/touched.

`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.

#### values

Values that can be selected by the user. It must be a list of strings.

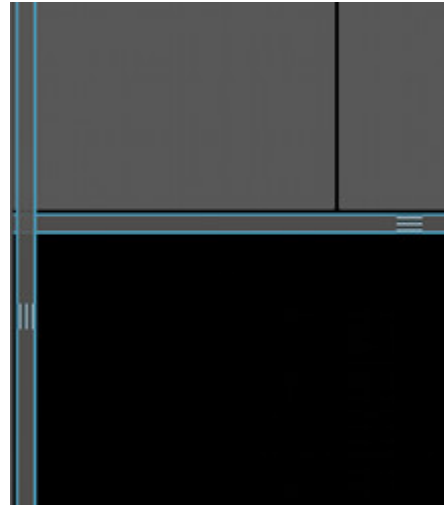
`values` is a `ListProperty` and defaults to [].

```
class kivy.uix.spinner.SpinnerOption(**kwargs)
    Bases: kivy.uix.button.Button
```

Special button used in the dropdown list. We just set the default `size_hint_y` and height.

## 126.37 Splitter

New in version 1.5.0.



The **Splitter** is a widget that helps you re-size it's 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 it's 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
    background_down: 'path to pressed horizontal image' if self.horizontal else 'path to vertical
```

```
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.

**border**

Border used for the `BorderImage` graphics instruction.

This must be a list of four values: (top, right, bottom, left). Read the `BorderImage` instructions for more information about how to use it.

`border` is a `ListProperty` and defaults to (4, 4, 4, 4).

**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 1.8.1.

**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`.

**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.

`keep_within_parent` is a `BooleanProperty` and defaults to False.

New in version 1.8.1.

**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`.

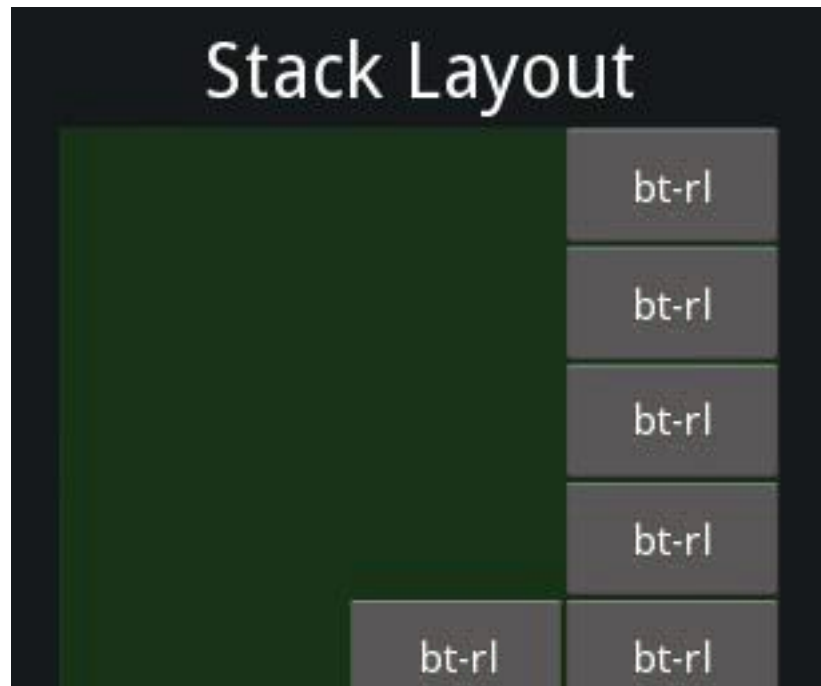
Changed in version 1.8.0: If you set a string, the `Factory` will be used to resolve the class.

**strip\_size**

Specifies the size of resize strip

`strip_size` is a `NumericProperty` defaults to `10pt`

## 126.38 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.

### **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].

## 126.39 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 8 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 their 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.

## 126.40 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.

---



```
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.

#### **touch\_control**

(internal) Contains the touch that currently interacts with the switch.

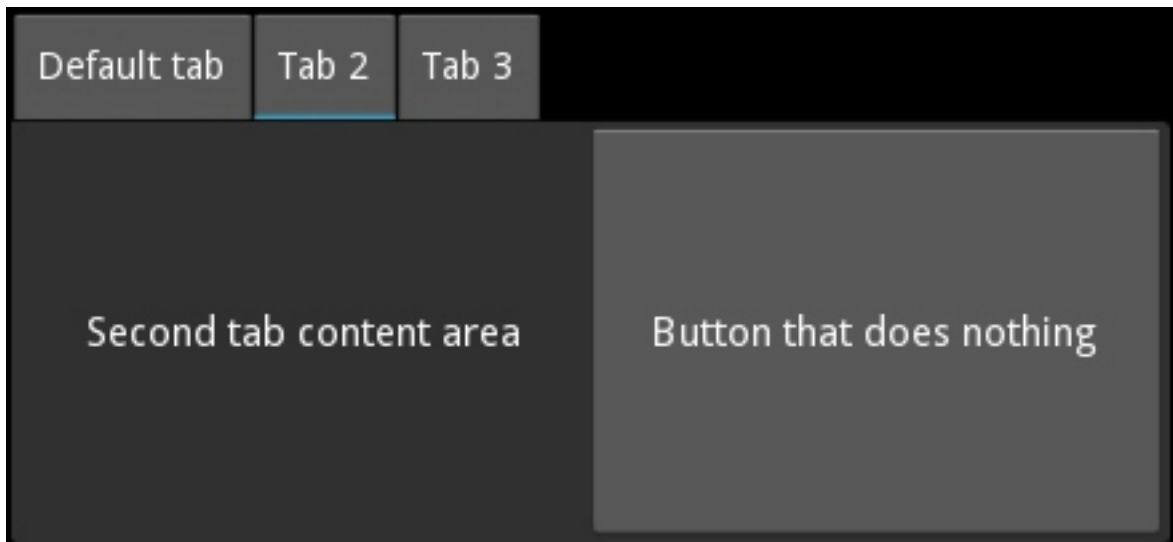
`touch_control` is an `ObjectProperty` and defaults to None.

#### **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.

## 126.41 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.

### 126.41.1 Simple example

```
'''
TabbedPanel
=====
```

```

Test of the widget TabbedPanel.
'''

from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.floatlayout import FloatLayout
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\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.

---

## 126.41.2 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.

---

**Note:** The `default_tab` functionality is turned off by default since 1.5.0. To turn it back on, set `do_default_tab = True`.

---

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.

#### **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: (top, right, bottom, left). Read the `BorderImage` instructions for more information.

`border` is a `ListProperty` and defaults to (16, 16, 16, 16)

#### **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'.

#### **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*)

Switch to a specific panel header.

### **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 'bottom\_mid'.

### **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.

`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.

```
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`.

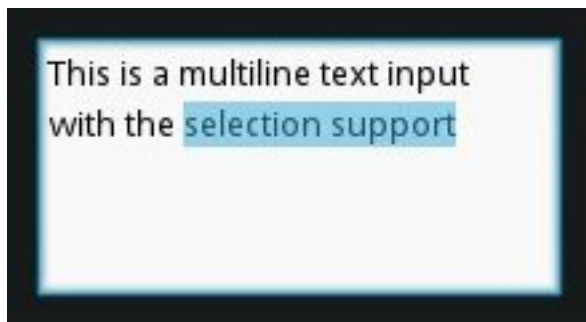
```
class kivy.uix.tabbedpanel.TabbedPanelException
```

Bases: `exceptions.Exception`

The TabbedPanelException class.

## 126.42 Text Input

New in version 1.0.4.



The `TextInput` widget provides a box of editable plain text.

Unicode, multiline, cursor navigation, selection and clipboard features are supported.

---

**Note:** Two different coordinate systems are used with `TextInput`:

- (x, y) - coordinates in pixels, mostly used for rendering on screen.
  - (row, col) - cursor index in characters / lines, used for selection and cursor movement.
- 

### 126.42.1 Usage example

To create a multiline textinput ('enter' key adds a new line):

```
from kivy.uix.textinput import TextInput
textinput = TextInput(text='Hello world')
```

To create a singleline textinput, set the multiline property to `False` ('enter' key will defocus the textinput and emit `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 on 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 'focus' 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)
```

## 126.42.2 Selection

The selection is automatically updated when the cursor position changes. You can get the currently selected text from the `TextInput.selection_text` property.

## 126.42.3 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 to 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)

```

#### 126.42.4 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, Right
Control + c	Copy selection
Control + x	Cut selection
Control + p	Paste selection
Control + a	Select all the content
Control + z	undo
Control + r	redo

```

class kivy.uix.textinput.TextInput(**kwargs)
    Bases: 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\(\)](#).

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\_active**

Background image of the TextInput when it's in focus and disabled.

New in version 1.8.0.

**background\_disabled\_active** is a **StringProperty** and defaults to 'atlas://data/images/defaulttheme/textinput\_disabled\_active'.

### **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'.

### **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: (top, right, bottom, 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 blink the cursor graphic. The value of **cursor\_blink** is automatically computed. Setting a value on it will have no impact.

**cursor\_blink** is a **BooleanProperty** and defaults to False.

**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 1.8.1.

**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.

**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*)

Move the cursor relative to it's 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

**Warning:** Current page has three lines before/after.

### **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.

### **focus**

If `focus` is `True`, the keyboard will be requested and you can start entering text into the `textInput`.

`focus` is a `BooleanProperty` and defaults to `False`.

---

**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`).

---

### **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 `'DroidSans'`.

**font\_size**

Font size of the text in pixels.

`font_size` is a `NumericProperty` and defaults to 10.

**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.

**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'.

**hide\_keyboard()**

Convenience function to hide the keyboard in managed mode

**hint\_text**

Hint text of the widget.

Shown if text is "" and focus is False.

New in version 1.6.0.

`hint_text` a `StringProperty` 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 1.8.1.

`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 parameter;

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.

### **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', 'address'.

### **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\_mode**

How the keyboard visibility should be managed (auto will have standard behaviour to show/hide on focus, managed requires setting `keyboard_visible` manually, or calling the helper functions `show_keyboard()` and `hide_keyboard()`).

New in version 1.8.0.

**keyboard\_mode** is an **OptionsProperty** and defaults to 'auto'. Can be one of 'auto' or 'managed'.

### **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** 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**.

### **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\_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 '\*'.

New in version 1.2.0.

`password` is a `BooleanProperty` and defaults to False.

### **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.

### **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`.

### **show\_keyboard()**

Convenience function to show the keyboard in managed mode

### **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` a `StringProperty`.

#### **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.

## 126.43 Toggle button

The `ToggleButton` widget acts like a checkbox. When you touch/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.ToggleButtonBehavior`, `kivy.uix.button.Button`

Toggle button class, see module documentation for more information.

## 126.44 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.

### 126.44.1 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)

        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'))
```

## 126.44.2 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*.

`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 `kivy.properties.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 `kivy.properties.NumericProperty` and defaults to 0.

### 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).

### **class kivy.uix.treeview.TreeViewException**

Bases: `exceptions.Exception`

Exception for errors in the `TreeView`.

### **class kivy.uix.treeview.TreeViewLabel(\*\*kwargs)**

Bases: `kivy.uix.label.Label`, `kivy.uix.treeview.TreeViewNode`

Combines a `Label` and a `TreeViewNode` 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.TreeViewNode(\*\*kwargs)**

Bases: `object`

`TreeViewNode` 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`.

## 126.45 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.

### 126.45.1 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).

### 126.45.2 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 1.8.1) mode. Keys for this row data must be named `normal_<row>`, `shift_<row>` and `spe-`

*cial\_<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  
["\u21B9", "      ", "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],      ["\u232b", null, "backspace", 2]  
  ],  
  "shift_1": [ ... ],  
  "normal_2": [ ... ],  
  "special_2": [ ... ],  
  ...  
}
```

### 126.45.3 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:  
    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** a **StringProperty** and defaults to `atlas://data/images/defaulttheme/vkeyboard`

### **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 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 `[kiivy]` section.

**docked** is a **BooleanProperty** and defaults to False.

### **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** 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** 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.

..versionadded:: 1.8.0

`key_disabled_background_normal` 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 be loaded and added to the `available_layouts`.

`layout` is a `StringProperty` and defaults to None.

### layout\_path

Path from which layouts are read.

`layout_path` 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]

### 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 find 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.

## 126.46 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*)

Change the position to a percentage of duration. Percentage must be a value between 0-1.

**Warning:** Calling `seek()` before the video is loaded has no impact.

New in version 1.2.0.

### **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 'play'.

### **unload** ()

Unload the video. The playback will be stopped.

New in version 1.8.0.

### **volume**

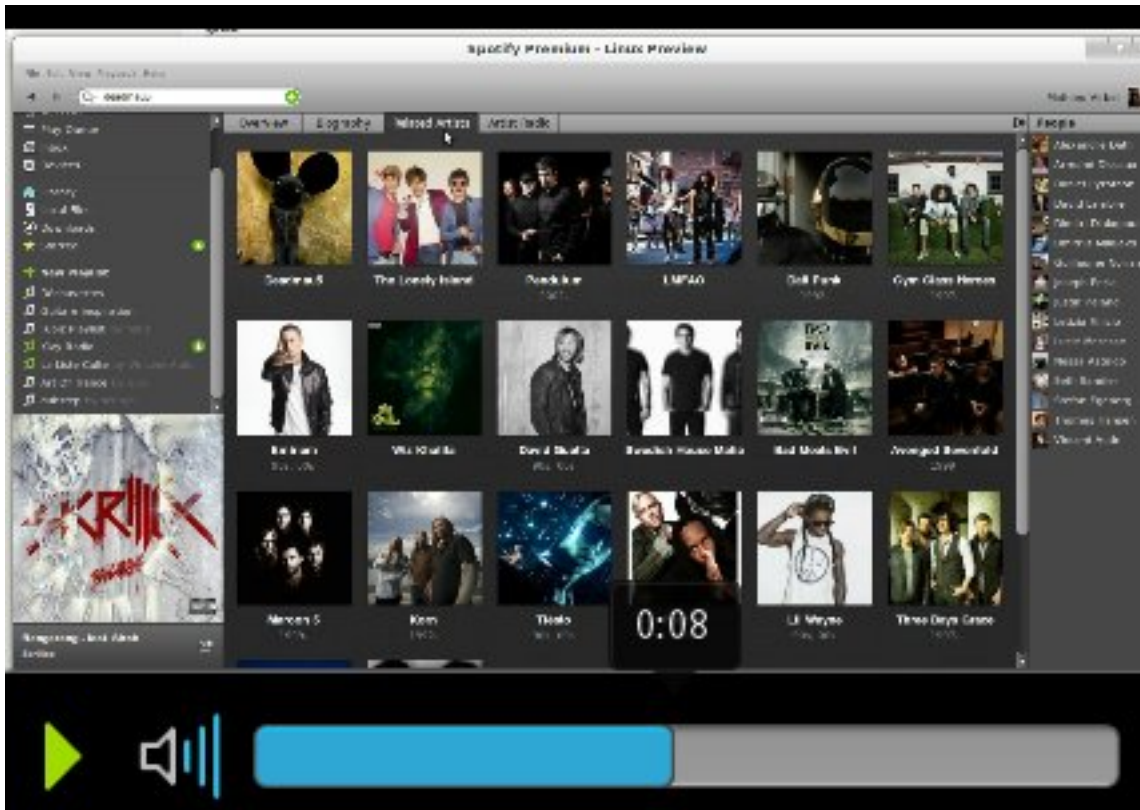
Volume of the video, in the range 0-1. 1 means full volume, 0 means mute.

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

## 126.47 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.



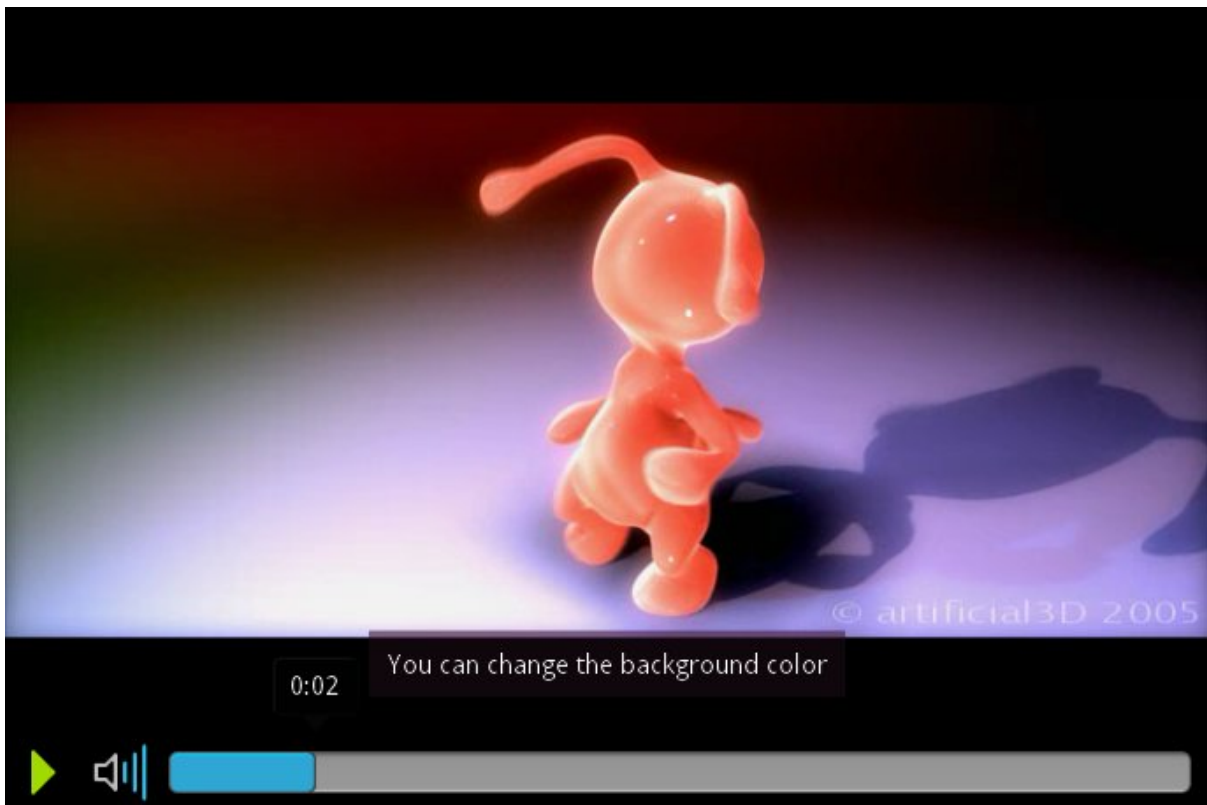
## 126.47.1 Annotations

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/softboy.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 `softboy.avi` example, the result will be:



If you want to experiment with annotation files, test with:

```
python -m kivy.uix.videoplayer examples/widgets/softboy.avi
```

### 126.47.2 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})
```

### 126.47.3 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 it's 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'`.

#### **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*)

Change the position to a percentage of the duration. Percentage must be a value between 0-1.

**Warning:** Calling `seek()` before video is loaded has no effect.

#### **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 'play'.

### 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
- `bgsouce`: '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.

## 126.48 Widget class

The `Widget` class is the base class required to create a Widget. This widget class is 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.

**Separate 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 want to only trigger an action when the button itself is actually touched. For this, you can use the

`Widget.collide_point()` method, which will return True if the point you pass 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 `Widget.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 size.
- `Widget.on_touch_down()`, `Widget.on_touch_move()`, `Widget.on_touch_up()` don't do any sort of collisions. If you want to know if the touch is inside your widget, use `Widget.collide_point()`.

## 126.48.1 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)

wid = Widget()
wid.bind(pos=callback_pos)
```

Read more about *Properties*.

## 126.48.2 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.

### 126.48.3 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 most recently added widget and then backwards through its children (from the most recently added back to the first child). This order is the same for the `on_touch_move` and `on_touch_up` events.

If you want to reverse this order, you can raise events in the children before the parent by using the `super` command. For example:

```

class MyWidget(Widget):
    def on_touch_down(self, touch):
        super(MyWidget, self).on_touch_down(touch)
        # Do stuff here

```

In general, this would seldom be the best approach as every event bubbles all the way through event time and there is no way of determining if it has been handled. In order to stop this the event bubbling, one of these methods must return `True`. At this point, Kivy assumes the event has been handled and propagation stops.

This means that the recommended approach is to let the event bubble naturally but swallow the event if it has been handled. For example:

```

class MyWidget(Widget):
    def on_touch_down(self, touch):
        if <some_condition>:
            # Do stuff here and kill the event
            return True
        else:
            # Continue normal event bubbling
            return super(MyWidget, self).on_touch_down(touch)

```

This approach gives you good control over exactly how events are dispatched and managed.

```

class kivy.uix.widget.Widget(**kwargs)
    Bases: kivy.uix.widget.WidgetBase

```

Widget class. See module documentation for more information.

#### Events

`on_touch_down`: Fired when a new touch event occurs  
`on_touch_move`: Fired when an existing touch moves  
`on_touch_up`: Fired when an existing touch disappears

**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*)

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

New in version 1.0.5.

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

**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`)

**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 widgets added to this widget.

Changed in version 1.8.0: `children` argument can be used to select the children we want to remove. It should be a list of children (or filtered list) of the current widget.

## 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 boundingbox.

### Parameters

**x:** numeric X position of the point (in window coordinates)

**y:** numeric Y position of the point (in window coordinates)

**Returns** bool, True if the point is inside the bounding box.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

## collide\_widget(wid)

Check if the other widget collides with this widget. Performs an axis-aligned bounding box intersection test by default.

### Parameters

**wid:** **Widget** class Widget to collide with.

**Returns** bool, True if the other widget collides with this widget.

```
>>> 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.

---

### 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.

**disabled** is a **BooleanProperty** and defaults to False.

## export\_to\_png(filename, \*args)

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 1.8.1.

## 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**.

## 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

Unique identifier of the widget in the tree.

`id` is a `StringProperty` and defaults to None.

**Warning:** If the `id` is already used in the tree, an exception will be raised.

## ids

This is a Dictionary of id's defined in your kv language. This will only be populated if you use id's in your kv language code.

New in version 1.7.0.

`ids` is a `DictProperty` and defaults to a 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:
        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 [RelativeLayout](#) for a discussion on coordinate systems.

**Returns** bool. If True, the dispatching of the touch event will stop.

### **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 the 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.

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

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.

### **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, in percent (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.

Position hint is only used by the `FloatLayout` and `Window`.

`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`).

See `size_hint_x` for more information

### **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 value is in percent as a float from 0. to 1., where 1. means the full size of his parent. 0.5 represents 50%.

`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

### **to\_local** (*x*, *y*, *relative=False*)

Transform parent coordinates to local 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 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 given coordinate from window to local widget coordinates. See [relativelayout](#) for details on the coordinate systems.

**to\_window**(*x, y, initial=True, relative=False*)

Transform local 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 1.8.1.

**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
  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 1.8.1.

### 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.

**class** `kivy.uix.widget.WidgetException`

Bases: `exceptions.Exception`

Fired when the widget gets an exception.

## 126.49 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.

**Warning:** This widget is highly experimental. The whole styling and implementation are not stable until this warning has been removed.

### 126.49.1 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:

top:' where 'top' is a blue link. At the bottom, there is a code block containing '\$ print \"Hello world\"'." data-bbox="124 454 648 594"/&gt;

### 126.49.2 Usage with Source

You can also render a rst file using the `RstDocument.source` property:

```
document = RstDocument(source='index.rst')
```

You can reference other documents with 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 is 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*, \**largs*)

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'`.



## ABSTRACT VIEW

New in version 1.5: This code is still experimental, and its API is subject to change in a future version.

The `AbstractView` widget has an `adapter` property for an adapter that mediates to data. The adapter manages an `item_view_instances` dict property that holds views for each data item, operating as a cache.

```
class kivy.uix.abstractview.AbstractView(**kwargs)
```

```
    Bases: kivy.uix.floatlayout.FloatLayout
```

```
    View using an Adapter as a data provider.
```

### **adapter**

The adapter can be one of several kinds of `adapters`. The most common example is the `ListAdapter` used for managing data items in a list.





# 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`.

## 128.1 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()
```

## 128.2 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.

### **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`.

### **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.

### **orientation**

Link to the [Accordion.orientation](#) property.

### **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
```

**class** kivy.uix.accordion.**AccordionException**

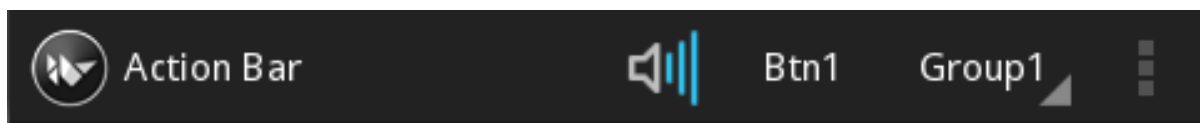
Bases: exceptions.Exception

AccordionException class.



# ACTION BAR

New in version 1.8.0.



The ActionBar widget is like Android's ActionBar, where items are stacked horizontally.

The `ActionBar` will contain one `ActionView` and many `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` will contain an `ActionOverflow`. A `ContextualActionView` is a subclass of an `ActionView`.

```
class kivy.uix.actionbar.ActionBarException
```

```
    Bases: exceptions.Exception
```

```
    ActionBarException class
```

```
class kivy.uix.actionbar.ActionItem
```

```
    Bases: 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 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.

`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`.

```
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 it's 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.

```
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.spinner.Spinner`

ActionGroup class, see module documentation for more information.

#### **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 a `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.

#### **overflow\_image**

Image to be used as an Overflow Image.

`overflow_image` is an `ObjectProperty` 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.

#### **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 an `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`.

#### **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.actionbar.ActionButton`

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'`.

#### **previous\_image**

Image for the 'previous' ActionButtons default graphical representation.

`previous_image` is a `StringProperty` and defaults to `'atlas://data/images/defaulttheme/previous_normal'`.

#### **title**

Title for ActionView.

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

#### **with\_previous**

Specifies whether clicking on ActionPrevious will load the previous screen or not. If True, the `previous_icon` will be shown otherwise it will not.

`with_previous` is a `BooleanProperty` and defaults to True.

**class** `kivy.uix.actionbar.ActionBar(**kwargs)`

Bases: `kivy.uix.boxlayout.BoxLayout`

ActionBar, see the module documentation for more information.

#### **Events**

`on_previous` Fired when `action_previous` of `action_view` is pressed.

#### **action\_view**

`action_view` of ActionBar.

`action_view` is an `ObjectProperty` and defaults to an instance of ActionView.

#### **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 ActionBars default graphical representation.

`background_image` is an `StringProperty` and defaults to `'atlas://data/images/defaulttheme/action_bar'`.

#### **border**

`border` to be applied to the `background_image`.

---

# ANCHOR LAYOUT

---



The **AnchorLayout** aligns 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'.

**padding**

Padding between the widget box and it's 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].

---

# BEHAVIORS

New in version 1.8.0.

This module implements behaviors that can be mixed with existing base widgets. For example, if you want to add a “button” capability to an *Image*, you could do:

```
class IconButton(ButtonBehavior, Image):  
    pass
```

---

**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.

---

```
class kivy.uix.behaviors.ButtonBehavior(**kwargs)
```

Bases: `object`

Button behavior.

#### 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).

#### **`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`, defaults to `None`.

#### **`state`**

State of the button, must be one of ‘normal’ or ‘down’. The state is ‘down’ only when the button is currently touched/clicked, otherwise ‘normal’.

**`state`** is an `OptionProperty`.

#### **`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.ButtonBehavior`

ToggleButton behavior, see `ToggleButton` module documentation for more information.

New in version 1.8.0.

**static** `get_widgets`(*groupname*)

Return the widgets contained in a specific group. If the group doesn't exist, an empty list will be returned.

---

**Important:** Always release the result of this method! 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. Garbage collector might need more elements before flushing it. The return of this method is informative, you've been warned!

**group**

Group of the button. If `None`, no group will be used (button is independent). If specified, `group` must be a hashable object, like a string. Only one button in a group can be in 'down' state.

`group` is a `ObjectProperty`

**class** `kivy.uix.behaviors.DragBehavior`(\*\**kwargs*)

Bases: `object`

Drag behavior. When combined with a widget, dragging in the rectangle defined by `drag_rectangle` will drag the widget.

For example, to make a popup which is draggable by its title do:

```
from kivy.uix.behaviors import DragBehavior
from kivy.uix.popup import Popup

class DragPopup(DragBehavior, Popup):
    pass
```

And in `.kv` do::

```
<DragPopup>:drag_rectangle: self.x, self.y+self._container.height, self.width, self.height -
    self._container.height drag_timeout: 10000000 drag_distance: 0
```

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 go to children. It is advisable that you base this value on the dpi of your target device's screen.

`drag_distance` is a `NumericProperty`, defaults to 20 (pixels), according to the default value of `scroll_distance` in user configuration.

**drag\_rect\_height**

Height of the axis aligned bounding rectangle where dragging is allowed.

`drag_rect_height` is a `NumericProperty`, defaults to 100.

**drag\_rect\_width**

Width of the axis aligned bounding rectangle where dragging is allowed.

`drag_rect_width` is a `NumericProperty`, 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`, 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`, 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 go to the children.

`drag_timeout` is a `NumericProperty`, defaults to 55 (milliseconds), according to the default value of `scroll_timeout` in user configuration.

```
class kivy.uix.behaviors.FocusBehavior(**kwargs)
```

Bases: `object`

Implements 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 than 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.

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 previous or next widget is not specified, `focus_next` and `focus_previous` default to `None`, which 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
```

New in version 1.8.1.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

### **focus\_next**

The **FocusBehavior** instance to acquire focus when tab is pressed when 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 a **ObjectProperty**, 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 *:attr:'focus\_previous* is the *StopIteration* class, focus won't move backward, but end here.

**focus\_previous** is a **ObjectProperty**, defaults to *None*.

### **focused**

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.

**focused** is a **BooleanProperty**, defaults to False.

### **is\_focusable**

Whether the instance can become focused. If focused, it'll lose focus when set to False.

**is\_focusable** is a **BooleanProperty**, defaults to True on a desktop (i.e. desktop is True in **config**), False otherwise.

### **keyboard**

The keyboard to bind, 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 get 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 information on the keyboard modes.

**keyboard** is a **AliasProperty**, defaults to *None*.

### **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` at the end after it is done if it didn't consume `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` at the end after it is done if it didn't consume `escape`.

See `on_key_down()`

#### **class kivy.uix.behaviors.CompoundSelectionBehavior**(*\*\*kwargs*)

Bases: `object`

Selection behavior implements the logic behind keyboard and touch selection of selectable widgets managed by the derived widget. For example, it could be combined with a `GridLayout` to add selection to the layout.

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, it 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.

When the controller needs to select a node it calls `select_node()` and `deselect_node()`. Therefore, they must be overwritten in order to affect the selected nodes. By default, the class doesn't listen to keyboard and touch events, therefore, 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.

For example, to add selection to a grid layout which will contain `Button` widgets:

```
class SelectableGrid(CompoundSelectionBehavior, GridLayout):

    def __init__(self, **kwargs):
        super(CompoundSelectionBehavior, self).__init__(**kwargs)
        keyboard = Window.request_keyboard(None, self)
        keyboard.bind(on_key_down=self.select_with_key_down,
                    on_key_up=self.select_with_key_up)

    def select_node(self, node):
        node.background_color = (1, 0, 0, 1)
        return super(CompoundSelectionBehavior, self).select_node(node)

    def deselect_node(self, node):
        node.background_color = (1, 1, 1, 1)
        super(CompoundSelectionBehavior, self).deselect_node(node)
```

Then, for each button added to the layout, bind `on_touch_down` of the button to `select_with_touch()` to pass on the touch events.

New in version 1.8.1.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

### **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\_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 is reversed to make it work by default for most layouts so that the final result is that e.g. home, although it will select the last element on this list, visually it'll 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**

*keystr*, 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**

Whether the keyboard can be used for selection. If False, keyboard inputs will be ignored.

`keyboard_select` is a `BooleanProperty`, 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`, defaults to False.

### **page\_count**

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when pageup (or pagedown) is pressed.

`page_count` is a `NumericProperty`, defaults to 10.

### **right\_count**

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the right (or left) arrow on the keyboard is pressed.

`right_count` is a `NumericProperty`, defaults to 1.

### **scroll\_count**

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the mouse scroll wheel is scrolled.

`right_count` is a `NumericProperty`, 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 to its selected state when this is 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 deslect 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 recieved 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.

`selected_nodes` is a `ListProperty` and defaults to the empty list, []. It is read-only and should not be modified.

### **touch\_multiselect**

A special touch mode which determines whether touch events, as processed with `select_with_touch()`, will add to the selection the currently touched node, 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 than `multiselect`, because when this is True simply touching an unselected node will select it, even if e.g. ctrl is not pressed. If this is False, however, ctrl is required to be held in order to add to selection when `multiselect` is True.

---

**Note:** `multiselect`, when False, will disable `touch_multiselect`.

---

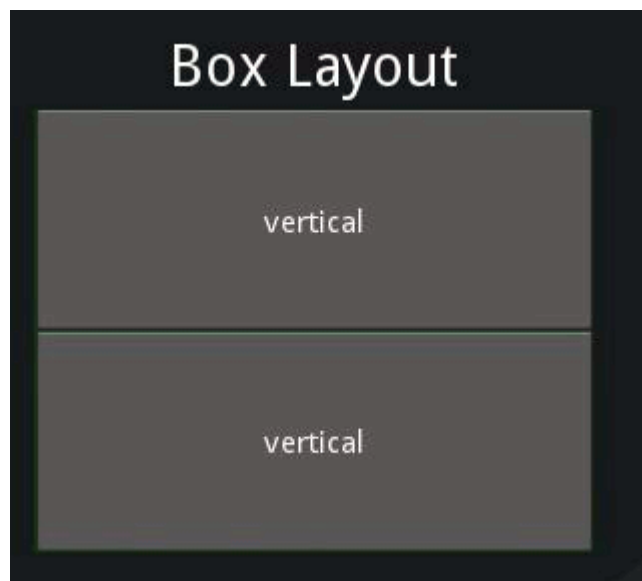
`touch_multiselect` is a `BooleanProperty`, defaults to False.

### **up\_count**

Determines by how much the selected node is moved up or down, relative to position of the last selected node, when the up (or down) arrow on the keyboard is pressed.

`up_count` is a `NumericProperty`, defaults to 1.

## 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.

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.

### **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].

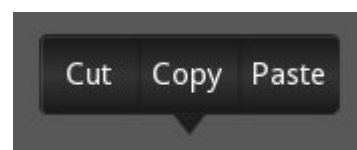
### **spacing**

Spacing between children, in pixels.

**spacing** is a **NumericProperty** and defaults to 0.

# 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.

## 133.1 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',
                    '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()

```

## 133.2 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.

#### **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).

`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: (top, right, bottom, left). Read the `BorderImage` instructions for more information about how to use it.

`border` is a `ListProperty` and defaults to `(16, 16, 16, 16)`

#### **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'`.

**show\_arrow**

Indicates whether to show arrow.

New in version 1.8.0.

`show_arrow` is a `BooleanProperty` and defaults to `True`.

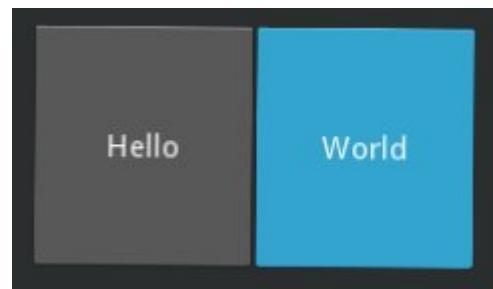
`class kivy.uix.bubble.BubbleButton(**kwargs)`

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.

# 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, you can use the same properties that you can use 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)
```

```
class kivy.uix.button.Button(**kwargs)
```

Bases: **kivy.uix.behaviors.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).

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 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 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: (top, right, bottom, left). Read the `BorderImage` instruction for more information about how to use it.

`border` is a `ListProperty` and defaults to (16, 16, 16, 16)

---

# 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]`.

---

# 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 use it horizontally or vertically. The carousel can display pages in loop or not.

Example:

```
class Example1(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 = Factory.AsyncImage(source=src, allow_stretch=True)  
            carousel.add_widget(image)  
        return carousel
```

```
Example1().run()
```

Changed in version 1.5.0: The carousel now supports active children, like the `ScrollView`. It will detect a swipe gesture according to `Carousel.scroll_timeout` and `Carousel.scroll_distance`.

In addition, the container used for adding a slide is now hidden in the API. We made a mistake by exposing it to the user. 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.

### **anim\_cancel\_duration**

Defines the duration of the animation when a swipe movement is not accepted. This is generally when the user doesn't swipe enough. 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 in the next/previous slide.

New in version 1.8.0.

### **current\_slide**

The currently shown slide.

`current_slide` is an `AliasProperty`.

Changed in version 1.5.0: The property doesn't expose the container used for storing the slide. It returns widget you have added.

### **direction**

Specifies the direction in which the slides are ordered i.e. the direction from which the user swipes to go from one slide to the next. 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 a `OptionProperty` and defaults to 'right'.

### **index**

Get/Set the current visible slide based on the index.

`index` is a `AliasProperty` and defaults to 0 (the first item).

### **load\_next** (*mode='next'*)

Animate to 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 swipe infinitely. When the user reaches the last page, they will return to first page when trying to swipe to the next.

`loop` is a `BooleanProperty` and defaults to False.

### **min\_move**

Defines the minimal distance from the edge where the movement is considered a swipe gesture and the Carousel will change its content. This is a percentage of the Carousel width. If the movement doesn't reach this minimal value, then 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. If `orientation` is 'horizontal', the next slide is to the right. If `orientation` is 'vertical', the next slide is towards the bottom.

`next_slide` is a `AliasProperty`.

Changed in version 1.5.0: The property doesn't expose the container used for storing the slide. It returns the widget you have added.

### **previous\_slide**

The previous slide in the Carousel. It is None if the current slide is the first slide in the Carousel. If `orientation` is 'horizontal', the previous slide is to the left. If `orientation` is 'vertical', the previous slide towards the bottom.

`previous_slide` is a `AliasProperty`.



Changed in version 1.5.0: This property doesn't expose the container used for storing the slide. It returns the widget you have added.

### **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, the scrolling will be disabled 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**

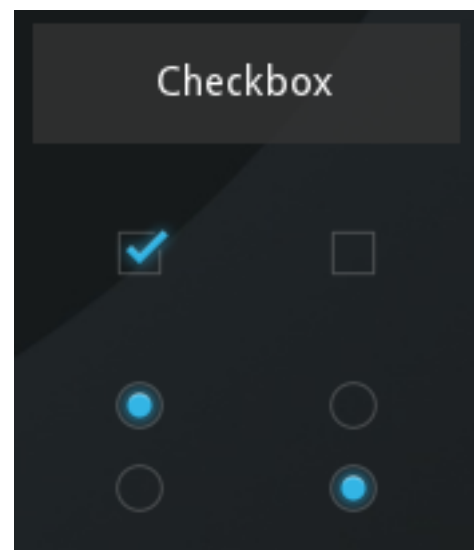
List of slides inside the `Carousel`. The slides are added when a widget is added to `Carousel` using `add_widget()`.

`slides` is a `ListProperty` and is read-only.



# 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.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 `BooleanProperty` and defaults to `False`.

## 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

```

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.

### 138.1 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.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`.

---

# 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 selected color. 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/DroidSansMono.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.

### 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.



---

# 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.

## 140.1 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))

runTouchApp(mainbutton)
```

## 140.2 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.

### 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 allow 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.

**container**

(internal) Property that will be set to the container of the dropdown list. It is a [GridLayout](#) by default.

**dismiss** (*\*largs*)

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.

**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.

**select** (*data*)

Call this method to trigger the *on\_select* event with the *data* selection. The *data* can be anything you want.



---

# EFFECTWIDGET

New in version 1.8.1: This code is still experimental, and its API is subject to change in a future version.

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 antialiasing, to emulating the appearance of a CRT monitor!

The basic usage is as follows:

```
w = EffectWidget()
w.add_widget(Button(text='Hello!'))
w.effects = [InvertEffect(), HorizontalBlurEffect(size=2.0)]
```

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.

---

**Note:** It is not efficient to resize an `EffectWidget`, as each `Fbo` is recreated every time. If you need to resize frequently, consider doing things a different way.

---

**Note:** 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.

---

## 141.1 Provided Effects

The module comes with several pre-written effects. Some have adjustable properties (e.g. blur radius), see the individual effect documentation for more details.

- `MonochromeEffect` - makes the widget grayscale.
- `InvertEffect` - inverts the widget colors.
- `ChannelMixEffect` - swaps around color channels.
- `ScanlinesEffect` - displays flickering scanlines.

- `PixelateEffect` - pixelates the image.
- `HorizontalBlurEffect` - Gaussian blurs horizontally.
- `VerticalBlurEffect` - Gaussian blurs vertically.
- `FXAAEffect` - applies a very basic AA.

## 141.2 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.gsl` property.

The `gsl` 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 colour at each point, then call your `effect` function to transform it. The parameters are:

- **color:** The normal colour 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 `gsl` 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 module documentation for full information on setting effects and creating your own.

### **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, 1)

### **effects**

List of all the effects to be applied. These should all be instances 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.

### **texture**

The output texture of our 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 module documentation for more details.

### **fbo**

The fbo currently using this effect. The `EffectBase` automatically handles this.

fbo is a `ObjectProperty` and defaults to None.

### **glsl**

The glsl string defining your effect function, see 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 if 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 values for the uniform variable with the given name are uploaded to the shader.

You must still manually declare your uniform variables at the top of your `glsl` string.

### **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 colours in monochrome.

```
class kivy.uix.effectwidget.InvertEffect(*args, **kwargs)
```

Bases: [kivy.uix.effectwidget.EffectBase](#)

Inverts the colours 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 antialiasing via fxaa.



---

# FILECHOOSER

New in version 1.0.5.

Changed in version 1.2.0: In the chooser template, the *controller* is not a direct reference anymore but a weak-reference. You must update all the notation *root.controller.xxx* to *root.controller().xxx*.

## 142.1 Simple example

main.py

```
#!/usr/bin/env python
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()

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"
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)

```

```
class kivy.uix.filechooser.FileChooserListView(**kwargs)
```

Bases: `kivy.uix.filechooser.FileChooserController`

Implementation of `FileChooserController` using a list view.

```
class kivy.uix.filechooser.FileChooserIconView(**kwargs)
```

Bases: `kivy.uix.filechooser.FileChooserController`

Implementation of `FileChooserController` using an icon view.

```
class kivy.uix.filechooser.FileChooserController(**kwargs)
```

Bases: `kivy.uix.floatlayout.FloatLayout`

Base for implementing a FileChooser. Don't use this class directly, but prefer using an implementation such as the `FileChooserListView` or `FileChooserIconView`.

#### Events

**`on_entry_added`:** `entry`, `parent`Fired when a root-level entry is added to the file list.

**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.

**on\_remove\_subentry: entry, parent** Fired when entries are removed from an entry, usually 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**

**BooleanProperty**, defaults to False. Determines whether directories are valid selections or not.

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 weird filename, undecodable without knowing it's 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.

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\_encodings is a **ListProperty** and defaults to ['utf-8', 'latin1', 'cp1252'],

**file\_system**

Implementation to access the file system. Must be an instance of **FileSystemAbstract**.

New in version 1.8.0.

**ObjectProperty**, defaults to **FileSystemLocal()**

**files**

Read-only **ListProperty**. The list of files in the directory specified by path after applying the filters.

**filter\_dirs**

**BooleanProperty**, defaults to False. Indicates whether filters should also apply to directories.

**filters**

**ListProperty**, defaults to [], equal to '\*'. Specifies the filters to be applied to the files in the directory.

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
<code>*</code>	matches everything
<code>?</code>	matches any single character
<code>[seq]</code>	matches any character in seq
<code>[!seq]</code>	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: If the filter is a callable (function or method), it will be called with the path and the file name as arguments for each file in the directory. The callable should return `True` to indicate a match and `False` otherwise.

### **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.).

### **multiselect**

**BooleanProperty**, defaults to `False`. Determines whether the user is able to select multiple files or not.

### **path**

**StringProperty**, 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 question marks (?) instead of their unicode characters.

### **progress\_cls**

Class to use for displaying a progress indicator for filechooser loading.

New in version 1.2.0.

**ObjectProperty**, defaults to `FileChooserProgress`.

Changed in version 1.8.0: If you set a string, the **Factory** will be used to resolve the class.

### **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`.

New in version 1.2.0.

**StringProperty**, defaults to `None`.

---

**Note:** Similar to `path`, if `rootpath` is specified, whether it's a bytes or unicode string determines the type of the filenames and paths read.

---

### **selection**

Read-only **ListProperty**. Contains the list of files that are currently selected.

### **show\_hidden**

**BooleanProperty**, defaults to `False`. Determines whether hidden files and folders should be shown.

### **sort\_func**

**ObjectProperty**. Provides a function to be called with a list of filenames, and the filesys-

tem implementation as the second argument. Returns a list of filenames sorted for display in the view.

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.

**path**

Current path of the FileChooser, read-only.

**total**

Total number of entries to load.

**class** `kivy.uix.filechooser.FileSystemAbstract`

Bases: `object`

Class for implementing a File System view that can be used with the `FileChooser`.attr:~*FileChooser.file\_system*.

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.

## FLOAT LAYOUT

The `FloatLayout` class honors only the `Widget.pos_hint` and `Widget.size_hint` attributes.



For example, say you create a `FloatLayout` with a size of (300, 300):

```
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 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.



---

# GESTURE SURFACE

New in version 1.8.1.

**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*  $\geq$  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, ie 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, ie 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.

**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 *kiivy.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



---

# 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.

## 145.1 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.

## 145.2 Column Width and Row Height

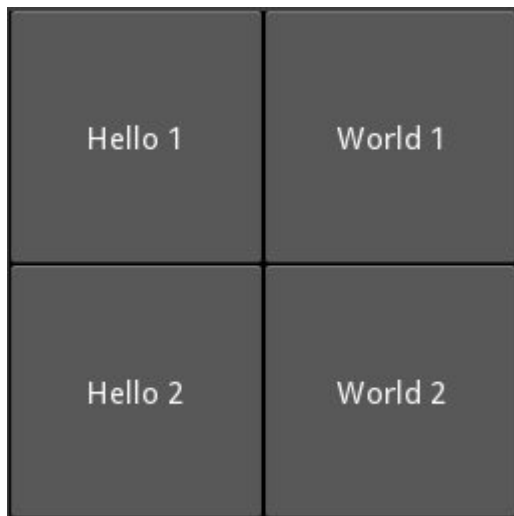
The column width/row height are determined in 3 steps:

- 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.

## 145.3 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'))
```

Hello 1	World 1
Hello 2	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'))
```

Hello 1	World 1
Hello 2	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 0.

#### **cols\_minimum**

List of minimum sizes for each column.

New in version 1.0.7.

`cols_minimum` is a `DictProperty` and defaults to {}.

**minimum\_height**

Minimum height needed to contain all children.

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.

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.

New in version 1.0.8.

`minimum_width` is a `kivy.properties.NumericProperty` and defaults to 0.

**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 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 0.

**rows\_minimum**

List of minimum sizes for each row.

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].



```
class kivy.uix.gridlayout.GridLayoutException
    Bases: exceptions.Exception
    Exception for errors if the grid layout manipulation fails.
```



# IMAGE

The **Image** widget is used to display an image:

```
wimg = Image(source='mylogo.png')
```

## 146.1 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.

## 146.2 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:  
            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).

#### **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 a `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 a `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.

Depending of the texture creation, the value will be a `Texture` or a `TextureRegion` object.

`texture` is a `ObjectProperty` and defaults to None.

### **texture\_size**

Texture size of the image.

**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.

---



# 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')
```

## 147.1 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

**[font=<str>][font]** Change the font

**[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.

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&bt;[/size]"
```

## 147.2 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]'
```

## 147.3 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):  
        """ 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]
    [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`.

---

### **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**

Text color, in the format (r, g, b, a)

New in version 1.8.0.

`disabled_color` is a `ListProperty` and defaults to [1, 1, 1, .5].

### **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 'DroidSans'.

### **font\_size**

Font size of the text, in pixels.

`font_size` is a `NumericProperty` and defaults to 12dp.

### **halign**

Horizontal alignment of the text.

`halign` is an `OptionProperty` and defaults to 'left'. Available options are : left, center, right and justify.

**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 1.6.0: A new option was added to `halign`, namely *justify*.

## **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.

## **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 1.8.1: `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 1.8.1: `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 1.8.1.

`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 `shorten_from` is the empty string, "", we split on every character fitting as much text as possible into the line.

New in version 1.8.1.

`split_str` is a `StringProperty` and defaults to "" (the empty string).

### 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 1.8.1.

`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\_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 constrains. 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(\*largs)**

Force texture recreation with the current Label properties.

After this function call, the `texture` and `texture_size` will be updated in this order.

### **valign**

Vertical alignment of the text.

`valign` is an `OptionProperty` and defaults to 'bottom'. Available options are : bottom, middle and top.

**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.

---

# LAYOUT

Layouts are used to calculate and assign widget positions.

The `Layout` class itself cannot be used directly. You should use one of the concrete 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`
- Stack layout : `kivy.uix.stacklayout.StackLayout`

## 148.1 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
```

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.

**do\_layout** (*\*args*)

This function is called when a layout is needed by a trigger. If you are writing a new Layout subclass, don't call this function directly but use `_trigger_layout()` instead.

New in version 1.0.8.



---

# LIST VIEW

New in version 1.5.

**Warning:** This code is still experimental, and its API is subject to change in a future version.

The `ListView` widget provides a scrollable/pannable viewport that is clipped to the scrollview's bounding box which contains list item view instances.

The `ListView` implements an `AbstractView` as a vertical, scrollable list. The `AbstractView` has one property: `adapter`. The `ListView` sets an adapter to one of a `SimpleListAdapter`, `ListAdapter` or a `DictAdapter`.

## 149.1 Introduction

Lists are central parts of many software projects. Kivy's approach to lists includes providing solutions for simple lists, along with a substantial framework for building lists of moderate to advanced complexity. For a new user, it can be difficult to ramp up from simple to advanced. For this reason, Kivy provides an extensive set of examples that you may wish to run first, to get a taste of the range of functionality offered. You can tell from the names of the examples that they illustrate the "ramping up" from simple to advanced:

- `kivy/examples/widgets/lists/list_simple.py`
- `kivy/examples/widgets/lists/list_simple_in_kv.py`
- `kivy/examples/widgets/lists/list_simple_in_kv_2.py`
- `kivy/examples/widgets/lists/list_master_detail.py`
- `kivy/examples/widgets/lists/list_two_up.py`
- `kivy/examples/widgets/lists/list_kv.py`
- `kivy/examples/widgets/lists/list_composite.py`
- `kivy/examples/widgets/lists/list_cascade.py`
- `kivy/examples/widgets/lists/list_cascade_dict.py`
- `kivy/examples/widgets/lists/list_cascade_images.py`
- `kivy/examples/widgets/lists/list_ops.py`

Many of the examples feature selection, some restricting selection to single selection, where only one item at a time can be selected, and others allowing multiple item selection. Many of the examples illustrate how selection in one list can be connected to actions and selections in another view or another list.

Find your own way of reading the documentation here, examining the source code for the example apps and running the examples. Some may prefer to read the documentation through first, others may want to run the examples and view their code. No matter what you do, going back and forth will likely be needed.

## 149.2 Basic Example

In its simplest form, we make a listview with 100 items:

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

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 2
        super(MainView, self).__init__(**kwargs)

        list_view = ListView(
            item_strings=[str(index) for index in range(100)])

        self.add_widget(list_view)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))
```

Or, we could declare the listview using the kv language:

```
from kivy.uix.modalview import ModalView
from kivy.uix.listview import ListView
from kivy.uix.gridlayout import GridLayout
from kivy.lang import Builder

Builder.load_string("""
<ListViewModal>:
    size_hint: None, None
    size: 400, 400
    ListView:
        size_hint: .8, .8
        item_strings: [str(index) for index in range(100)]
""")

class ListViewModal(ModalView):
    def __init__(self, **kwargs):
        super(ListViewModal, self).__init__(**kwargs)

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 1
        super(MainView, self).__init__(**kwargs)
```

```

listview_modal = ListViewModal()

self.add_widget(listview_modal)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))

```

## 149.3 Using an Adapter

Behind the scenes, the basic example above uses the `SimpleListAdapter`. When the constructor for the `ListView` sees that only a list of strings is provided as an argument (called `item_strings`), it creates an instance of `SimpleListAdapter` using the list of strings.

Simple in `SimpleListAdapter` means: *without selection support*. It is a scrollable list of items that does not respond to touch events.

To use a `SimpleListAdapter` explicitly when creating a `ListView` instance, do:

```

simple_list_adapter = SimpleListAdapter(
    data=["Item #{0}".format(i) for i in range(100)],
    cls=Label)

list_view = ListView(adapter=simple_list_adapter)

```

The instance of `SimpleListAdapter` has a required data argument which contains data items to use for instantiating Label views for the list view (note the `cls=Label` argument). The data items are strings. Each item string is set by the `SimpleListAdapter` as the `text` argument for each Label instantiation.

You can declare a `ListView` with an adapter in a kv file with special attention given to the way longer python blocks are indented:

```

from kivy.uix.modalview import ModalView
from kivy.uix.listview import ListView
from kivy.uix.gridlayout import GridLayout
from kivy.lang import Builder
from kivy.factory import Factory

# Note the special nature of indentation in the adapter declaration, where
# the adapter: is on one line, then the value side must be given at one
# level of indentation.

Builder.load_string("""
#:import label kivy.uix.label
#:import sla kivy.adapters.simplelistadapter

<ListViewModal>:
    size_hint: None, None
    size: 400, 400
    ListView:
        size_hint: .8, .8
        adapter:
            sla.SimpleListAdapter(
                data=["Item #{0}".format(i) for i in range(100)],
                cls=label.Label)
""")

```

```

class ListViewModal(ModalView):
    def __init__(self, **kwargs):
        super(ListViewModal, self).__init__(**kwargs)

class MainView(GridLayout):

    def __init__(self, **kwargs):
        kwargs['cols'] = 1
        super(MainView, self).__init__(**kwargs)

        listview_modal = ListViewModal()

        self.add_widget(listview_modal)

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MainView(width=800))

```

## 149.4 ListAdapter and DictAdapter

For many uses of a list, the data is more than a simple list of strings. Selection functionality is also often needed. The `ListAdapter` and `DictAdapter` cover these more elaborate needs.

The `ListAdapter` is the base class for `DictAdapter`, so we can start with it.

See the `ListAdapter` docs for details, but here are synopses of its arguments:

- *data*: strings, class instances, dicts, etc. that form the basis data for instantiating views.
- *cls*: a Kivy view that is to be instantiated for each list item. There are several built-in types available, including `ListItemLabel` and `ListItemButton`, or you can make your own class that mixes in the required `SelectableView`.
- *template*: the name of a Kivy language (kv) template that defines the Kivy view for each list item.

---

**Note:** Pick only one, *cls* or *template*, to provide as an argument.

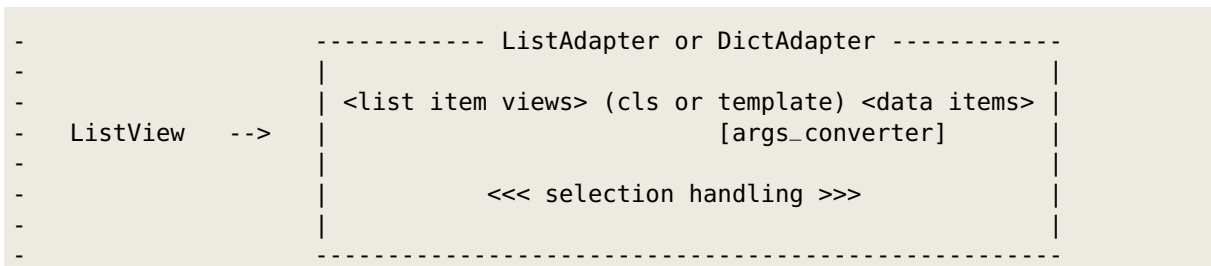
---

- *args\_converter*: a function that takes a data item object as input and uses it to build and return an args dict, ready to be used in a call to instantiate item views using the item view *cls* or *template*. In the case of *cls*, the args dict acts as a kwargs object. For a *template*, it is treated as a context (ctx) but is essentially similar in form to the kwargs usage.
- *selection\_mode*: a string with the value 'single', 'multiple' or others (See `selection_mode` for details).
- *allow\_empty\_selection*: a boolean, which if False (the default), forces there to always be a selection if there is data available. If True, selection happens only as a result of user action.

In narrative, we can summarize as follows:

A listview's adapter takes data items and uses an `args_converter` function to transform them into arguments for making list item view instances, using either a *cls* or a kv *template*.

In a graphic, a summary of the relationship between a listview and its list adapter, looks like this:



A `DictAdapter` has the same arguments and requirements as `ListAdapter` except for two things:

1. There is an additional argument, `sorted_keys`, which must meet the requirements of normal python dictionary keys.
2. The data argument is, as you would expect, a dict. Keys in the dict must include the keys in the `sorted_keys` argument, but they may form a superset of the keys in `sorted_keys`. Values may be strings, class instances, dicts, etc. (The `args_converter` uses it accordingly).

## 149.5 Using an Args Converter

A `ListView` allows use of built-in list item views, such as `ListItemButton`, your own custom item view class or a custom kv template. Whichever type of list item view is used, an `args_converter` function is needed to prepare, per list data item, args for the cls or template.

**Note:** Only the `ListItemLabel`, `ListItemButton` or custom classes like them (and not the simple `Label` or `Button` classes) are to be used in the listview system.

**Warning:** `ListItemButton` inherits the `background_normal` and `background_down` properties from the `Button` widget, so the `selected_color` and `deselected_color` are not represented faithfully by default.

Here is an `args_converter` for use with the built-in `ListItemButton` specified as a normal Python function:

```

def args_converter(row_index, an_obj):
    return {'text': an_obj.text,
            'size_hint_y': None,
            'height': 25}

```

and as a lambda:

```

args_converter = lambda row_index, an_obj: {'text': an_obj.text, 'size_hint_y': None,
                                             'height': 25}

```

In the args converter example above, the data item is assumed to be an object (class instance), hence the reference `an_obj.text`.

Here is an example of an args converter that works with list data items that are dicts:

```

args_converter = lambda row_index, obj: {'text': obj['text'],
                                         'size_hint_y': None,
                                         'height': 25}

```

So, it is the responsibility of the developer to code the `args_converter` according to the data at hand. The `row_index` argument can be useful in some cases, such as when custom labels are needed.

## 149.6 An Example ListView

Now, to some example code:

```
from kivy.adapters.listadapter import ListAdapter
from kivy.uix.listview import ListItemButton, ListView

data = [{'text': str(i), 'is_selected': False} for i in range(100)]

args_converter = lambda row_index, rec: {'text': rec['text'],
                                         'size_hint_y': None,
                                         'height': 25}

list_adapter = ListAdapter(data=data,
                           args_converter=args_converter,
                           cls=ListItemButton,
                           selection_mode='single',
                           allow_empty_selection=False)

list_view = ListView(adapter=list_adapter)
```

This listview will show 100 buttons with text of 0 to 100. The args converter function works on dict items in the data. `ListItemButton` views will be instantiated from the args converted by `args_converter` for each data item. The listview will only allow single selection: additional touches will be ignored. When the listview is first shown, the first item will already be selected because `allow_empty_selection` is `False`.

The `ListItemLabel` works in much the same way as the `ListItemButton`.

## 149.7 Using a Custom Item View Class

The data used in an adapter can be any of the normal Python types, such as strings, class instances and dictionaries. They can also be custom classes, as shown below. It is up to the programmer to assure that the `args_converter` performs the appropriate conversions.

Here we make a simple `DataItem` class that has the required `text` and `is_selected` properties:

```
from kivy.uix.listview import ListItemButton
from kivy.adapters.listadapter import ListAdapter

class DataItem(object):
    def __init__(self, text='', is_selected=False):
        self.text = text
        self.is_selected = is_selected

data_items = []
data_items.append(DataItem(text='cat'))
data_items.append(DataItem(text='dog'))
data_items.append(DataItem(text='frog'))

list_item_args_converter = lambda row_index, obj: {'text': obj.text,
                                                  'size_hint_y': None,
                                                  'height': 25}

list_adapter = ListAdapter(data=data_items,
                           args_converter=list_item_args_converter,
                           selection_mode='single',
```

```
propagate_selection_to_data=True,  
allow_empty_selection=False,  
cls=ListItemButton)
```

```
list_view = ListView(adapter=list_adapter)
```

The data is set in a `ListAdapter` along with a list item `args_converter` function above (lambda) and arguments concerning selection: only single selection is allowed, and selection in the listview will propagate to the data items. The propagation setting means that the `is_selected` property for each data item will be set and kept in sync with the list item views. By having `allow_empty_selection=False`, when the listview first appears, the first item, 'cat', will already be selected. The list adapter will instantiate a `ListItemButton` class instance for each data item, using the assigned `args_converter`.

The `list_view` would be added to a view with `add_widget()` after the last line, where it is created. See the basic example at the top of this documentation for an example of `add_widget()` use in the context of a sample app.

You may also use the provided `SelectableDataItem` mixin to make a custom class. Instead of the “manually-constructed” `DataItem` class above, we could do:

```
from kivy.adapters.models import SelectableDataItem  
  
class DataItem(SelectableDataItem):  
    # Add properties here.  
    pass
```

`SelectableDataItem` is a simple mixin class that has an `is_selected` property.

## 149.8 Using an Item View Template

`SelectableView` is another simple mixin class that has required properties for a list item: `text`, and `is_selected`. To make your own template, mix it in as follows:

```
from kivy.uix.listview import ListItemButton  
from kivy.uix.listview import SelectableView  
  
Builder.load_string("""  
[CustomListItem@SelectableView+BoxLayout]:  
    size_hint_y: ctx.size_hint_y  
    height: ctx.height  
    ListItemButton:  
        text: ctx.text  
        is_selected: ctx.is_selected  
""")
```

A class called `CustomListItem` will be instantiated for each list item. Note that it is a `BoxLayout` and is thus a kind of container. It contains a `ListItemButton` instance.

Using the power of the Kivy language (kv), you can easily build composite list items – in addition to `ListItemButton`, you could have a `ListItemLabel`, or a custom class you have defined and registered with the system.

An `args_converter` needs to be constructed that goes along with such a kv template. For example, to use the kv template above:

```
list_item_args_converter = \  
    lambda row_index, rec: {'text': rec['text']},
```

```

        'is_selected': rec['is_selected'],
        'size_hint_y': None,
        'height': 25}

integers_dict = \
    { str(i): {'text': str(i), 'is_selected': False} for i in range(100)}

dict_adapter = DictAdapter(sorted_keys=[str(i) for i in range(100)],
                           data=integers_dict,
                           args_converter=list_item_args_converter,
                           template='CustomListItem')

list_view = ListView(adapter=dict_adapter)

```

A dict adapter is created with 1..100 integer strings as `sorted_keys`, and an `integers_dict` as data. `integers_dict` has the integer strings as keys and dicts with `text` and `is_selected` properties. The `CustomListItem` defined above in the `Builder.load_string()` call is set as the kv template for the list item views. The `list_item_args_converter` lambda function will take each dict in `integers_dict` and will return an args dict, ready for passing as the context (ctx) for the template.

The `list_view` would be added to a view with `add_widget()` after the last line, where it is created. Again, see the basic example above for `add_widget()` use.

## 149.9 Using CompositeListItem

The class `CompositeListItem` is another option for building advanced composite list items. The kv language approach has its advantages, but here we build a composite list view using a straight Kivy widget method:

```

args_converter = lambda row_index, rec:
    {'text': rec['text'],
     'size_hint_y': None,
     'height': 25,
     'cls_dicts': [{'cls': ListItemButton,
                    'kwargs': {'text': rec['text']}},
                  {'cls': ListItemLabel,
                    'kwargs': {'text': "Middle-{}".format(rec['text']),
                               'is_representing_cls': True}},
                  {'cls': ListItemButton,
                    'kwargs': {'text': rec['text']}]}}

item_strings = ["{}"].format(index) for index in range(100)]

integers_dict =
    { str(i): {'text': str(i), 'is_selected': False} for i in range(100)}

dict_adapter = DictAdapter(sorted_keys=item_strings,
                           data=integers_dict,
                           args_converter=args_converter,
                           selection_mode='single',
                           allow_empty_selection=False,
                           cls=CompositeListItem)

list_view = ListView(adapter=dict_adapter)

```

The `args_converter` is somewhat complicated, so we should go through the details. Observe in the `DictAdapter` instantiation that `CompositeListItem` instance is set as the `cls` to be instantiated for each list item. The `args_converter` will make args dicts for this `cls`. In the `args_converter`, the first three items, `text`, `size_hint_y`, and `height`, are arguments for `CompositeListItem` itself. After that you see a `cls_dicts` list that contains argument sets for each of the member widgets for this composite:



`ListItemButton` and `ListItemLabel`. This is a similar approach to using a kv template described above.

The `sorted_keys` and `data` arguments for the dict adapter are the same as in the previous code example.

For details on how `CompositeListItem` works, examine the code, looking for how parsing of the `cls_dicts` list and `kwargs` processing is done.

## 149.10 Uses for Selection

What can we do with selection? Combining selection with the system of bindings in Kivy, we can build a wide range of user interface designs.

We could make data items that contain the names of dog breeds, and connect the selection of dog breed to the display of details in another view, which would update automatically on selection. This is done via a binding to the `on_selection_change` event:

```
list_adapter.bind(on_selection_change=callback_function)
```

where `callback_function()` does whatever is needed for the update. See the example called `list_master_detail.py`, and imagine that the list on the left would be a list of dog breeds, and the detail view on the right would show details for a selected dog breed.

In another example, we could set the `selection_mode` of a listview to 'multiple', and load it with a list of answers to a multiple-choice question. The question could have several correct answers. A color swatch view could be bound to selection change, as above, so that it turns green as soon as the correct choices are made, unless the number of touches exceeds a limit, when the answer session would be terminated. See the examples that feature thumbnail images to get some ideas, e.g., `list_cascade_dict.py`.

In a more involved example, we could chain together three listviews, where selection in the first controls the items shown in the second, and selection in the second controls the items shown in the third. If `allow_empty_selection` were set to `False` for these listviews, a dynamic system of selection "cascading" from one list to the next, would result.

There are so many ways that listviews and Kivy bindings functionality can be used, that we have only scratched the surface here. For on-disk examples, see these:

```
kivy/examples/widgets/lists/list_*.py
```

Several examples show the "cascading" behavior described above. Others demonstrate the use of kv templates and composite list views.

```
class kivy.uix.listview.SelectableView(**kwargs)
```

Bases: `object`

The `SelectableView` mixin is used to design list items and other classes that are to be instantiated by an adapter for use in a listview. The `ListAdapter` and `DictAdapter` adapters are selection-enabled. `select()` and `deselect()` are to be overridden with display code to mark items as selected or not, if desired.

```
deselect(*args)
```

The list item is responsible for updating the display for being unselected, if desired.

**index**

The index into the underlying data list or the data item this view represents.

`index` is a `NumericProperty`, default to -1.

### **is\_selected**

A `SelectableView` instance carries this property, which should be kept in sync with the equivalent property in the data item it represents.

`is_selected` is a `BooleanProperty`, default to `False`.

### **select(\*args)**

The list item is responsible for updating the display for being selected, if desired.

`class kivy.uix.listview.ListItemButton(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.button.Button`

`ListItemButton` mixes `SelectableView` with `Button` to produce a button suitable for use in `ListView`.

### **deselected\_color**

`deselected_color` is a `ListProperty` and defaults to `[0., 1., 0., 1]`.

### **selected\_color**

`selected_color` is a `ListProperty` and defaults to `[1., 0., 0., 1]`.

`class kivy.uix.listview.ListItemLabel(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.label.Label`

`ListItemLabel` mixes `SelectableView` with `Label` to produce a label suitable for use in `ListView`.

`class kivy.uix.listview.CompositeListItem(**kwargs)`

Bases: `kivy.uix.listview.SelectableView`, `kivy.uix.boxlayout.BoxLayout`

`CompositeListItem` mixes `SelectableView` with `BoxLayout` for a generic container-style list item, to be used in `ListView`.

### **background\_color**

`ListItem` subclasses `Button`, which has `background_color`, but for a composite list item, we must add this property.

`background_color` is a `ListProperty` and defaults to `[1, 1, 1, 1]`.

### **deselected\_color**

`deselected_color` is a `ListProperty` and defaults to  `[.33, .33, .33, 1]`.

### **representing\_cls**

Which component view class, if any, should represent for the composite list item in `__repr__()`?

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

### **selected\_color**

`selected_color` is a `ListProperty` and defaults to `[1., 0., 0., 1]`.

`class kivy.uix.listview.ListView(**kwargs)`

Bases: `kivy.uix.abstractview.AbstractView`, `kivy.event.EventDispatcher`

`ListView` is a primary high-level widget, handling the common task of presenting items in a scrolling list. Flexibility is afforded by use of a variety of adapters to interface with data.

The adapter property comes via the mixed in `AbstractView` class.

`ListView` also subclasses `EventDispatcher` for scrolling. The event `on_scroll_complete` is used in refreshing the main view.

For a simple list of string items, without selection, use `SimpleListAdapter`. For list items that respond to selection, ranging from simple items to advanced composites, use `ListAdapter`. For an alternate powerful adapter, use `DictAdapter`, rounding out the choice for designing highly interactive lists.

## Events

**`on_scroll_complete`**: (boolean, )Fired when scrolling completes.

## container

The container is a `GridLayout` widget held within a `ScrollView` widget. (See the associated kv block in the `Builder.load_string()` setup). Item view instances managed and provided by the adapter are added to this container. The container is cleared with a call to `clear_widgets()` when the list is rebuilt by the `populate()` method. A padding `Widget` instance is also added as needed, depending on the row height calculations.

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

## divider

[TODO] Not used.

## divider\_height

[TODO] Not used.

## item\_strings

If `item_strings` is provided, create an instance of `SimpleListAdapter` with this list of strings, and use it to manage a no-selection list.

`item_strings` is a `ListProperty` and defaults to `[]`.

## row\_height

The `row_height` property is calculated on the basis of the height of the container and the count of items.

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

## scrolling

If the `scroll_to()` method is called while scrolling operations are happening, a call recursion error can occur. `scroll_to()` checks to see that `scrolling` is `False` before calling `populate()`. `scroll_to()` dispatches a `scrolling_complete` event, which sets `scrolling` back to `False`.

`scrolling` is a `BooleanProperty` and defaults to `False`.



---

# 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.

## 150.1 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)

# bind the on_press event of the button to the dismiss function
content.bind(on_press=view.dismiss)

# open the view
view.open()
```

## 150.2 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\_open*: Fired when the ModalView is opened.

*on\_dismiss*: Fired when the ModalView is closed. If the callback returns True, the dismiss will be canceled.

### 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).

`background_color` is a `ListProperty` and defaults to [0, 0, 0, .7].

### 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: (top, right, bottom, 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(\*args, \*\*kwargs)

Close the view if it is open. If you really want to close the view, whatever the `on_dismiss` event returns, you can use the *force* argument:

```
view = ModalView(...)  
view.dismiss(force=True)
```

When the view is dismissed, it will be faded out before being removed from the parent. If you don't want animation, use:

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

**open** (*\*args*)

Show the view window from the `attach_to` widget. If set, it will attach to the nearest window. If the widget is not attached to any window, the view will attach to the global `Window`.





---

# PAGELAYOUT

The `PageLayout` class allow to create a simple multiple page layout, in a way that allows easy flipping of one page to another using borders.

`PageLayout` doesn't honor `size_hint` or `pos_hint` in any way currently.

New in version 1.8.0.

example:

```
PageLayout:
    Button:
        text: 'page1'

    Button:
        text: 'page2'

    Button:
        text: 'page3'
```

```
class kivy.uix.pagelayout.PageLayout (**kwargs)
```

Bases: `kivy.uix.layout.Layout`

`PageLayout` class. See module documentation for more information

## **border**

Width of the border used around current page to display previous/next page when needed.

`border` is a `NumericProperty`, default to 0.

## **page**

Currently displayed page.

`page` is a `NumericProperty`, default to 0.

## **swipe\_threshold**

Thresold to the swipe action triggering, as percentage of the widget size.

`swipe_threshold` is a `NumericProperty`, default to .5.



# 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.

## 152.1 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 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()
```

## 152.2 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.

### content

Content of the popup that is displayed just under the title.

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

### 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\_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\_size**

Represents the font size of the popup title.

New in version 1.6.0.

`title_size` is a `NumericProperty` and defaults to '14sp'.

**class kivy.uix.popup.PopupException**

Bases: `exceptions.Exception`

Popup exception, fired when multiple content widgets are added to the popup.

New in version 1.4.0.



---

# 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 progressbar. If the value is  $< 0$  or  $> \text{max}$ , 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
>>> slider.value_normalized
0.5
```

`value_normalized` is an `AliasProperty`.



---

# 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.

For example, if you create a [RelativeLayout](#), add a widget with `position = (0,0)`, the child widget will also move when you change the position of the [RelativeLayout](#). The child widgets coordinates remain `(0,0)` i.e. they are always relative to the containing layout.

## 154.1 Coordinate Systems

### 154.1.1 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.

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.

### 154.1.2 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.

### 154.1.3 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.

#### 154.1.4 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.

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.



---

# 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.

**Warning:** This widget is highly experimental. The whole styling and implementation are not stable until this warning has been removed.

## 15.1 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:

# Hello world

---

This is an **emphased text**, some interpreted text. And this is a reference to [top](#):

```
$ print "Hello world"
```

## 155.2 Usage with Source

You can also render a rst file using the `RstDocument.source` property:

```
document = RstDocument(source='index.rst')
```

You can reference other documents with 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, \*largs*)

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'`.



---

# 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.

**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.



# 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 similar to a RelativeLayout (see [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 it's 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()`.

## 157.1 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)
```

## 157.2 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)
```

## 157.3 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.

## 157.4 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.)
```

## 157.5 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 1.8.1: 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`.

### **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.

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 1.8.1.

### **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.

`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.

**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.

---

# SCATTER LAYOUT

New in version 1.6.0.

This layout behaves just like a `RelativeLayout`. For example, if you create a `ScatterLayout`, add a widget with position = (0,0), the child widget will also move when you change the position of the `ScatterLayout`. The child widget's coordinates remain (0,0), i.e. they are relative to the containing layout.

However, since `ScatterLayout` is implemented using a `Scatter` widget, you can also translate, rotate and scale the layout using touches (mouse or fingers) just like 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
```

```
    RelativeLayout class, see module documentation for more information.
```





---

# SCREEN MANAGER

New in version 1.4.0.

**Warning:** This widget is still experimental, and its API is subject to change in a future version.

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.

## 159.1 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'
```

From 1.8.0, you can now switch dynamically to a new screen, change the transition options and remove the previous one by using `ScreenManager.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')
```

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.

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'
            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()
```

## 159.2 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
- **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 it's 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.

#### **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\_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.

#### **previous()**

Return the name of the previous screen from the screen list.

#### **screen\_names**

List of the names of all the `Screen` widgets added. The list is read only.

`screen_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 must not change the list manually. Use `Screen.add_widget()` instead.

`screens` is a `ListProperty` and defaults to [], read-only.

#### **switch\_to(screen, \*\*options)**

Add a new screen to the `ScreenManager` and switch to it. The previous screen will be removed from the children. `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 screen that will be hidden and the screen that will be shown. By default, an instance of `SlideTransition` will be given.

For example, if you want to change to a `WipeTransition`:

```

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'

```

Changed in version 1.8.0: Default transition has been changed from `SwapTransition` to `SlideTransition`.

### class kivy.uix.screenmanager.ScreenManagerException

Bases: `exceptions.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)
```

**clearcolor**

Sets the color of Fbo ClearColor.

New in version 1.8.1.

`clearcolor` is a `ListProperty` and defaults to `[0, 0, 0, 1]`.

**fs**

Fragment shader to use.

`fs` is a `StringProperty` and defaults to `None`.

**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`

Bases: `kivy.uix.screenmanager.TransitionBase`

Swap transition that looks like iOS transition when a new window appears on the screen.

**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.





---

# SCROLL VIEW

New in version 1.0.4.

The `ScrollView` widget provides a scrollable/pannable viewport that is clipped at the scrollview's bounding box.

## 160.1 Scrolling Behavior

The `ScrollView` accepts only one child and applies a viewport/window to it according to the `ScrollView.scroll_x` and `ScrollView.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:

- **`ScrollView.scroll_distance`**: the minimum distance to travel, defaults to 20 pixels.
- **`ScrollView.scroll_timeout`**: the maximum time period, defaults to 250 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.

## 160.2 Limiting to the X or Y Axis

By default, the `ScrollView` allows scrolling in both the X and Y axes. You can explicitly disable scrolling on an axis by setting `ScrollView.do_scroll_x` or `ScrollView.do_scroll_y` to `False`.

## 160.3 Managing the Content Size and Position

`ScrollView` manages the position of its children similarly to a `RelativeLayout` (see `relativelayout`) but not the size. You must carefully specify the `size_hint` of your content to get the desired scroll/pan effect.

By default, `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.

To scroll a `GridLayout` on Y-axis/vertically, set the child's width identical to that of the `ScrollView` (`size_hint_x=1`, default), and set the `size_hint_y` property to `None`:

```
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(30):
    btn = Button(text=str(i), size_hint_y=None, height=40)
    layout.add_widget(btn)
root = ScrollView(size_hint=(None, None), size=(400, 400))
root.add_widget(layout)
```

## 160.4 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 `ScrollView.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 `ScrollView.effect_x` and/or `ScrollView.effect_y` to an instance of the effect you want to use. This will override the default effect set in `ScrollView.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.

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

### **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 1.8.1.

`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**, default 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**, default 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.

**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\_type**

Sets the type of scrolling to use for the content of the scrollview. Available options are: ['content'], ['bars'], ['bars', 'content'].

New in version 1.8.0.

`scroll_type` is a `OptionProperty`, 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.

**update\_from\_scroll** (*\*args*)

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**

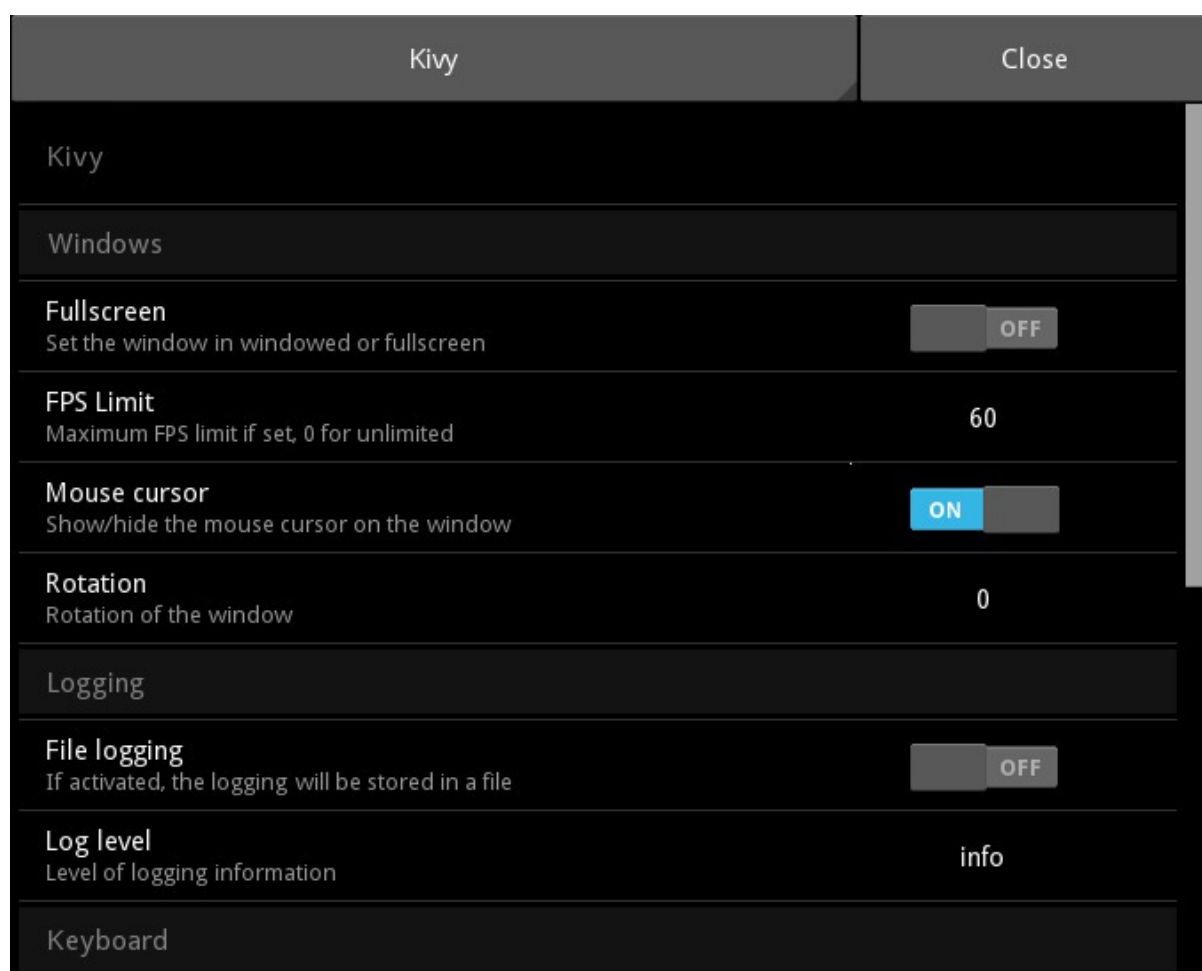
(internal) Size of the internal viewport. This is the size of your only child in the scrollview.



# SETTINGS

New in version 1.0.7.

This module is 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 with the `App` class. Use `Settings.add_kivy_panel()` to configure the

Kivy core settings in a panel.

## 161.1 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 default values for all 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",
    "true": "auto"
  }
]
```

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	<code>SettingTitle</code>
bool	<code>SettingBoolean</code>
numeric	<code>SettingNumeric</code>
options	<code>SettingOptions</code>
string	<code>SettingString</code>
path	<code>SettingPath</code> (new from 1.1.0)

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 `SettingTitle.title` property 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...
```

## 161.2 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.

```
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 section/key/value 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`.

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 `:class'InterfaceWithSidebar'`.

Changed in version 1.8.0: If you set a string, the [Factory](#) will be used to resolve the class.

### **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.

### **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`.

**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>'.  
`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.

### popup

(internal) Used to store the current popup when it is 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.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** – A `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** – A `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.

### **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.

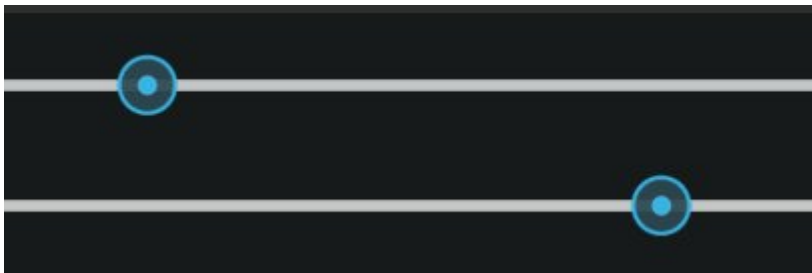
**Parametersuid** – A panel uid. It should be used to retrieve and display a settings panel that has previously been added with `add_panel()`.

### **panels**

(internal) Stores a dictionary mapping settings panels to their uids.

`panels` is a `DictProperty` and defaults to `{}`.

# 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')
```

```
class kivy.uix.slider.Slider(**kwargs)
```

Bases: `kivy.uix.widget.Widget`

Class for creating a Slider widget.

Check module documentation for more details.

### 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.

### 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 10. The range of the slider is reduced from padding \*2 on the screen. It allows drawing a cursor of 20px width without having the cursor go out of the widget.

`padding` is a `NumericProperty` and defaults to 10.

### 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`).

### 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
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`.



# SPINNER

New in version 1.4.0.



Spinner is a widget that provide a quick way to select one value from a set. In the default state, a spinner show its currently selected value. Touching the spinner displays a dropdown menu with all 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 showed
    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, 'have text', text)

spinner.bind(text=show_selected_value)
```

runTouchApp(spinner)

**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 you set a string, the `Factory` will be used to resolve the class.

#### **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 at least:

- a `text` property, used to display the value.
- an `on_release` event, used to trigger the option when pressed/touched.

`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.

#### **values**

Values that can be selected by the user. It must be a list of strings.

`values` is a `ListProperty` and defaults to [].

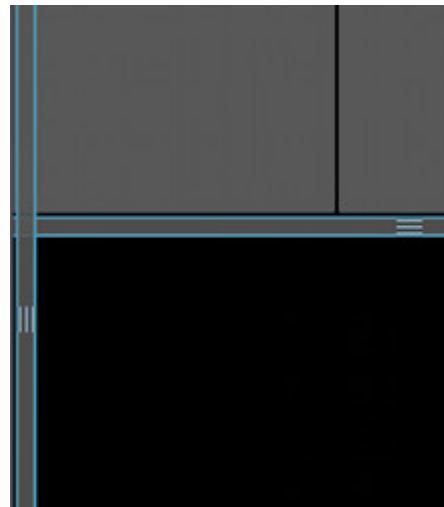
**class** `kivy.uix.spinner.SpinnerOption` (\*\*kwargs)

Bases: `kivy.uix.button.Button`

Special button used in the dropdown list. We just set the default `size_hint_y` and `height`.

# SPLITTER

New in version 1.5.0.



The **Splitter** is a widget that helps you re-size it's 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 it's 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
    background_down: 'path to pressed horizontal image' if self.horizontal else 'path to vertical
```

`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.

#### border

Border used for the `BorderImage` graphics instruction.

This must be a list of four values: (top, right, bottom, left). Read the `BorderImage` instructions for more information about how to use it.

`border` is a `ListProperty` and defaults to (4, 4, 4, 4).

#### 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 1.8.1.

#### 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`.

#### 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.

`keep_within_parent` is a `BooleanProperty` and defaults to False.

New in version 1.8.1.

#### 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`.

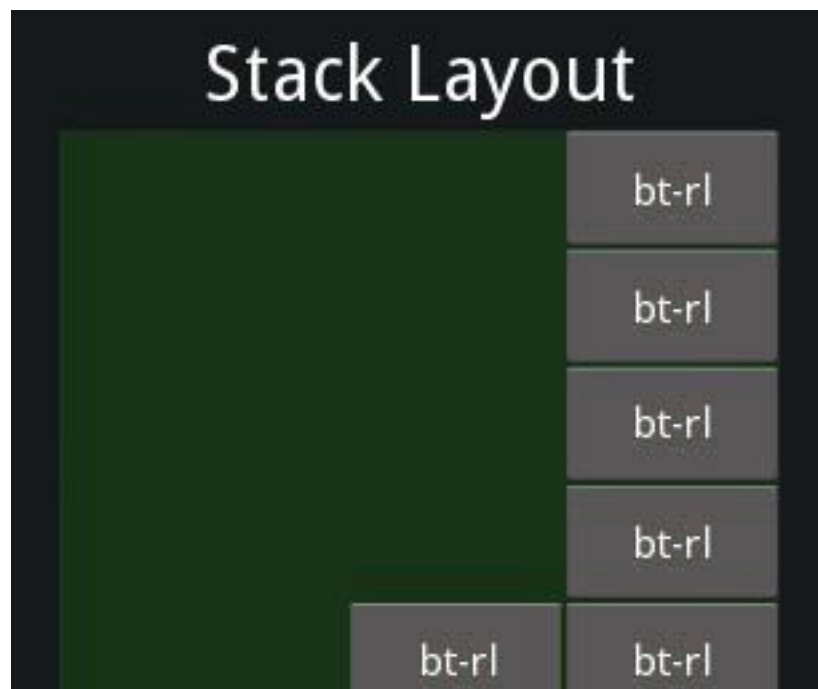
Changed in version 1.8.0: If you set a string, the `Factory` will be used to resolve the class.

#### strip\_size

Specifies the size of resize strip

`strip_size` is a `NumericProperty` defaults to `10pt`

# 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.

#### **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].



## 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 8 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 their behavior. For example:

```
class BoxStencil(BoxLayout, StencilView):  
    pass
```

---

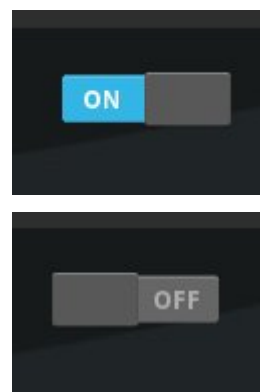
```
class kivy.uix.stencilview.StencilView(**kwargs)  
    Bases: kivy.uix.widget.Widget
```

StencilView class. See module documentation for more information.



# 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.

---

```
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.

**touch\_control**

(internal) Contains the touch that currently interacts with the switch.

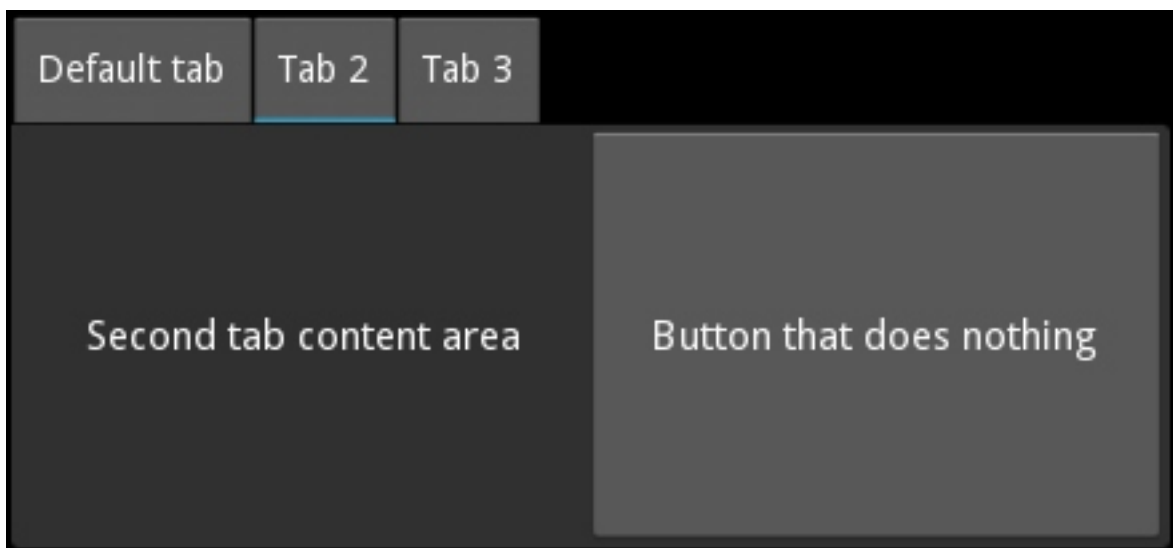
`touch_control` is an `ObjectProperty` and defaults to None.

**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.

# 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.

## 168.1 Simple example

```
'''
TabbedPanel
=====

Test of the widget TabbedPanel.
'''

from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.floatlayout import FloatLayout
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\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.

---

## 168.2 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.

---

**Note:** The `default_tab` functionality is turned off by default since 1.5.0. To turn it back on, set `do_default_tab = True`.

---

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.

#### **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: (top, right, bottom, left). Read the `BorderImage` instructions for more information.

`border` is a `ListProperty` and defaults to (16, 16, 16, 16)

#### **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'.

#### **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 an empty image.

#### **switch\_to**(*header*)

Switch to a specific panel header.

#### **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 'bottom\_mid'.

### **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`.

`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.

`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`.

**class** kivy.uix.tabbedpanel.**TabbedPanelException**

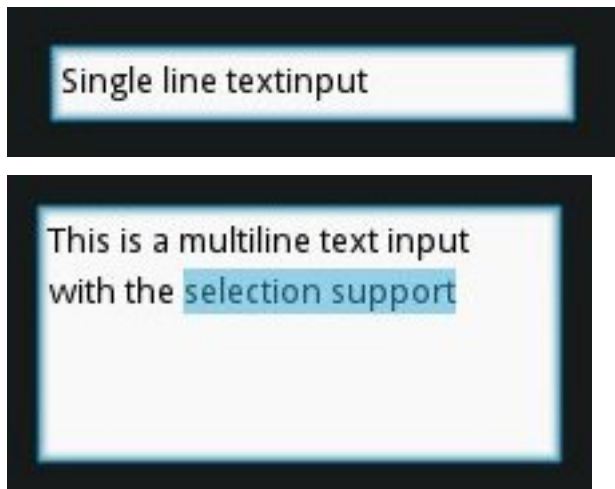
Bases: exceptions.Exception

The TabbedPanelException class.



# TEXT INPUT

New in version 1.0.4.



The `TextInput` widget provides a box of editable plain text.

Unicode, multiline, cursor navigation, selection and clipboard features are supported.

**Note:** Two different coordinate systems are used with `TextInput`:

- (x, y) - coordinates in pixels, mostly used for rendering on screen.
- (row, col) - cursor index in characters / lines, used for selection and cursor movement.

## 169.1 Usage example

To create a multiline textinput ('enter' key adds a new line):

```
from kivy.uix.textinput import TextInput
textinput = TextInput(text='Hello world')
```

To create a singleline textinput, set the multiline property to False ('enter' key will defocus the textinput and emit `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 on 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 'focus' 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)
```

## 169.2 Selection

The selection is automatically updated when the cursor position changes. You can get the currently selected text from the `TextInput.selection_text` property.

## 169.3 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 to 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)

```

## 169.4 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, Right
Control + c	Copy selection
Control + x	Cut selection
Control + p	Paste selection
Control + a	Select all the content
Control + z	undo
Control + r	redo

`class kivy.uix.textinput.TextInput(**kwargs)`

Bases: `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()`.

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\_active**

Background image of the `TextInput` when it's in focus and disabled.

New in version 1.8.0.

`background_disabled_active` is a `StringProperty` and defaults to `'atlas://data/images/defaulttheme/textinput_disabled_active'`.

### **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'`.

### **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: (top, right, bottom, 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 blink the cursor graphic. The value of **cursor\_blink** is automatically computed. Setting a value on it will have no impact.

**cursor\_blink** is a **BooleanProperty** and defaults to False.

### **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 1.8.1.

**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.

### **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*)

Move the cursor relative to it's 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

**Warning:** Current page has three lines before/after.

### **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.

### **focus**

If focus is True, the keyboard will be requested and you can start entering text into the `textInput`.

`focus` is a `BooleanProperty` and defaults to False.

---

**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`).

---

### **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 'DroidSans'.

### **font\_size**

Font size of the text in pixels.

`font_size` is a `NumericProperty` and defaults to 10.

### **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.

#### **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'`.

#### **hide\_keyboard**()

Convenience function to hide the keyboard in managed mode

#### **hint\_text**

Hint text of the widget.

Shown if text is "" and focus is False.

New in version 1.6.0.

`hint_text` a `StringProperty` 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 1.8.1.

`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 parameter; 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.

#### **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', 'address'.

#### **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\_mode**

How the keyboard visibility should be managed (auto will have standard behaviour to show/hide on focus, managed requires setting `keyboard_visible` manually, or calling the helper functions `show_keyboard()` and `hide_keyboard()`).

New in version 1.8.0.

`keyboard_mode` is an `OptionsProperty` and defaults to 'auto'. Can be one of 'auto' or 'managed'.

#### **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` 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`.

#### **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\_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 '\*'.

New in version 1.2.0.

`password` is a `BooleanProperty` and defaults to False.

### **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.

### **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.

### **show\_keyboard()**

Convenience function to show the keyboard in managed mode

### **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` a `StringProperty`.



**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.



# TOGGLE BUTTON

The `ToggleButton` widget acts like a checkbox. When you touch/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.ToggleButtonBehavior`, `kivy.uix.button.Button`

Toggle button class, see module documentation for more information.



---

# 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.

## 171.1 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)

        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'))
```

## 171.2 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:

$$\text{treeview.width} - \text{treeview.indent\_start} - \text{treeview.indent\_level} * \text{node.level}$$

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.

Returnsthe node *node*.

**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 **kivy.properties.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 `kivy.properties.NumericProperty` and defaults to 0.

### **remove\_node(*node*)**

Removes a node from the tree.

New in version 1.0.7.

#### **Parameters**

***node***: instance of a `TreeViewNode` 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).

### **class kivy.uix.treeview.TreeViewException**

Bases: `exceptions.Exception`

Exception for errors in the `TreeView`.

### **class kivy.uix.treeview.TreeViewLabel(\**kwargs*)**

Bases: `kivy.uix.label.Label`, `kivy.uix.treeview.TreeViewNode`

Combines a `Label` and a `TreeViewNode` 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.TreeViewNode(**kwargs)
```

Bases: object

TreeViewNode 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`.

---

# 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*)

Change the position to a percentage of duration. Percentage must be a value between 0-1.

**Warning:** Calling `seek()` before the video is loaded has no impact.

New in version 1.2.0.

## 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 `'play'`.

## unload()

Unload the video. The playback will be stopped.

New in version 1.8.0.

## volume

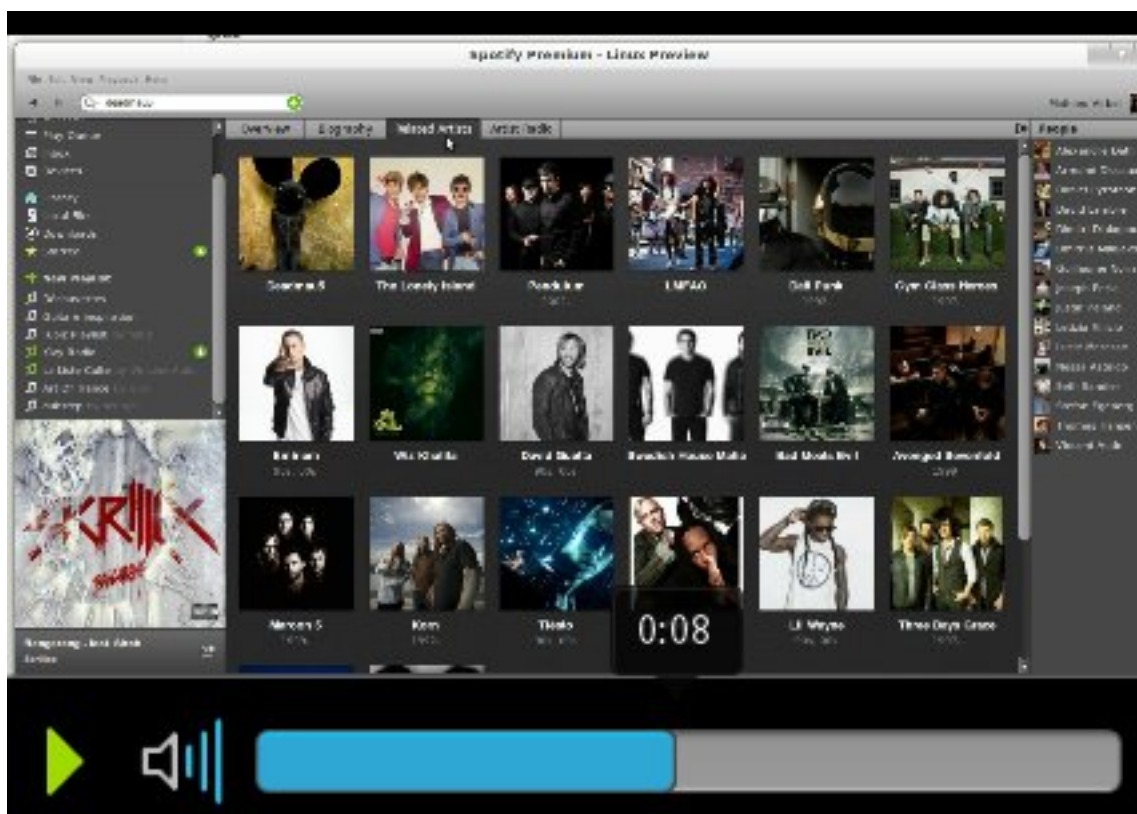
Volume of the video, in the range 0-1. 1 means full volume, 0 means mute.

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

## 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.



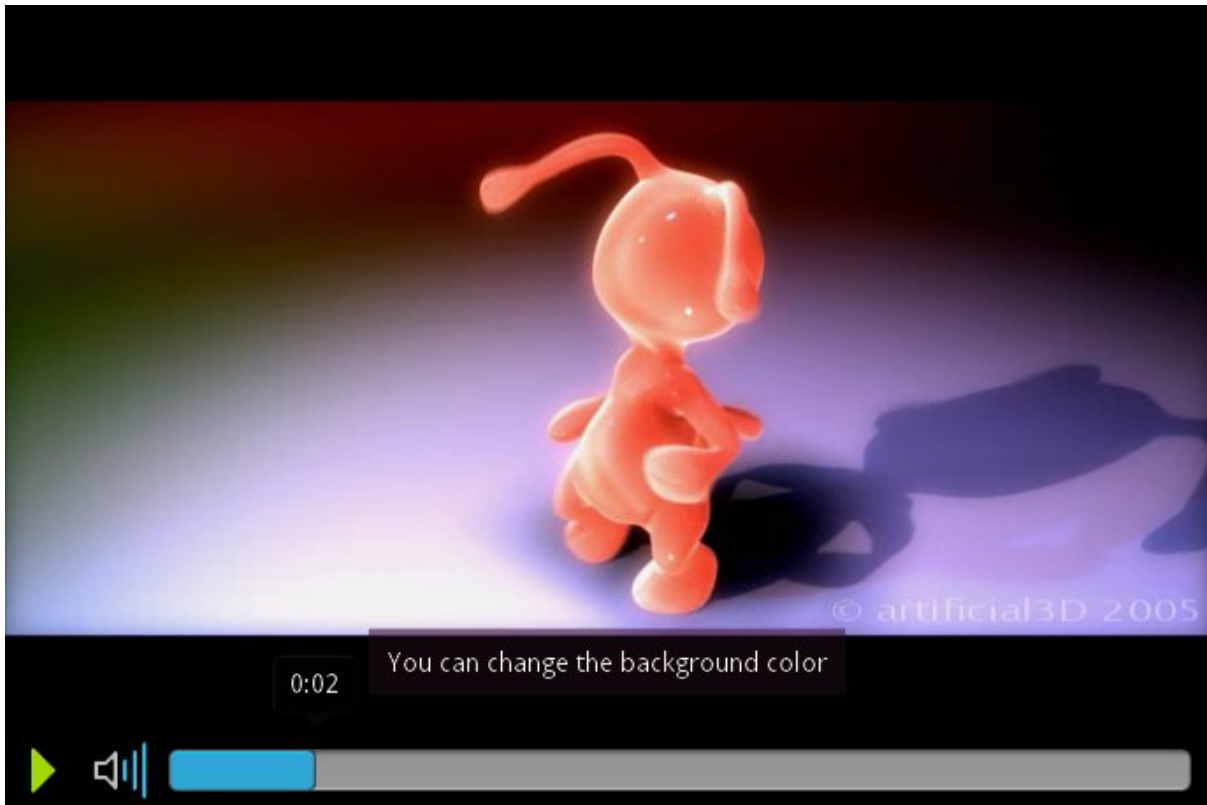
### 173.1 Annotations

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/softboy.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 softboy.avi example, the result will be:



If you want to experiment with annotation files, test with:

```
python -m kivy.uix.videoplayer examples/widgets/softboy.avi
```

## 173.2 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})
```

## 173.3 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 it's 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'`.

### **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)**

Change the position to a percentage of the duration. Percentage must be a value between 0-1.



**Warning:** Calling `seek()` before video is loaded has no effect.

### 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 'play'.

### thumbnail

Thumbnail of the video to show. If None, `VideoPlayer` will try to find the thumbnail from the `source` + '.png'.

`thumbnail` is 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
- `bgsouce`: '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.

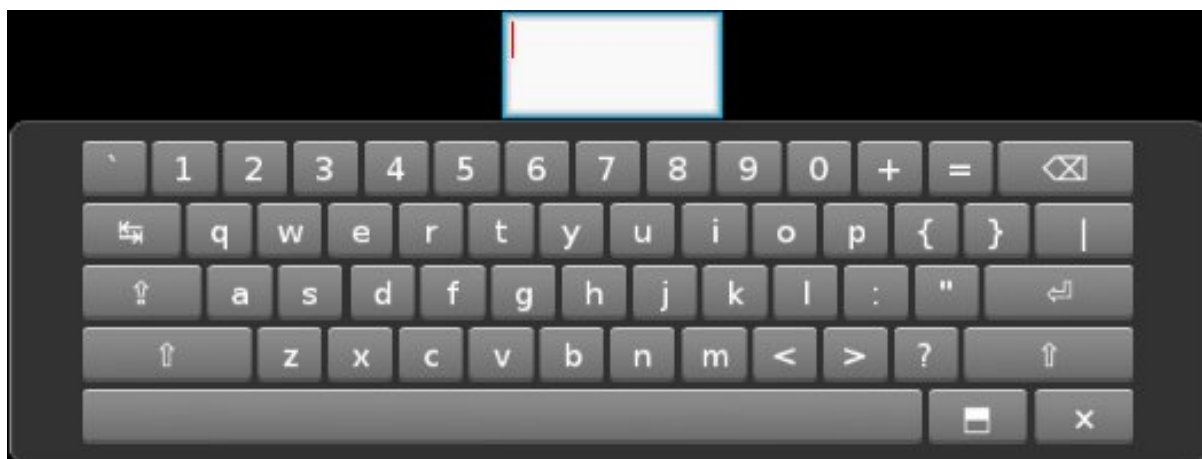
### start

Start time of the annotation.

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



# 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.

## 174.1 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).

## 174.2 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 1.8.1) 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
["\u21B9", "      ", "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],      ["\u232b", null, "backspace", 2]
  ],
  "shift_1": [ ... ],
  "normal_2": [ ... ],
  "special_2": [ ... ],
  ...
}
```

## 174.3 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:
    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** a `StringProperty` and defaults to `atlas://data/images/defaulttheme/vkeyboard`

#### 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 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.

### **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` 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` 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.

..versionadded:: 1.8.0

`key_disabled_background_normal` 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]

**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 find 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.





---

# WIDGET CLASS

The `Widget` class is the base class required to create a `Widget`. This widget class is 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_<proprname>'` callback. If nothing changes, nothing will be done. That's the main goal of the `Property` class.

**Separate 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 want to only trigger an action when the button itself is actually touched. For this, you can use the `Widget.collide_point()` method, which will return `True` if the point you pass 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 `Widget.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 buttons 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 size.
- `Widget.on_touch_down()`, `Widget.on_touch_move()`, `Widget.on_touch_up()` don't do any sort of collisions. If you want to know if the touch is inside your widget, use `Widget.collide_point()`.

## 175.1 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)

wid = Widget()
wid.bind(pos=callback_pos)
```

Read more about [Properties](#).

## 175.2 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.

## 175.3 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 most recently added widget and then backwards through its children (from the most recently added back to the first child). This order is the same for the `on_touch_move` and `on_touch_up` events.

If you want to reverse this order, you can raise events in the children before the parent by using the `super` command. For example:

```
class MyWidget(Widget):
    def on_touch_down(self, touch):
        super(MyWidget, self).on_touch_down(touch)
        # Do stuff here
```

In general, this would seldom be the best approach as every event bubbles all the way through event time and there is no way of determining if it has been handled. In order to stop this the event bubbling, one of these methods must return *True*. At this point, Kivy assumes the event has been handled and propagation stops.

This means that the recommended approach is to let the event bubble naturally but swallow the event if it has been handled. For example:

```
class MyWidget(Widget):
    def on_touch_down(self, touch):
        if <some_condition>:
            # Do stuff here and kill the event
            return True
        else:
            # Continue normal event bubbling
            return super(MyWidget, self).on_touch_down(touch)
```

This approach gives you good control over exactly how events are dispatched and managed.

```
class kivy.uix.widget.Widget(**kwargs)
    Bases: kivy.uix.widget.WidgetBase
```

Widget class. See module documentation for more information.

#### Events

*on\_touch\_down*: Fired when a new touch event occurs  
*on\_touch\_move*: Fired when an existing touch moves  
*on\_touch\_up*: Fired when an existing touch disappears

**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*)

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 0Index to insert the widget in the list

New in version 1.0.5.

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

**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)`

### **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 widgets added to this widget.

Changed in version 1.8.0: *children* argument can be used to select the children we want to remove. It should be a list of children (or filtered list) of the current widget.

### **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 boundingbox.

#### Parameters

*x*: **numeric**X position of the point (in window coordinates)

*y*: **numeric**Y position of the point (in window coordinates)

**Returns**bool, True if the point is inside the bounding box.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, 40)
True
```

### **collide\_widget** (*wid*)

Check if the other widget collides with this widget. Performs an axis-aligned bounding box intersection test by default.

#### Parameters

*wid*: **Widget** classWidget to collide with.

**Returns**bool, True if the other widget collides with this widget.

```
>>> 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.

---

### Note:

- 1.Child Widgets, when added to a disabled widget, will be disabled automatically.
  - 2.Disabling/enabling a parent disables/enables all of it's children.
- 

New in version 1.8.0.

`disabled` is a `BooleanProperty` and defaults to `False`.

## export\_to\_png(filename, \*args)

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 1.8.1.

## 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`.

## 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

Unique identifier of the widget in the tree.

`id` is a `StringProperty` and defaults to `None`.

**Warning:** If the `id` is already used in the tree, an exception will be raised.

## ids

This is a Dictionary of id's defined in your kv language. This will only be populated if you use id's in your kv language code.

New in version 1.7.0.

`ids` is a `DictProperty` and defaults to a 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:
  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 [relativelayout](#) for a discussion on coordinate systems.

**Returns** bool. If True, the dispatching of the touch event will stop.

#### **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 the 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.

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

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.

### 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, in percent (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.

Position hint is only used by the `FloatLayout` and `Window`.

`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`).

See [size\\_hint\\_x](#) for more information

## **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 value is in percent as a float from 0. to 1., where 1. means the full size of his parent. 0.5 represents 50%.

`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

## **to\_local**(*x, y, relative=False*)

Transform parent coordinates to local 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 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 given coordinate from window to local widget coordinates. See [relativelayout](#) for details on the coordinate systems.

## **to\_window**(*x, y, initial=True, relative=False*)

Transform local 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 1.8.1.

#### **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
  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 1.8.1.

### **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.

**class** `kivy.uix.widget.WidgetException`

Bases: `exceptions.Exception`

Fired when the widget gets an exception.

## UTILS

Changed in version 1.6.0: The `OrderedDict` class has been removed. Use the `collections.OrderedDict`.

`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)`

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: `list`

List with a `clear()` method.

**Warning:** Usage of the `iterate()` function will decrease your performance.

`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: `dict`

`QueryDict` is a `dict()` that can be queried with `dot`.

New in version 1.0.4.

```
d = QueryDict()
# create a key named toto, with the value 1
d.toto = 1
# it's the same as
d['toto'] = 1
```

`kivy.utils.platform = platform name: 'linux' from: <kivy.utils.Platform object at 0x90ae7ac>`

New in version 1.3.0.

Deprecated since 1.8.0: Use `platform` as variable instead of a function.

Calling `platform()` will return one of: *win, linux, android, macosx, ios* or *unknown*.

Changed in version 1.8.0.

`platform` also behaves like a regular variable in comparisons like so:

```
from kivy import platform
if platform == 'linux':
    do_linux_things()
if platform() == 'linux': # triggers deprecation warning
    do_more_linux_things()
foo = {'linux' : do_linux_things}
foo[platform]() # calls do_linux_things
p = platform # assigns to a module object
if p is 'android':
    do_android_things()
p += 'some string' # error!
```

`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: 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](#).



# VECTOR

The `Vector` represents a 2D vector (x, y). Our implementation is made on top of a Python list.

Example for 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
>>> v[1]
45
>>> v.y
45
```

## 177.1 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))
```

## 177.2 Vector operators

The `Vector` supports some numeric operators like `+`, `-`, `/`:

```
>>> 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 do in-place operations:

```
>>> v = Vector(1, 1)
>>> v += 2
>>> v
[3, 3]
>>> v *= 5
[15, 15]
>>> v /= 2.
[7.5, 7.5]
```

`class kivy.vector.Vector(*largs)`

Bases: `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](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)
>>> v
[70.710678118654755, 70.710678118654741]
```

**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
```

## WEAK METHOD

The **WeakMethod** is used in the Clock class to allow a reference to a bound method that permits the associated object to be garbage collected. Check `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. (thanks to him !)

```
class kivy.weakmethod.WeakMethod(method)
```

```
    Bases: 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.



## 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 1.7.2 and 1.8 are now under MIT License. Previous version are still under LGPL 3 license.

Kivy is released under the terms of the MIT License. You should have received a copy of the MIT alongside your Kivy distribution. See the file LICENSE 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 2010-2014, The Kivy Authors. All rights reserved.





# PYTHON MODULE INDEX

k

- kivy, 159
- kivy.adapters, 251
  - kivy.adapters.adapter, 259
  - kivy.adapters.args\_converters, 261
  - kivy.adapters.dictadapter, 263
  - kivy.adapters.listadapter, 265
  - kivy.adapters.models, 269
  - kivy.adapters.simplelistadapter, 271
- kivy.animation, 273
- kivy.app, 285
- kivy.atlas, 297
- kivy.base, 301
- kivy.cache, 305
- kivy.clock, 307
- kivy.compat, 313
- kivy.config, 315
- kivy.context, 321
- kivy.core, 323
  - kivy.core.audio, 345
  - kivy.core.camera, 347
  - kivy.core.clipboard, 349
  - kivy.core.gl, 351
  - kivy.core.image, 353
  - kivy.core.spelling, 357
  - kivy.core.text, 359
    - kivy.core.text.markup, 367
    - kivy.core.text.text\_layout, 369
  - kivy.core.video, 373
  - kivy.core.window, 375
- kivy.effects, 383
  - kivy.effects.dampedscroll, 387
  - kivy.effects.kinetic, 389
  - kivy.effects.opacityscroll, 391
  - kivy.effects.scroll, 393
- kivy.event, 395
- kivy.ext, 401
- kivy.factory, 403
- kivy.garden, 405
- kivy.geometry, 407
- kivy.gesture, 409
- kivy.graphics, 411
  - kivy.graphics.compiler, 473
  - kivy.graphics.context, 475
  - kivy.graphics.context\_instructions, 477
  - kivy.graphics.fbo, 481
  - kivy.graphics.gl\_instructions, 485
  - kivy.graphics.instructions, 487
  - kivy.graphics.opengl, 493
  - kivy.graphics.opengl\_utils, 503
  - kivy.graphics.shader, 505
  - kivy.graphics.stencil\_instructions, 507
  - kivy.graphics.texture, 509
  - kivy.graphics.transformation, 515
  - kivy.graphics.vertex\_instructions, 464
- kivy.input, 519
  - kivy.input.factory, 537
  - kivy.input.motionevent, 539
  - kivy.input.postproc, 545
    - kivy.input.postproc.calibration, 549
    - kivy.input.postproc.dejitter, 551
    - kivy.input.postproc.doubletap, 553
    - kivy.input.postproc.ignorelist, 555
    - kivy.input.postproc.retain\_touch, 557
    - kivy.input.postproc.tripletap, 559
  - kivy.input.provider, 561
  - kivy.input.providers, 563
    - kivy.input.providers.androidjoystick, 569
    - kivy.input.providers.hidinput, 571
    - kivy.input.providers.leapfinger, 573
    - kivy.input.providers.linuxwacom, 575
    - kivy.input.providers.mactouch, 577
    - kivy.input.providers.mouse, 579
    - kivy.input.providers.mtdev, 581
    - kivy.input.providers.probesysfs, 583
    - kivy.input.providers.tuio, 585
    - kivy.input.providers.wm\_common, 587
    - kivy.input.providers.wm\_pen, 589
    - kivy.input.providers.wm\_touch, 591
  - kivy.input.recorder, 593
  - kivy.input.shape, 597
- kivy.interactive, 599
- kivy.lang, 603
- kivy.lib, 617

kivy.lib.gstplayer, 619  
kivy.loader, 621  
kivy.logger, 625  
kivy.metrics, 627  
kivy.modules, 631  
kivy.modules.inspector, 637  
kivy.modules.keybinding, 639  
kivy.modules.monitor, 641  
kivy.modules.recorder, 643  
kivy.modules.screen, 645  
kivy.modules.touchring, 647  
kivy.modules.webdebugger, 649  
kivy.multistroke, 651  
kivy.network, 661  
kivy.network.urlrequest, 665  
kivy.parser, 669  
kivy.properties, 671  
kivy.resources, 683  
kivy.storage, 685  
kivy.storage.dictstore, 691  
kivy.storage.jsonstore, 693  
kivy.storage.redisstore, 695  
kivy.support, 697  
kivy.uix, 699  
kivy.uix.abstractview, 855  
kivy.uix.accordion, 857  
kivy.uix.actionbar, 863  
kivy.uix.anchorlayout, 867  
kivy.uix.behaviors, 869  
kivy.uix.boxlayout, 877  
kivy.uix.bubble, 879  
kivy.uix.button, 883  
kivy.uix.camera, 885  
kivy.uix.carousel, 887  
kivy.uix.checkbox, 891  
kivy.uix.codeinput, 893  
kivy.uix.colorpicker, 895  
kivy.uix.dropdown, 897  
kivy.uix.effectwidget, 901  
kivy.uix.filechooser, 905  
kivy.uix.floatlayout, 911  
kivy.uix.gesturesurface, 913  
kivy.uix.gridlayout, 917  
kivy.uix.image, 923  
kivy.uix.label, 927  
kivy.uix.layout, 935  
kivy.uix.listview, 937  
kivy.uix.modalview, 949  
kivy.uix.pagelayout, 953  
kivy.uix.popup, 955  
kivy.uix.progressbar, 959  
kivy.uix.relativelayout, 961  
kivy.uix.rst, 965  
kivy.uix.sandbox, 969  
kivy.uix.scatter, 971  
kivy.uix.scatterlayout, 975  
kivy.uix.screenmanager, 977  
kivy.uix.scrollview, 985  
kivy.uix.settings, 991  
kivy.uix.slider, 999  
kivy.uix.spinner, 1003  
kivy.uix.splitter, 1005  
kivy.uix.stacklayout, 1007  
kivy.uix.stencilview, 1011  
kivy.uix.switch, 1013  
kivy.uix.tabbedpanel, 1015  
kivy.uix.textinput, 1023  
kivy.uix.togglebutton, 1035  
kivy.uix.treeview, 1037  
kivy.uix.video, 1043  
kivy.uix.videoplayer, 1045  
kivy.uix.vkeyboard, 1051  
kivy.uix.widget, 1057  
kivy.utils, 1067  
kivy.vector, 1071  
kivy.weakmethod, 1075

# INDEX

## A

- a (kivy.graphics.ClearColor attribute), 428
- a (kivy.graphics.Color attribute), 415
- a (kivy.graphics.context\_instructions.Color attribute), 435, 478
- a (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485
- a (kivy.uix.colorpicker.ColorWheel attribute), 728, 896
- AbstractStore (class in kivy.storage), 687
- AbstractView (class in kivy.uix.abstractview), 699, 855
- accept\_stroke() (kivy.uix.gesturesurface.GestureContainer method), 744, 915
- Accordion (class in kivy.uix.accordion), 701, 858
- accordion (kivy.uix.accordion.AccordionItem attribute), 702, 859
- AccordionException (class in kivy.uix.accordion), 703, 861
- AccordionItem (class in kivy.uix.accordion), 702, 859
- action\_previous (kivy.uix.actionbar.ActionView attribute), 706, 865
- action\_view (kivy.uix.actionbar.ActionBar attribute), 707, 866
- ActionBar (class in kivy.uix.actionbar), 707, 866
- ActionBarException (class in kivy.uix.actionbar), 704, 863
- ActionButton (class in kivy.uix.actionbar), 705, 864
- ActionCheck (class in kivy.uix.actionbar), 705, 864
- ActionDropDown (class in kivy.uix.actionbar), 705, 864
- ActionGroup (class in kivy.uix.actionbar), 705, 864
- ActionItem (class in kivy.uix.actionbar), 704, 863
- ActionOverflow (class in kivy.uix.actionbar), 706, 865
- ActionPrevious (class in kivy.uix.actionbar), 706, 866
- ActionSeparator (class in kivy.uix.actionbar), 705, 864
- ActionToggleButton (class in kivy.uix.actionbar), 705, 864
- ActionView (class in kivy.uix.actionbar), 706, 865
- active (kivy.uix.checkbox.CheckBox attribute), 726, 891
- active (kivy.uix.switch.Switch attribute), 809, 1013
- active\_norm\_pos (kivy.uix.switch.Switch attribute), 809, 1013
- Adapter (class in kivy.adapters.adapter), 252, 259
- adapter (kivy.uix.abstractview.AbstractView attribute), 699, 855
- add() (kivy.graphics.InstructionGroup method), 418
- add() (kivy.graphics.instructions.InstructionGroup method), 430, 487
- add\_callback() (kivy.config.ConfigParser method), 196, 318
- add\_event\_listener() (kivy.base.EventLoopBase method), 208, 301
- add\_gesture() (kivy.gesture.GestureDatabase method), 204, 410
- add\_gesture() (kivy.multistroke.Recognizer method), 226, 653
- add\_handler() (kivy.base.ExceptionManagerBase method), 209, 302
- add\_input\_provider() (kivy.base.EventLoopBase method), 208, 301
- add\_interface() (kivy.uix.settings.Settings method), 795, 993
- add\_json\_panel() (kivy.uix.settings.Settings method), 795, 993
- add\_kivy\_panel() (kivy.uix.settings.Settings method), 796, 993
- add\_mipmap() (kivy.core.image.ImageData method), 328, 355
- add\_node() (kivy.uix.treeview.TreeView method), 827, 1038
- add\_panel() (kivy.uix.settings.ContentPanel method), 799, 997
- add\_panel() (kivy.uix.settings.InterfaceWithSidebar method), 799, 997
- add\_point() (kivy.gesture.GestureStroke method), 204, 410
- add\_point() (kivy.graphics.Point method), 423
- add\_point() (kivy.graphics.vertex\_instructions.Point method), 469
- add\_point() (kivy.multistroke.UnistrokeTemplate

method), 231, 658  
 add\_postproc\_module() (kivy.base.EventLoopBase method), 208, 301  
 add\_reload\_observer() (kivy.graphics.context.Context method), 438, 475  
 add\_reload\_observer() (kivy.graphics.Fbo method), 417  
 add\_reload\_observer() (kivy.graphics.fbo.Fbo method), 439, 482  
 add\_reload\_observer() (kivy.graphics.texture.Texture method), 458, 512  
 add\_screen() (kivy.uix.screenmanager.TransitionBase method), 786, 981  
 add\_stroke() (kivy.gesture.Gesture method), 204, 409  
 add\_stroke() (kivy.multistroke.Candidate method), 231, 658  
 add\_stroke() (kivy.multistroke.MultistrokeGesture method), 230, 657  
 add\_stroke() (kivy.uix.gesturesurface.GestureContainer method), 744, 915  
 add\_widget() (kivy.core.window.WindowBase method), 339, 377  
 add\_widget() (kivy.uix.widget.Widget method), 844, 1059  
 adddefaultsection() (kivy.config.ConfigParser method), 196, 318  
 AdvancedEffectBase (class in kivy.uix.effectwidget), 733, 903  
 after (kivy.graphics.Canvas attribute), 414  
 after (kivy.graphics.instructions.Canvas attribute), 432, 489  
 AliasProperty (class in kivy.properties), 239, 678  
 allow\_copy (kivy.uix.textinput.TextInput attribute), 818, 1025  
 allow\_empty\_selection (kivy.adapters.listadapter.ListAdapter attribute), 254, 265  
 allow\_fullscreen (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 allow\_stretch (kivy.uix.image.Image attribute), 749, 924  
 anchor\_x (kivy.uix.anchorlayout.AnchorLayout attribute), 708, 867  
 anchor\_y (kivy.uix.anchorlayout.AnchorLayout attribute), 708, 867  
 AnchorLayout (class in kivy.uix.anchorlayout), 708, 867  
 anchors (kivy.core.text.markup.MarkupLabel attribute), 333, 362, 367  
 anchors (kivy.uix.label.Label attribute), 753, 929  
 angle (kivy.graphics.context\_instructions.Rotate attribute), 436, 479  
 angle (kivy.graphics.Rotate attribute), 425  
 angle() (kivy.vector.Vector method), 247, 1072  
 angle\_end (kivy.graphics.Ellipse attribute), 416  
 angle\_end (kivy.graphics.vertex\_instructions.Ellipse attribute), 465  
 angle\_start (kivy.graphics.Ellipse attribute), 416  
 angle\_start (kivy.graphics.vertex\_instructions.Ellipse attribute), 465  
 anim\_available (kivy.core.image.Image attribute), 326, 353  
 anim\_cancel\_duration (kivy.uix.carousel.Carousel attribute), 724, 887  
 anim\_delay (kivy.core.image.Image attribute), 326, 353  
 anim\_delay (kivy.uix.image.Image attribute), 749, 924  
 anim\_duration (kivy.uix.accordion.Accordion attribute), 701, 858  
 anim\_func (kivy.uix.accordion.Accordion attribute), 701, 858  
 anim\_index (kivy.core.image.Image attribute), 326, 353  
 anim\_move\_duration (kivy.uix.carousel.Carousel attribute), 724, 887  
 anim\_reset() (kivy.core.image.Image method), 326, 353  
 anim\_type (kivy.uix.carousel.Carousel attribute), 724, 887  
 animated\_properties (kivy.animation.Animation attribute), 161, 274  
 Animation (class in kivy.animation), 161, 274  
 AnimationTransition (class in kivy.animation), 162, 275  
 annotations (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 App (class in kivy.app), 176, 290  
 app\_icon (kivy.uix.actionbar.ActionPrevious attribute), 706, 866  
 append() (kivy.cache.Cache static method), 186, 305  
 apply() (kivy.lang.BuilderBase method), 220, 614  
 apply\_transform() (kivy.uix.scatter.Scatter method), 779, 972  
 apply\_transform\_2d() (kivy.input.MotionEvent method), 519  
 apply\_transform\_2d() (kivy.input.motionevent.MotionEvent method), 532, 540  
 ApplyContextMatrix (class in kivy.graphics), 429  
 args\_converter (kivy.adapters.adapter.Adapter attribute), 252, 259  
 arrow\_image (kivy.uix.bubble.Bubble attribute), 719, 881  
 arrow\_pos (kivy.uix.bubble.Bubble attribute), 720,

881  
 ask\_update() (kivy.graphics.Callback method), 413  
 ask\_update() (kivy.graphics.Canvas method), 414  
 ask\_update() (kivy.graphics.instructions.Callback method), 434, 491  
 ask\_update() (kivy.graphics.instructions.Canvas method), 432, 489  
 ask\_update() (kivy.graphics.texture.Texture method), 459, 512  
 async\_clear() (kivy.storage.AbstractStore method), 687  
 async\_count() (kivy.storage.AbstractStore method), 687  
 async\_delete() (kivy.storage.AbstractStore method), 687  
 async\_exists() (kivy.storage.AbstractStore method), 687  
 async\_find() (kivy.storage.AbstractStore method), 687  
 async\_get() (kivy.storage.AbstractStore method), 687  
 async\_keys() (kivy.storage.AbstractStore method), 687  
 async\_put() (kivy.storage.AbstractStore method), 687  
 AsyncImage (class in kivy.uix.image), 751, 925  
 Atlas (class in kivy.atlas), 185, 299  
 attach\_to (kivy.uix.dropdown.DropDown attribute), 730, 898  
 attach\_to (kivy.uix.modalview.ModalView attribute), 770, 950  
 auto\_bring\_to\_front (kivy.uix.scatter.Scatter attribute), 780, 973  
 auto\_dismiss (kivy.uix.dropdown.DropDown attribute), 730, 898  
 auto\_dismiss (kivy.uix.modalview.ModalView attribute), 770, 950  
 auto\_indent (kivy.uix.textinput.TextInput attribute), 818, 1026  
 auto\_width (kivy.uix.dropdown.DropDown attribute), 730, 899  
 available\_layouts (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 axis (kivy.graphics.context\_instructions.Rotate attribute), 436, 479  
 axis (kivy.graphics.Rotate attribute), 425  
**B**  
 b (kivy.graphics.ClearColor attribute), 428  
 b (kivy.graphics.Color attribute), 415  
 b (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 b (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485  
 b (kivy.uix.colorpicker.ColorWheel attribute), 728, 896  
 background (kivy.uix.modalview.ModalView attribute), 771, 950  
 background (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 background\_active (kivy.uix.textinput.TextInput attribute), 818, 1026  
 background\_border (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 background\_color (kivy.uix.actionbar.ActionBar attribute), 707, 866  
 background\_color (kivy.uix.actionbar.ActionView attribute), 706, 865  
 background\_color (kivy.uix.bubble.Bubble attribute), 720, 881  
 background\_color (kivy.uix.button.Button attribute), 721, 883  
 background\_color (kivy.uix.effectwidget.EffectWidget attribute), 733, 902  
 background\_color (kivy.uix.listview.CompositeListItem attribute), 768, 946  
 background\_color (kivy.uix.modalview.ModalView attribute), 771, 950  
 background\_color (kivy.uix.rst.RstDocument attribute), 851, 966  
 background\_color (kivy.uix.tabbedpanel.TabbedPanel attribute), 812, 1018  
 background\_color (kivy.uix.textinput.TextInput attribute), 818, 1026  
 background\_color (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 background\_disabled (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 background\_disabled\_active (kivy.uix.textinput.TextInput attribute), 818, 1026  
 background\_disabled\_down (kivy.uix.button.Button attribute), 721, 884  
 background\_disabled\_image (kivy.uix.tabbedpanel.TabbedPanel attribute), 812, 1018  
 background\_disabled\_normal (kivy.uix.accordion.AccordionItem attribute), 702, 859  
 background\_disabled\_normal (kivy.uix.button.Button attribute), 721, 884  
 background\_disabled\_normal (kivy.uix.textinput.TextInput attribute), 818, 1026  
 background\_disabled\_selected (kivy.uix.accordion.AccordionItem



attribute), 702, 859  
 background\_down (kivy.uix.actionbar.ActionItem attribute), 704, 863  
 background\_down (kivy.uix.button.Button attribute), 722, 884  
 background\_image (kivy.uix.actionbar.ActionBar attribute), 707, 866  
 background\_image (kivy.uix.actionbar.ActionSeparator attribute), 705, 864  
 background\_image (kivy.uix.actionbar.ActionView attribute), 706, 865  
 background\_image (kivy.uix.bubble.Bubble attribute), 720, 881  
 background\_image (kivy.uix.tabbedpanel.StripLayout attribute), 812, 1018  
 background\_image (kivy.uix.tabbedpanel.TabbedPanel attribute), 812, 1018  
 background\_normal (kivy.uix.accordion.AccordionItem attribute), 702, 859  
 background\_normal (kivy.uix.actionbar.ActionItem attribute), 704, 863  
 background\_normal (kivy.uix.button.Button attribute), 722, 884  
 background\_normal (kivy.uix.textinput.TextInput attribute), 818, 1026  
 background\_selected (kivy.uix.accordion.AccordionItem attribute), 702, 859  
 bar\_color (kivy.uix.scrollview.ScrollView attribute), 790, 986  
 bar\_inactive\_color (kivy.uix.scrollview.ScrollView attribute), 790, 986  
 bar\_margin (kivy.uix.scrollview.ScrollView attribute), 790, 986  
 bar\_pos (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 bar\_pos\_x (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 bar\_pos\_y (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 bar\_width (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 base\_font\_size (kivy.uix.rst.RstDocument attribute), 852, 966  
 bbox (kivy.uix.scatter.Scatter attribute), 780, 973  
 before (kivy.graphics.Canvas attribute), 414  
 before (kivy.graphics.instructions.Canvas attribute), 432, 489  
 best (kivy.multistroke.ProgressTracker attribute), 229, 656  
 Bezier (class in kivy.graphics), 412  
 Bezier (class in kivy.graphics.vertex\_instructions), 470  
 bezier (kivy.graphics.Line attribute), 420  
 bezier (kivy.graphics.vertex\_instructions.Line attribute), 467  
 bezier\_precision (kivy.graphics.Line attribute), 420  
 bezier\_precision (kivy.graphics.vertex\_instructions.Line attribute), 467  
 bind() (kivy.event.EventDispatcher method), 197, 395  
 bind() (kivy.graphics.Fbo method), 417  
 bind() (kivy.graphics.fbo.Fbo method), 439, 482  
 bind() (kivy.graphics.texture.Texture method), 459, 512  
 bind() (kivy.lang.Observable method), 219, 613  
 bind() (kivy.properties.Property method), 236, 674  
 BindTexture (class in kivy.graphics), 412  
 BindTexture (class in kivy.graphics.context\_instructions), 435, 478  
 blit\_buffer() (kivy.graphics.texture.Texture method), 459, 512  
 blit\_data() (kivy.graphics.texture.Texture method), 459, 512  
 bold (kivy.uix.label.Label attribute), 754, 930  
 BooleanProperty (class in kivy.properties), 238, 676  
 border (kivy.graphics.BorderImage attribute), 413  
 border (kivy.graphics.vertex\_instructions.BorderImage attribute), 465  
 border (kivy.uix.actionbar.ActionBar attribute), 707, 866  
 border (kivy.uix.bubble.Bubble attribute), 720, 881  
 border (kivy.uix.button.Button attribute), 722, 884  
 border (kivy.uix.modalview.ModalView attribute), 771, 950  
 border (kivy.uix.pagelayout.PageLayout attribute), 772, 953  
 border (kivy.uix.splitter.Splitter attribute), 804, 1006  
 border (kivy.uix.tabbedpanel.StripLayout attribute), 812, 1018  
 border (kivy.uix.tabbedpanel.TabbedPanel attribute), 812, 1018  
 border (kivy.uix.textinput.TextInput attribute), 818, 1026  
 BorderImage (class in kivy.graphics), 413  
 BorderImage (class in kivy.graphics.vertex\_instructions), 465  
 boundary() (in module kivy.utils), 244, 1067  
 BoundedNumericProperty (class in kivy.properties), 238, 676  
 bounds (kivy.properties.BoundedNumericProperty attribute), 238, 677  
 BoxLayout (class in kivy.uix.boxlayout), 717, 878  
 Bubble (class in kivy.uix.bubble), 719, 881  
 BubbleButton (class in kivy.uix.bubble), 720, 882  
 bufferfmt (kivy.graphics.texture.Texture attribute), 459, 512

build() (kivy.app.App method), 176, 291  
 build\_config() (kivy.app.App method), 176, 291  
 build\_settings() (kivy.app.App method), 176, 291  
 Builder (in module kivy.lang), 219, 613  
 BuilderBase (class in kivy.lang), 220, 613  
 BuilderException (class in kivy.lang), 220, 614  
 Button (class in kivy.uix.button), 721, 883  
 ButtonBehavior (class in kivy.uix.behaviors), 709, 869

## C

Cache (class in kivy.cache), 186, 305  
 cached\_views (kivy.adapters.listadapter.ListAdapter attribute), 254, 265  
 Callback (class in kivy.graphics), 413  
 Callback (class in kivy.graphics.instructions), 433, 490  
 callback (kivy.core.window.Keyboard attribute), 338, 376  
 callback (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
 Camera (class in kivy.uix.camera), 722, 885  
 CameraBase (class in kivy.core.camera), 325, 347  
 cancel() (kivy.animation.Animation method), 161, 274  
 cancel() (kivy.clock.ClockEvent method), 192, 311  
 cancel() (kivy.effects.kinetic.KineticEffect method), 384, 389  
 cancel() (kivy.uix.filechooser.FileChooserController method), 738, 908  
 cancel() (kivy.uix.filechooser.FileChooserProgressBase method), 740, 910  
 cancel\_all() (kivy.animation.Animation static method), 162, 274  
 cancel\_property() (kivy.animation.Animation method), 162, 275  
 cancel\_selection() (kivy.uix.textinput.TextInput method), 818, 1026  
 Candidate (class in kivy.multistroke), 231, 658  
 Canvas (class in kivy.graphics), 414  
 Canvas (class in kivy.graphics.instructions), 431, 489  
 canvas (kivy.uix.widget.Widget attribute), 844, 1059  
 CanvasBase (class in kivy.graphics), 415  
 CanvasBase (class in kivy.graphics.instructions), 432, 490  
 cap (kivy.graphics.Line attribute), 420  
 cap (kivy.graphics.vertex\_instructions.Line attribute), 467  
 cap\_precision (kivy.graphics.Line attribute), 420  
 cap\_precision (kivy.graphics.vertex\_instructions.Line attribute), 467  
 Carousel (class in kivy.uix.carousel), 724, 887  
 center (kivy.core.window.WindowBase attribute), 339, 377  
 center (kivy.uix.widget.Widget attribute), 844, 1060  
 center\_stroke() (kivy.gesture.GestureStroke method), 204, 410  
 center\_x (kivy.uix.widget.Widget attribute), 844, 1060  
 center\_y (kivy.uix.widget.Widget attribute), 844, 1060  
 ChangeState (class in kivy.graphics), 429  
 ChannelMixEffect (class in kivy.uix.effectwidget), 734, 904  
 check() (kivy.core.spelling.SpellingBase method), 329, 357  
 CheckBox (class in kivy.uix.checkbox), 726, 891  
 children (kivy.core.window.WindowBase attribute), 339, 377  
 children (kivy.uix.widget.Widget attribute), 844, 1060  
 chunk\_size (kivy.network.urlrequest.UrlRequest attribute), 662, 666  
 circle (kivy.graphics.Line attribute), 420  
 circle (kivy.graphics.vertex\_instructions.Line attribute), 467  
 circumcircle() (in module kivy.geometry), 203, 407  
 clear() (kivy.core.window.WindowBase method), 339, 377  
 clear() (kivy.graphics.Canvas method), 414  
 clear() (kivy.graphics.InstructionGroup method), 418  
 clear() (kivy.graphics.instructions.Canvas method), 432, 489  
 clear() (kivy.graphics.instructions.InstructionGroup method), 430, 487  
 clear() (kivy.storage.AbstractStore method), 687  
 clear\_buffer() (kivy.graphics.Fbo method), 417  
 clear\_buffer() (kivy.graphics.fbo.Fbo method), 440, 482  
 clear\_color (kivy.graphics.ClearBuffers attribute), 428  
 clear\_color (kivy.graphics.Fbo attribute), 417  
 clear\_color (kivy.graphics.fbo.Fbo attribute), 440, 483  
 clear\_color (kivy.graphics.gl\_instructions.ClearBuffers attribute), 441, 486  
 clear\_depth (kivy.graphics.ClearBuffers attribute), 428  
 clear\_depth (kivy.graphics.gl\_instructions.ClearBuffers attribute), 441, 486  
 clear\_selection() (kivy.uix.behaviors.CompoundSelectionBehavior method), 713, 874  
 clear\_stencil (kivy.graphics.ClearBuffers attribute), 428  
 clear\_stencil (kivy.graphics.gl\_instructions.ClearBuffers attribute), 441, 486  
 clear\_widgets() (kivy.uix.widget.Widget method),

[844, 1060](#)  
 ClearBuffers (class in kivy.graphics), [428](#)  
 ClearBuffers (class in kivy.graphics.gl\_instructions), [441, 485](#)  
 ClearColor (class in kivy.graphics), [428](#)  
 ClearColor (class in kivy.graphics.gl\_instructions), [441, 485](#)  
 clearcolor (kivy.core.window.WindowBase attribute), [339, 377](#)  
 clearcolor (kivy.uix.screenmanager.ShaderTransition attribute), [787, 982](#)  
 Clock (in module kivy.clock), [190, 310](#)  
 ClockBase (class in kivy.clock), [190, 310](#)  
 ClockEvent (class in kivy.clock), [192, 311](#)  
 close (kivy.graphics.Line attribute), [421](#)  
 close (kivy.graphics.vertex\_instructions.Line attribute), [467](#)  
 close() (kivy.base.EventLoopBase method), [208, 301](#)  
 close() (kivy.core.window.WindowBase method), [339, 377](#)  
 close\_settings() (kivy.app.App method), [176, 291](#)  
 cls (kivy.adapters.adapter.Adapter attribute), [252, 259](#)  
 cls (kivy.uix.widget.Widget attribute), [844, 1060](#)  
 cm() (in module kivy.metrics), [224, 629](#)  
 CodeInput (class in kivy.uix.codeinput), [727, 893](#)  
 col\_default\_width (kivy.uix.gridlayout.GridLayout attribute), [747, 919](#)  
 col\_force\_default (kivy.uix.gridlayout.GridLayout attribute), [747, 919](#)  
 collapse (kivy.uix.accordion.AccordionItem attribute), [702, 859](#)  
 collapse\_alpha (kivy.uix.accordion.AccordionItem attribute), [702, 859](#)  
 collide\_margin() (kivy.uix.vkeyboard.VKeyboard method), [833, 1053](#)  
 collide\_point() (kivy.uix.widget.Widget method), [845, 1060](#)  
 collide\_widget() (kivy.uix.widget.Widget method), [845, 1060](#)  
 Color (class in kivy.graphics), [415](#)  
 Color (class in kivy.graphics.context\_instructions), [434, 477](#)  
 color (kivy.uix.colorpicker.ColorPicker attribute), [728, 895](#)  
 color (kivy.uix.colorpicker.ColorWheel attribute), [728, 896](#)  
 color (kivy.uix.image.Image attribute), [749, 924](#)  
 color (kivy.uix.label.Label attribute), [754, 930](#)  
 color\_selected (kivy.uix.treeview.TreeViewNode attribute), [829, 1041](#)  
 colorfmt (kivy.graphics.texture.Texture attribute), [459, 513](#)  
 ColorPicker (class in kivy.uix.colorpicker), [728, 895](#)  
 colors (kivy.uix.rst.RstDocument attribute), [852, 966](#)  
 ColorWheel (class in kivy.uix.colorpicker), [728, 896](#)  
 cols (kivy.uix.gridlayout.GridLayout attribute), [747, 919](#)  
 cols\_minimum (kivy.uix.gridlayout.GridLayout attribute), [747, 919](#)  
 complete\_stroke() (kivy.uix.gesturesurface.GestureContainer method), [744, 915](#)  
 CompositeListItem (class in kivy.uix.listview), [768, 946](#)  
 CompoundSelectionBehavior (class in kivy.uix.behaviors), [713, 873](#)  
 Config (in module kivy.config), [195, 318](#)  
 config (kivy.app.App attribute), [177, 291](#)  
 config (kivy.uix.settings.SettingsPanel attribute), [796, 994](#)  
 ConfigParser (class in kivy.config), [195, 318](#)  
 ConfigParserProperty (class in kivy.properties), [241, 679](#)  
 container (kivy.uix.accordion.AccordionItem attribute), [702, 860](#)  
 container (kivy.uix.dropdown.DropDown attribute), [731, 899](#)  
 container (kivy.uix.listview.ListView attribute), [769, 947](#)  
 container (kivy.uix.settings.ContentPanel attribute), [800, 998](#)  
 container\_title (kivy.uix.accordion.AccordionItem attribute), [702, 860](#)  
 content (kivy.uix.bubble.Bubble attribute), [720, 881](#)  
 content (kivy.uix.popup.Popup attribute), [773, 956](#)  
 content (kivy.uix.settings.InterfaceWithSidebar attribute), [799, 997](#)  
 content (kivy.uix.settings.SettingItem attribute), [797, 994](#)  
 content (kivy.uix.tabbedpanel.TabbedPanel attribute), [813, 1018](#)  
 content (kivy.uix.tabbedpanel.TabbedPanelHeader attribute), [814, 1020](#)  
 content\_height (kivy.core.text.LabelBase attribute), [331, 360](#)  
 content\_size (kivy.core.text.LabelBase attribute), [331, 360](#)  
 content\_size (kivy.uix.accordion.AccordionItem attribute), [703, 860](#)  
 content\_width (kivy.core.text.LabelBase attribute), [331, 360](#)  
 ContentPanel (class in kivy.uix.settings), [799, 997](#)  
 Context (class in kivy.graphics.context), [437, 475](#)  
 ContextInstruction (class in kivy.graphics), [416](#)  
 ContextInstruction (class in



kivy.graphics.instructions), 430, 488  
 ContextualActionView (class in kivy.uix.actionbar), 706, 865  
 convert\_distance\_to\_scroll() (kivy.uix.scrollview.ScrollView method), 791, 987  
 copy() (kivy.uix.textinput.TextInput method), 819, 1026  
 copy\_to() (kivy.input.MotionEvent method), 519  
 copy\_to() (kivy.input.motionevent.MotionEvent method), 532, 540  
 count() (kivy.storage.AbstractStore method), 687  
 counter (kivy.input.recorder.Recorder attribute), 530, 594  
 create() (kivy.atlas.Atlas static method), 185, 299  
 create() (kivy.graphics.texture.Texture method), 459, 513  
 create() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 528, 568, 586  
 create\_from\_data() (kivy.graphics.texture.Texture method), 460, 513  
 create\_inspector() (in module kivy.modules.inspector), 633, 638  
 create\_json\_panel() (kivy.uix.settings.Settings method), 796, 994  
 create\_property() (kivy.event.EventDispatcher method), 199, 396  
 create\_settings() (kivy.app.App method), 177, 291  
 create\_trigger() (kivy.clock.ClockBase method), 190, 310  
 create\_view() (kivy.adapters.listadapter.ListAdapter method), 254, 266  
 create\_window() (kivy.core.window.WindowBase method), 339, 377  
 current (kivy.uix.screenmanager.ScreenManager attribute), 785, 980  
 current\_panel (kivy.uix.settings.ContentPanel attribute), 800, 998  
 current\_screen (kivy.uix.screenmanager.ScreenManager attribute), 785, 980  
 current\_slide (kivy.uix.carousel.Carousel attribute), 724, 887  
 current\_tab (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
 current\_uid (kivy.uix.settings.ContentPanel attribute), 800, 998  
 cursor (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cursor\_blink (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cursor\_col (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cursor\_color (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cursor\_index() (kivy.uix.textinput.TextInput method), 819, 1027  
 cursor\_offset() (kivy.uix.textinput.TextInput method), 819, 1027  
 cursor\_pos (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cursor\_row (kivy.uix.textinput.TextInput attribute), 819, 1027  
 cut() (kivy.uix.textinput.TextInput method), 819, 1027  
 cut\_to\_sel() (kivy.adapters.dictadapter.DictAdapter method), 252, 263  
 cut\_to\_sel() (kivy.adapters.listadapter.ListAdapter method), 254, 266

## D

DampedScrollEffect (class in kivy.effects.dampedscroll), 383, 387  
 dash\_length (kivy.graphics.Bezier attribute), 412  
 dash\_length (kivy.graphics.Line attribute), 421  
 dash\_length (kivy.graphics.vertex\_instructions.Bezier attribute), 470  
 dash\_length (kivy.graphics.vertex\_instructions.Line attribute), 468  
 dash\_offset (kivy.graphics.Bezier attribute), 412  
 dash\_offset (kivy.graphics.Line attribute), 421  
 dash\_offset (kivy.graphics.vertex\_instructions.Bezier attribute), 470  
 dash\_offset (kivy.graphics.vertex\_instructions.Line attribute), 468  
 data (kivy.adapters.adapter.Adapter attribute), 252, 259  
 data (kivy.adapters.dictadapter.DictAdapter attribute), 252, 263  
 data (kivy.adapters.listadapter.ListAdapter attribute), 254, 266  
 data (kivy.adapters.simplelistadapter.SimpleListAdapter attribute), 257, 271  
 data (kivy.core.image.ImageData attribute), 328, 355  
 decode\_result() (kivy.network.urlrequest.UrlRequest method), 662, 666  
 default\_tab (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
 default\_tab\_cls (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
 default\_tab\_content (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
 default\_tab\_text (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
 delete() (kivy.storage.AbstractStore method), 688  
 delete\_selection() (kivy.uix.textinput.TextInput method), 819, 1027  
 density() (kivy.metrics.MetricsBase method), 223, 628  
 depack() (kivy.input.MotionEvent method), 519

`depack()` (kivy.input.motionevent.MotionEvent method), 532, 540  
`deprecated()` (in module kivy.utils), 244, 1067  
`desc` (kivy.uix.settings.SettingItem attribute), 797, 995  
`deselect()` (kivy.uix.listview.SelectableView method), 767, 945  
`deselect_node()` (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 769, 947  
`deselected_color` (kivy.uix.listview.CompositeListItem attribute), 768, 946  
`deselected_color` (kivy.uix.listview.ListItemButton attribute), 768, 946  
`destroy_settings()` (kivy.app.App method), 177, 291  
`device` (kivy.input.motionevent.MotionEvent attribute), 532, 540  
`DictAdapter` (class in kivy.adapters.dictadapter), 252, 263  
`DictProperty` (class in kivy.properties), 240, 678  
`DictStore` (class in kivy.storage.dictstore), 688, 691  
`difference()` (in module kivy.utils), 244, 1067  
`direction` (kivy.uix.carousel.Carousel attribute), 724, 888  
`direction` (kivy.uix.screenmanager.SlideTransition attribute), 788, 983  
`directory` (kivy.app.App attribute), 177, 292  
`dirselect` (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
`disabled` (kivy.uix.settings.SettingItem attribute), 797, 995  
`disabled` (kivy.uix.widget.Widget attribute), 845, 1060  
`disabled_color` (kivy.uix.label.Label attribute), 754, 930  
`disabled_foreground_color` (kivy.uix.textinput.TextInput attribute), 819, 1027  
`dismiss()` (kivy.uix.dropdown.DropDown method), 731, 899  
`dismiss()` (kivy.uix.modalview.ModalView method), 771, 950  
`dismiss_on_select` (kivy.uix.dropdown.DropDown attribute), 731, 899  
`dispatch()` (kivy.event.EventDispatcher method), 199, 397  
`dispatch()` (kivy.properties.Property method), 236, 674  
`dispatch_input()` (kivy.base.EventLoopBase method), 208, 301  
`displacement` (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
`display_settings()` (kivy.app.App method), 177, 292  
`distance()` (kivy.input.MotionEvent method), 519  
`distance()` (kivy.input.motionevent.MotionEvent method), 532, 540  
`distance()` (kivy.vector.Vector method), 247, 1072  
`distance2()` (kivy.vector.Vector method), 247, 1072  
`divider` (kivy.uix.listview.ListView attribute), 769, 947  
`divider_height` (kivy.uix.listview.ListView attribute), 769, 947  
`do_backspace()` (kivy.uix.textinput.TextInput method), 819, 1027  
`do_collide_after_children` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_cursor_movement()` (kivy.uix.textinput.TextInput method), 820, 1027  
`do_default_tab` (kivy.uix.tabbedpanel.TabbedPanel attribute), 813, 1019  
`do_layout()` (kivy.uix.layout.Layout method), 759, 935  
`do_redo()` (kivy.uix.textinput.TextInput method), 820, 1028  
`do_rotation` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_scale` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_scroll` (kivy.uix.scrollview.ScrollView attribute), 791, 987  
`do_scroll_x` (kivy.uix.scrollview.ScrollView attribute), 791, 987  
`do_scroll_y` (kivy.uix.scrollview.ScrollView attribute), 791, 987  
`do_translation` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_translation_x` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_translation_y` (kivy.uix.scatter.Scatter attribute), 780, 973  
`do_undo()` (kivy.uix.textinput.TextInput method), 820, 1028  
`docked` (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053  
`document_root` (kivy.uix.rst.RstDocument attribute), 852, 966  
`dot()` (kivy.vector.Vector method), 247, 1072  
`dot_product()` (kivy.gesture.Gesture method), 204, 409  
`double_tap_time` (kivy.input.motionevent.MotionEvent attribute), 532, 540  
`dp()` (in module kivy.metrics), 224, 629  
`dpi()` (kivy.core.window.WindowBase method), 340, 377  
`dpi()` (kivy.metrics.MetricsBase method), 223, 628  
`dpi_rounded()` (kivy.metrics.MetricsBase method), 223, 628  
`dpos` (kivy.input.MotionEvent attribute), 519

dpos (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 drag\_distance (kivy.uix.behaviors.DragBehavior attribute), 710, 870  
 drag\_rect\_height (kivy.uix.behaviors.DragBehavior attribute), 710, 870  
 drag\_rect\_width (kivy.uix.behaviors.DragBehavior attribute), 710, 870  
 drag\_rect\_x (kivy.uix.behaviors.DragBehavior attribute), 710, 870  
 drag\_rect\_y (kivy.uix.behaviors.DragBehavior attribute), 710, 871  
 drag\_rectangle (kivy.uix.behaviors.DragBehavior attribute), 711, 871  
 drag\_threshold (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
 drag\_timeout (kivy.uix.behaviors.DragBehavior attribute), 711, 871  
 DragBehavior (class in kivy.uix.behaviors), 710, 870  
 draw() (kivy.graphics.Canvas method), 414  
 draw() (kivy.graphics.instructions.Canvas method), 432, 489  
 DropDown (class in kivy.uix.dropdown), 730, 898  
 dropdown\_cls (kivy.uix.spinner.Spinner attribute), 803, 1004  
 dsx (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 dsy (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 dsz (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 duration (kivy.animation.Animation attribute), 162, 275  
 duration (kivy.core.video.VideoBase attribute), 336, 373  
 duration (kivy.uix.screenmanager.FallOutTransition attribute), 788, 983  
 duration (kivy.uix.screenmanager.RiseInTransition attribute), 788, 983  
 duration (kivy.uix.screenmanager.TransitionBase attribute), 786, 981  
 duration (kivy.uix.video.Video attribute), 835, 1043  
 duration (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 duration (kivy.uix.videoplayer.VideoPlayerAnnotation attribute), 841, 1049  
 dx (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 dy (kivy.input.motionevent.MotionEvent attribute), 532, 540  
 dz (kivy.input.motionevent.MotionEvent attribute), 533, 541

## E

edge\_damping (kivy.effects.dampedscroll.DampedScrollEffect attribute), 383, 387  
 effect\_cls (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 effect\_x (kivy.uix.scrollview.ScrollView attribute), 791, 987  
 effect\_y (kivy.uix.scrollview.ScrollView attribute), 791, 988  
 EffectBase (class in kivy.uix.effectwidget), 733, 903  
 effects (kivy.uix.effectwidget.EffectWidget attribute), 733, 902  
 EffectWidget (class in kivy.uix.effectwidget), 733, 902  
 Ellipse (class in kivy.graphics), 416  
 Ellipse (class in kivy.graphics.vertex\_instructions), 465  
 ellipse (kivy.graphics.Line attribute), 421  
 ellipse (kivy.graphics.vertex\_instructions.Line attribute), 468  
 ensure\_window() (kivy.base.EventLoopBase method), 208, 301  
 entry\_released() (kivy.uix.filechooser.FileChooserController method), 738, 908  
 entry\_touched() (kivy.uix.filechooser.FileChooserController method), 738, 908  
 eos (kivy.uix.video.Video attribute), 835, 1043  
 error (kivy.network.urlrequest.UrlRequest attribute), 663, 666  
 error\_image (kivy.loader.LoaderBase attribute), 182, 621  
 escape\_markup() (in module kivy.utils), 245, 1068  
 even\_color (kivy.uix.treeview.TreeViewNode attribute), 829, 1041  
 EventDispatcher (class in kivy.event), 197, 395  
 EventLoop (in module kivy.base), 207, 301  
 EventLoopBase (class in kivy.base), 208, 301  
 events() (kivy.event.EventDispatcher method), 199, 397  
 ExceptionHandler (class in kivy.base), 209, 302  
 ExceptionManager (in module kivy.base), 209, 302  
 ExceptionManagerBase (class in kivy.base), 209, 302  
 exists() (kivy.storage.AbstractStore method), 688  
 exit() (kivy.base.EventLoopBase method), 208, 301  
 export\_gesture() (kivy.multistroke.Recognizer method), 226, 653  
 export\_to\_png() (kivy.uix.widget.Widget method), 845, 1061

## F

Factory (in module kivy.factory), 202, 403  
 FadeTransition (class in kivy.uix.screenmanager), 788, 983

FallOutTransition (class in kivy.uix.screenmanager), 788, 983  
 Fbo (class in kivy.graphics), 416  
 Fbo (class in kivy.graphics.fbo), 439, 482  
 fbo (kivy.uix.effectwidget.EffectBase attribute), 733, 903  
 fbo\_list (kivy.uix.effectwidget.EffectWidget attribute), 733, 902  
 file\_encodings (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
 file\_system (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
 FileChooserController (class in kivy.uix.filechooser), 737, 907  
 FileChooserIconView (class in kivy.uix.filechooser), 737, 907  
 FileChooserListView (class in kivy.uix.filechooser), 737, 907  
 FileChooserProgressBase (class in kivy.uix.filechooser), 740, 910  
 filename (kivy.atlas.Atlas attribute), 186, 299  
 filename (kivy.core.audio.Sound attribute), 324, 345  
 filename (kivy.core.image.Image attribute), 327, 354  
 filename (kivy.core.video.VideoBase attribute), 336, 373  
 filename (kivy.input.recorder.Recorder attribute), 530, 594  
 files (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
 FileSystemAbstract (class in kivy.uix.filechooser), 740, 910  
 FileSystemLocal (class in kivy.uix.filechooser), 740, 910  
 filter() (kivy.multistroke.Recognizer method), 226, 653  
 filter\_dirs (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
 filters (kivy.uix.filechooser.FileChooserController attribute), 738, 908  
 find() (kivy.gesture.GestureDatabase method), 204, 410  
 find() (kivy.storage.AbstractStore method), 688  
 find\_colliding\_gesture() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 find\_double\_tap() (kivy.input.postproc.doubletap.InputPostprocDoubleTap method), 522, 546, 553  
 find\_triple\_tap() (kivy.input.postproc.tripletap.InputPostprocTripleTap method), 523, 547, 559  
 flip() (kivy.core.window.WindowBase method), 340, 378  
 flip\_horizontal() (kivy.graphics.texture.Texture method), 460, 513  
 flip\_vertical (kivy.core.image.ImageData attribute), 328, 355  
 flip\_vertical() (kivy.graphics.texture.Texture method), 460, 513  
 FloatLayout (class in kivy.uix.floatlayout), 741, 912  
 fmt (kivy.core.image.ImageData attribute), 328, 355  
 focus (kivy.uix.textinput.TextInput attribute), 820, 1028  
 focus\_next (kivy.uix.behaviors.FocusBehavior attribute), 711, 871  
 focus\_previous (kivy.uix.behaviors.FocusBehavior attribute), 712, 872  
 FocusBehavior (class in kivy.uix.behaviors), 711, 871  
 focused (kivy.uix.behaviors.FocusBehavior attribute), 712, 872  
 font\_name (kivy.uix.colorpicker.ColorPicker attribute), 728, 895  
 font\_name (kivy.uix.label.Label attribute), 754, 930  
 font\_name (kivy.uix.textinput.TextInput attribute), 820, 1028  
 font\_size (kivy.uix.label.Label attribute), 754, 930  
 font\_size (kivy.uix.textinput.TextInput attribute), 820, 1028  
 fontid (kivy.core.text.LabelBase attribute), 331, 360  
 fontsize() (kivy.metrics.MetricsBase method), 223, 628  
 foreground\_color (kivy.uix.textinput.TextInput attribute), 821, 1028  
 frames (kivy.clock.ClockBase attribute), 191, 310  
 frames\_displayed (kivy.clock.ClockBase attribute), 191, 310  
 frametime (kivy.clock.ClockBase attribute), 191, 310  
 friction (kivy.effects.kinetic.KineticEffect attribute), 384, 389  
 fs (kivy.graphics.shader.Shader attribute), 453, 506  
 fs (kivy.uix.screenmanager.ShaderTransition attribute), 788, 982  
 fullscreen (kivy.core.window.WindowBase attribute), 340, 378  
 fullscreen (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 func\_op (kivy.graphics.stencil\_instructions.StencilUse attribute), 455, 508  
 func\_op (kivy.graphics.StencilUse attribute), 426  
 FXAAEffect (class in kivy.uix.effectwidget), 735, 904

## G

g (kivy.graphics.ClearColor attribute), 428  
 g (kivy.graphics.Color attribute), 415  
 g (kivy.graphics.context\_instructions.Color attribute), 435, 478



g (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485  
 g (kivy.uix.colorpicker.ColorWheel attribute), 728, 896  
 garden\_system\_dir (in module kivy.garden), 202, 406  
 Gesture (class in kivy.gesture), 204, 409  
 gesture\_to\_str() (kivy.gesture.GestureDatabase method), 204, 410  
 GestureContainer (class in kivy.uix.gesturesurface), 743, 914  
 GestureDatabase (class in kivy.gesture), 204, 409  
 GestureStroke (class in kivy.gesture), 204, 410  
 GestureSurface (class in kivy.uix.gesturesurface), 742, 913  
 get() (kivy.cache.Cache static method), 186, 305  
 get() (kivy.input.factory.MotionEventFactory static method), 535, 537  
 get() (kivy.input.MotionEventFactory static method), 521  
 get() (kivy.properties.Property method), 236, 674  
 get() (kivy.storage.AbstractStore method), 688  
 get\_angle\_similarity() (kivy.multistroke.Candidate method), 231, 658  
 get\_application\_config() (kivy.app.App method), 177, 292  
 get\_application\_icon() (kivy.app.App method), 178, 292  
 get\_application\_name() (kivy.app.App method), 178, 292  
 get\_boottime() (kivy.clock.ClockBase method), 191, 310  
 get\_cached\_extents() (kivy.core.text.LabelBase method), 331, 360  
 get\_color\_from\_hex() (in module kivy.utils), 244, 1067  
 get\_configparser() (kivy.config.ConfigParser static method), 196, 318  
 get\_connection\_for\_scheme() (kivy.network.urlrequest.UrlRequest method), 663, 666  
 get\_current\_context() (in module kivy.context), 197, 321  
 get\_cursor\_from\_index() (kivy.uix.textinput.TextInput method), 821, 1029  
 get\_cursor\_from\_xy() (kivy.uix.textinput.TextInput method), 821, 1029  
 get\_distance() (kivy.multistroke.MultistrokeGesture method), 230, 657  
 get\_extents() (kivy.core.text.LabelBase method), 332, 360  
 get\_format() (kivy.properties.NumericProperty method), 237, 675  
 get\_fps() (kivy.clock.ClockBase method), 191, 310  
 get\_gesture() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 get\_group() (kivy.graphics.InstructionGroup method), 418  
 get\_group() (kivy.graphics.instructions.InstructionGroup method), 430, 487  
 get\_hex\_from\_color() (in module kivy.utils), 244, 1067  
 get\_lastaccess() (kivy.cache.Cache static method), 187, 305  
 get\_max() (kivy.properties.BoundedNumericProperty method), 238, 677  
 get\_min() (kivy.properties.BoundedNumericProperty method), 238, 677  
 get\_mipmap() (kivy.core.image.ImageData method), 328, 355  
 get\_nice\_size() (kivy.uix.filechooser.FileChooserController method), 739, 909  
 get\_node\_at\_pos() (kivy.uix.treeview.TreeView method), 827, 1039  
 get\_parent\_window() (kivy.uix.widget.Widget method), 845, 1061  
 get\_pixel\_color() (kivy.graphics.Fbo method), 417  
 get\_pixel\_color() (kivy.graphics.fbo.Fbo method), 440, 483  
 get\_pos() (kivy.core.audio.Sound method), 324, 345  
 get\_property\_observers() (kivy.event.EventDispatcher method), 200, 397  
 get\_protractor\_vector() (kivy.multistroke.Candidate method), 231, 659  
 get\_random\_color() (in module kivy.utils), 244, 1067  
 get\_region() (kivy.graphics.texture.Texture method), 460, 513  
 get\_rfps() (kivy.clock.ClockBase method), 191, 310  
 get\_rigid\_rotation() (kivy.gesture.Gesture method), 204, 409  
 get\_root\_window() (kivy.uix.widget.Widget method), 845, 1061  
 get\_running\_app() (kivy.app.App static method), 178, 292  
 get\_score() (kivy.gesture.Gesture method), 204, 409  
 get\_screen() (kivy.uix.screenmanager.ScreenManager method), 785, 980  
 get\_selectable\_nodes() (kivy.uix.behaviors.CompoundSelectionBehavior method), 714, 874  
 get\_start\_unit\_vector() (kivy.multistroke.Candidate method),

[231, 659](#)  
 get\_time() (kivy.clock.ClockBase method), [191, 310](#)  
 get\_timestamp() (kivy.cache.Cache static method),  
[187, 306](#)  
 get\_value() (kivy.uix.settings.SettingsPanel  
 method), [796, 994](#)  
 get\_vectors() (kivy.uix.gesturesurface.GestureContainer  
 method), [744, 915](#)  
 get\_widgets() (kivy.uix.behaviors.ToggleButtonBehavior  
 static method), [709, 870](#)  
 getdefault() (kivy.config.ConfigParser method),  
[196, 318](#)  
 getdefaultint() (kivy.config.ConfigParser method),  
[196, 318](#)  
 getsize() (kivy.uix.filechooser.FileSystemAbstract  
 method), [740, 910](#)  
 getter() (kivy.event.EventDispatcher method), [200,](#)  
[397](#)  
 gl\_get\_extensions() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 gl\_get\_texture\_formats() (in module  
 kivy.graphics.opengl\_utils), [452, 504](#)  
 gl\_get\_version() (in module  
 kivy.graphics.opengl\_utils), [452, 504](#)  
 gl\_get\_version\_major() (in module  
 kivy.graphics.opengl\_utils), [452, 504](#)  
 gl\_get\_version\_minor() (in module  
 kivy.graphics.opengl\_utils), [452, 504](#)  
 gl\_has\_capability() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 gl\_has\_extension() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 gl\_has\_texture\_conversion() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 gl\_has\_texture\_format() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 gl\_has\_texture\_native\_format() (in module  
 kivy.graphics.opengl\_utils), [451, 504](#)  
 gl\_register\_get\_size() (in module  
 kivy.graphics.opengl\_utils), [451, 503](#)  
 glActiveTexture() (in module  
 kivy.graphics.opengl), [442, 493](#)  
 glAttachShader() (in module  
 kivy.graphics.opengl), [442, 493](#)  
 glBindAttribLocation() (in module  
 kivy.graphics.opengl), [442, 493](#)  
 glBindBuffer() (in module kivy.graphics.opengl),  
[443, 493](#)  
 glBindFramebuffer() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glBindRenderbuffer() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glBindTexture() (in module kivy.graphics.opengl),  
[443, 493](#)  
 glBlendColor() (in module kivy.graphics.opengl),  
[443, 493](#)  
 glBlendEquation() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glBlendEquationSeparate() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glBlendFunc() (in module kivy.graphics.opengl),  
[443, 493](#)  
 glBlendFuncSeparate() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glBufferData() (in module kivy.graphics.opengl),  
[443, 493](#)  
 glBufferSubData() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glCheckFramebufferStatus() (in module  
 kivy.graphics.opengl), [443, 493](#)  
 glClear() (in module kivy.graphics.opengl), [443,](#)  
[494](#)  
 glClearColor() (in module kivy.graphics.opengl),  
[443, 494](#)  
 glClearStencil() (in module kivy.graphics.opengl),  
[443, 494](#)  
 glColorMask() (in module kivy.graphics.opengl),  
[443, 494](#)  
 glCompileShader() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCompressedTexImage2D() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCompressedTexSubImage2D() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCopyTexImage2D() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCopyTexSubImage2D() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCreateProgram() (in module  
 kivy.graphics.opengl), [443, 494](#)  
 glCreateShader() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glCullFace() (in module kivy.graphics.opengl),  
[444, 494](#)  
 glDeleteBuffers() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDeleteFramebuffers() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDeleteProgram() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDeleteRenderbuffers() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDeleteShader() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDeleteTextures() (in module  
 kivy.graphics.opengl), [444, 494](#)  
 glDepthFunc() (in module kivy.graphics.opengl),  
[444, 494](#)  
 glDepthMask() (in module kivy.graphics.opengl),  
[444, 494](#)

glDetachShader() (in module kivy.graphics.opengl), 444, 494  
 glDisable() (in module kivy.graphics.opengl), 444, 494  
 glDisableVertexAttribArray() (in module kivy.graphics.opengl), 444, 495  
 glDrawArrays() (in module kivy.graphics.opengl), 444, 495  
 glDrawElements() (in module kivy.graphics.opengl), 444, 495  
 glEnable() (in module kivy.graphics.opengl), 444, 495  
 glEnableVertexAttribArray() (in module kivy.graphics.opengl), 444, 495  
 glFinish() (in module kivy.graphics.opengl), 444, 495  
 glFlush() (in module kivy.graphics.opengl), 444, 495  
 glFramebufferRenderbuffer() (in module kivy.graphics.opengl), 444, 495  
 glFramebufferTexture2D() (in module kivy.graphics.opengl), 444, 495  
 glFrontFace() (in module kivy.graphics.opengl), 444, 495  
 glGenBuffers() (in module kivy.graphics.opengl), 445, 495  
 glGenMipmap() (in module kivy.graphics.opengl), 445, 495  
 glGenFramebuffers() (in module kivy.graphics.opengl), 445, 495  
 glGenRenderbuffers() (in module kivy.graphics.opengl), 445, 495  
 glGenTextures() (in module kivy.graphics.opengl), 445, 495  
 glGetActiveAttrib() (in module kivy.graphics.opengl), 445, 495  
 glGetActiveUniform() (in module kivy.graphics.opengl), 445, 495  
 glGetAttachedShaders() (in module kivy.graphics.opengl), 445, 495  
 glGetAttribLocation() (in module kivy.graphics.opengl), 445, 496  
 glGetBooleanv() (in module kivy.graphics.opengl), 445, 496  
 glGetBufferParameteriv() (in module kivy.graphics.opengl), 445, 496  
 glGetError() (in module kivy.graphics.opengl), 445, 496  
 glGetFloatv() (in module kivy.graphics.opengl), 445, 496  
 glGetFramebufferAttachmentParameteriv() (in module kivy.graphics.opengl), 445, 496  
 glGetIntegerv() (in module kivy.graphics.opengl), 446, 496  
 glGetProgramInfoLog() (in module kivy.graphics.opengl), 446, 496  
 glGetProgramiv() (in module kivy.graphics.opengl), 446, 496  
 glGetRenderbufferParameteriv() (in module kivy.graphics.opengl), 446, 496  
 glGetShaderInfoLog() (in module kivy.graphics.opengl), 446, 496  
 glGetShaderiv() (in module kivy.graphics.opengl), 446, 497  
 glGetShaderPrecisionFormat() (in module kivy.graphics.opengl), 446, 496  
 glGetShaderSource() (in module kivy.graphics.opengl), 446, 496  
 getString() (in module kivy.graphics.opengl), 446, 497  
 glGetTexParameterfv() (in module kivy.graphics.opengl), 446, 497  
 glGetTexParameteriv() (in module kivy.graphics.opengl), 446, 497  
 glGetUniformfv() (in module kivy.graphics.opengl), 446, 497  
 glGetUniformiv() (in module kivy.graphics.opengl), 446, 497  
 glGetUniformLocation() (in module kivy.graphics.opengl), 446, 497  
 glGetVertexAttribfv() (in module kivy.graphics.opengl), 447, 497  
 glGetVertexAttribiv() (in module kivy.graphics.opengl), 447, 497  
 glGetVertexAttribPointerv() (in module kivy.graphics.opengl), 446, 497  
 glHint() (in module kivy.graphics.opengl), 447, 497  
 glIsBuffer() (in module kivy.graphics.opengl), 447, 497  
 glIsEnabled() (in module kivy.graphics.opengl), 447, 497  
 glIsFramebuffer() (in module kivy.graphics.opengl), 447, 497  
 glIsProgram() (in module kivy.graphics.opengl), 447, 497  
 glIsRenderbuffer() (in module kivy.graphics.opengl), 447, 497  
 glIsShader() (in module kivy.graphics.opengl), 447, 497  
 glIsTexture() (in module kivy.graphics.opengl), 447, 497  
 glLineWidth() (in module kivy.graphics.opengl), 447, 497  
 glLinkProgram() (in module kivy.graphics.opengl), 447, 497  
 glPixelStorei() (in module kivy.graphics.opengl), 447, 498  
 glPolygonOffset() (in module kivy.graphics.opengl), 447, 498

[glReadPixels\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glReleaseShaderCompiler\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glRenderbufferStorage\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glSampleCoverage\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glScissor\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glShaderBinary\(\)](#) (in module `kivy.graphics.opengl`), [447, 498](#)  
[glShaderSource\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glsl](#) (`kivy.uix.effectwidget.EffectBase` attribute), [733, 903](#)  
[glStencilFunc\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glStencilFuncSeparate\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glStencilMask\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glStencilMaskSeparate\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glStencilOp\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glStencilOpSeparate\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glTexImage2D\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glTexParameterf\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glTexParameterfv\(\)](#) (in module `kivy.graphics.opengl`), [448, 498](#)  
[glTexParameterI\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glTexParameterIv\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glTexSubImage2D\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glUniform1f\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glUniform1fv\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glUniform1i\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glUniform1iv\(\)](#) (in module `kivy.graphics.opengl`), [448, 499](#)  
[glUniform2f\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform2fv\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform2i\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform2iv\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform3f\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform3fv\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform3i\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform3iv\(\)](#) (in module `kivy.graphics.opengl`), [449, 499](#)  
[glUniform4f\(\)](#) (in module `kivy.graphics.opengl`), [449, 500](#)  
[glUniform4fv\(\)](#) (in module `kivy.graphics.opengl`), [449, 500](#)  
[glUniform4i\(\)](#) (in module `kivy.graphics.opengl`), [449, 500](#)  
[glUniform4iv\(\)](#) (in module `kivy.graphics.opengl`), [449, 500](#)  
[glUniformMatrix2fv\(\)](#) (in module `kivy.graphics.opengl`), [449, 500](#)  
[glUniformMatrix3fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glUniformMatrix4fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glUseProgram\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glValidateProgram\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glVertexAttrib1f\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glVertexAttrib1fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glVertexAttrib2f\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glVertexAttrib2fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 500](#)  
[glVertexAttrib3f\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[glVertexAttrib3fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[glVertexAttrib4f\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[glVertexAttrib4fv\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[glVertexAttribPointer\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[glViewport\(\)](#) (in module `kivy.graphics.opengl`), [450, 501](#)  
[goto\(\)](#) (`kivy.uix.rst.RstDocument` method), [852, 966](#)  
[goto\\_node\(\)](#) (`kivy.uix.behaviors.CompoundSelectionBehavior` method), [714, 874](#)  
[grab\(\)](#) (`kivy.input.MotionEvent` method), [519](#)  
[grab\(\)](#) (`kivy.input.motionevent.MotionEvent` method), [533, 541](#)  
[grab\\_current](#) (`kivy.input.motionevent.MotionEvent`



attribute), 533, 541  
 GraphicException (class in kivy.graphics), 418  
 GraphicException (class in kivy.graphics.vertex\_instructions), 470  
 GridLayout (class in kivy.uix.gridlayout), 747, 919  
 GridLayoutException (class in kivy.uix.gridlayout), 748, 920  
 group (kivy.uix.behaviors.ToggleButtonBehavior attribute), 710, 870

## H

h (kivy.graphics.Color attribute), 415  
 h (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 valign (kivy.uix.label.Label attribute), 754, 930  
 handle\_exception() (kivy.base.ExceptionHandler method), 209, 302  
 handle\_exception() (kivy.base.ExceptionManagerBase method), 209, 302  
 handle\_image\_left (kivy.uix.textinput.TextInput attribute), 821, 1029  
 handle\_image\_middle (kivy.uix.textinput.TextInput attribute), 821, 1029  
 handle\_image\_right (kivy.uix.textinput.TextInput attribute), 821, 1029  
 handles() (kivy.uix.gesturesurface.GestureContainer method), 744, 915  
 has\_after (kivy.graphics.Canvas attribute), 414  
 has\_after (kivy.graphics.instructions.Canvas attribute), 432, 489  
 has\_before (kivy.graphics.Canvas attribute), 414  
 has\_before (kivy.graphics.instructions.Canvas attribute), 432, 489  
 has\_screen() (kivy.uix.screenmanager.ScreenManager method), 785, 980  
 have\_properties\_to\_animate() (kivy.animation.Animation method), 162, 275  
 hbar (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 height (kivy.core.image.Image attribute), 327, 354  
 height (kivy.core.image.ImageData attribute), 329, 355  
 height (kivy.core.window.WindowBase attribute), 340, 378  
 height (kivy.graphics.texture.Texture attribute), 460, 513  
 height (kivy.input.shape.ShapeRect attribute), 536, 597  
 height (kivy.uix.widget.Widget attribute), 845, 1061  
 hex\_color (kivy.uix.colorpicker.ColorPicker attribute), 728, 895  
 hide\_keyboard() (kivy.uix.textinput.TextInput method), 821, 1029  
 hide\_root (kivy.uix.treeview.TreeView attribute), 827, 1039  
 hint\_text (kivy.uix.textinput.TextInput attribute), 821, 1029  
 hint\_text\_color (kivy.uix.textinput.TextInput attribute), 821, 1029  
 HorizontalBlurEffect (class in kivy.uix.effectwidget), 734, 904  
 hsv (kivy.graphics.Color attribute), 416  
 hsv (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 hsv (kivy.uix.colorpicker.ColorPicker attribute), 728, 895

## I

icon (kivy.app.App attribute), 178, 292  
 icon (kivy.uix.actionbar.ActionButton attribute), 705, 864  
 icon (kivy.uix.actionbar.ActionToggleButton attribute), 705, 864  
 id (kivy.graphics.texture.Texture attribute), 460, 513  
 id (kivy.input.motionevent.MotionEvent attribute), 533, 541  
 id (kivy.uix.widget.Widget attribute), 846, 1061  
 identity() (kivy.graphics.transformation.Matrix method), 462, 515  
 idle() (kivy.base.EventLoopBase method), 208, 301  
 ids (kivy.uix.widget.Widget attribute), 846, 1061  
 Image (class in kivy.core.image), 326, 353  
 Image (class in kivy.uix.image), 749, 923  
 image (kivy.core.image.Image attribute), 327, 354  
 image() (kivy.loader.LoaderBase method), 182, 621  
 image\_loading (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 image\_overlay\_play (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 image\_pause (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 image\_play (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1047  
 image\_ratio (kivy.uix.image.Image attribute), 749, 924  
 image\_stop (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1048  
 image\_volumehigh (kivy.uix.videoplayer.VideoPlayer attribute), 839, 1048  
 image\_volumelow (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048  
 image\_volumemedium (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048  
 image\_volumemuted (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048

ImageData (class in kivy.core.image), 328, 355  
 import\_gesture() (kivy.multistroke.Recognizer method), 227, 654  
 important (kivy.uix.actionbar.ActionItem attribute), 704, 863  
 in\_back() (kivy.animation.AnimationTransition static method), 162, 275  
 in\_bbox() (kivy.vector.Vector static method), 247, 1073  
 in\_bounce() (kivy.animation.AnimationTransition static method), 163, 276  
 in\_circ() (kivy.animation.AnimationTransition static method), 163, 276  
 in\_cubic() (kivy.animation.AnimationTransition static method), 163, 276  
 in\_elastic() (kivy.animation.AnimationTransition static method), 163, 276  
 in\_expo() (kivy.animation.AnimationTransition static method), 164, 277  
 in\_out\_back() (kivy.animation.AnimationTransition static method), 164, 277  
 in\_out\_bounce() (kivy.animation.AnimationTransition static method), 164, 277  
 in\_out\_circ() (kivy.animation.AnimationTransition static method), 164, 277  
 in\_out\_cubic() (kivy.animation.AnimationTransition static method), 165, 278  
 in\_out\_elastic() (kivy.animation.AnimationTransition static method), 165, 278  
 in\_out\_expo() (kivy.animation.AnimationTransition static method), 165, 278  
 in\_out\_quad() (kivy.animation.AnimationTransition static method), 165, 278  
 in\_out\_quart() (kivy.animation.AnimationTransition static method), 166, 279  
 in\_out\_quint() (kivy.animation.AnimationTransition static method), 166, 279  
 in\_out\_sine() (kivy.animation.AnimationTransition static method), 166, 279  
 in\_quad() (kivy.animation.AnimationTransition static method), 166, 279  
 in\_quart() (kivy.animation.AnimationTransition static method), 167, 280  
 in\_quint() (kivy.animation.AnimationTransition static method), 167, 280  
 in\_sine() (kivy.animation.AnimationTransition static method), 167, 280  
 inch() (in module kivy.metrics), 224, 629  
 indent\_level (kivy.uix.treeview.TreeView attribute), 827, 1039  
 indent\_start (kivy.uix.treeview.TreeView attribute), 827, 1039  
 index (kivy.core.camera.CameraBase attribute), 325, 347  
 index (kivy.uix.camera.Camera attribute), 722, 885  
 index (kivy.uix.carousel.Carousel attribute), 724, 888  
 index (kivy.uix.filechooser.FileChooserProgressBase attribute), 740, 910  
 index (kivy.uix.listview.SelectableView attribute), 767, 945  
 indices (kivy.graphics.Mesh attribute), 423  
 indices (kivy.graphics.vertex\_instructions.Mesh attribute), 470  
 init\_camera() (kivy.core.camera.CameraBase method), 325, 347  
 init\_gesture() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 input\_filter (kivy.uix.textinput.TextInput attribute), 821, 1029  
 input\_type (kivy.uix.textinput.TextInput attribute), 822, 1029  
 InputPostprocCalibration (class in kivy.input.postproc.calibration), 521, 545, 549  
 InputPostprocDejitter (class in kivy.input.postproc.dejitter), 522, 546, 551  
 InputPostprocDoubleTap (class in kivy.input.postproc.doubletap), 522, 546, 553  
 InputPostprocIgnoreList (class in kivy.input.postproc.ignorelist), 522, 546, 555  
 InputPostprocRetainTouch (class in kivy.input.postproc.retaintouch), 523, 547, 557  
 InputPostprocTripleTap (class in kivy.input.postproc.tripletap), 523, 547, 559  
 insert() (kivy.graphics.InstructionGroup method), 418  
 insert() (kivy.graphics.instructions.InstructionGroup method), 430, 487  
 insert\_text() (kivy.uix.textinput.TextInput method), 822, 1030  
 inside\_group (kivy.uix.actionbar.ActionItem attribute), 704, 863  
 install\_android() (in module kivy.support), 243, 697  
 install\_gobject\_iteration() (in module kivy.support), 243, 697  
 install\_twisted\_reactor() (in module kivy.support), 243, 697  
 Instruction (class in kivy.graphics), 418  
 Instruction (class in kivy.graphics.instructions), 429, 487  
 InstructionGroup (class in kivy.graphics), 418  
 InstructionGroup (class in kivy.graphics.instructions), 430, 487

InteractiveLauncher (class in kivy.interactive), 207, 601

interface (kivy.uix.settings.Settings attribute), 796, 994

interface\_cls (kivy.uix.settings.Settings attribute), 796, 994

InterfaceWithSidebar (class in kivy.uix.settings), 799, 997

interpolate() (in module kivy.utils), 244, 1068

intersection() (in module kivy.utils), 243, 1067

inverse() (kivy.graphics.transformation.Matrix method), 462, 515

InvertEffect (class in kivy.uix.effectwidget), 734, 903

is\_active (kivy.uix.screenmanager.TransitionBase attribute), 787, 981

is\_color\_transparent() (in module kivy.utils), 244, 1067

is\_dead() (kivy.weakmethod.WeakMethod method), 249, 1075

is\_dir() (kivy.uix.filechooser.FileSystemAbstract method), 740, 910

is\_double\_tap (kivy.input.motionevent.MotionEvent attribute), 533, 541

is\_event\_type() (kivy.event.EventDispatcher method), 200, 397

is\_finished (kivy.network.urlrequest.UrlRequest attribute), 663, 667

is\_focusable (kivy.uix.behaviors.FocusBehavior attribute), 712, 872

is\_hidden() (kivy.uix.filechooser.FileSystemAbstract method), 740, 910

is\_leaf (kivy.uix.treeview.TreeViewNode attribute), 829, 1041

is\_loaded (kivy.uix.treeview.TreeViewNode attribute), 829, 1041

is\_manual (kivy.effects.kinetic.KineticEffect attribute), 384, 389

is\_mouse\_scrolling (kivy.input.MotionEvent attribute), 520

is\_mouse\_scrolling (kivy.input.motionevent.MotionEvent attribute), 533, 541

is\_open (kivy.uix.spinner.Spinner attribute), 803, 1004

is\_open (kivy.uix.treeview.TreeViewNode attribute), 829, 1041

is\_selected (kivy.adapters.models.SelectableDataItem attribute), 256, 269

is\_selected (kivy.uix.listview.SelectableView attribute), 767, 945

is\_selected (kivy.uix.treeview.TreeViewNode attribute), 830, 1041

is\_touch (kivy.input.motionevent.MotionEvent attribute), 533, 541

is\_triple\_tap (kivy.input.motionevent.MotionEvent attribute), 533, 541

italic (kivy.uix.label.Label attribute), 755, 930

item\_strings (kivy.uix.listview.ListView attribute), 769, 947

iterate\_all\_nodes() (kivy.uix.treeview.TreeView method), 827, 1039

iterate\_mipmaps() (kivy.core.image.ImageData method), 329, 356

iterate\_open\_nodes() (kivy.uix.treeview.TreeView method), 827, 1039

## J

joint (kivy.graphics.Line attribute), 421

joint (kivy.graphics.vertex\_instructions.Line attribute), 468

joint\_precision (kivy.graphics.Line attribute), 421

joint\_precision (kivy.graphics.vertex\_instructions.Line attribute), 468

JsonStore (class in kivy.storage.jsonstore), 688, 693

## K

keep\_data (kivy.uix.image.Image attribute), 749, 924

keep\_ratio (kivy.uix.image.Image attribute), 750, 924

keep\_within\_parent (kivy.uix.splitter.Splitter attribute), 805, 1006

key (kivy.uix.settings.SettingItem attribute), 797, 995

key\_background\_color (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1053

key\_background\_down (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1054

key\_background\_normal (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1054

key\_border (kivy.uix.vkeyboard.VKeyboard attribute), 833, 1054

key\_disabled\_background\_normal (kivy.uix.vkeyboard.VKeyboard attribute), 834, 1054

key\_margin (kivy.uix.vkeyboard.VKeyboard attribute), 834, 1054

Keyboard (class in kivy.core.window), 337, 375

keyboard (kivy.uix.behaviors.FocusBehavior attribute), 712, 872

keyboard\_height (kivy.core.window.WindowBase attribute), 340, 378

keyboard\_mode (kivy.uix.textinput.TextInput attribute), 822, 1030

keyboard\_on\_key\_down() (kivy.uix.behaviors.FocusBehavior method), 712, 872

keyboard\_on\_key\_up()  
     (kivy.uix.behaviors.FocusBehavior  
     method), 712, 873

keyboard\_select (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 714, 875

keyboard\_suggestions  
     (kivy.uix.textinput.TextInput attribute),  
     822, 1030

keycode\_to\_string() (kivy.core.window.Keyboard  
     method), 338, 376

keys() (kivy.storage.AbstractStore method), 688

KineticEffect (class in kivy.effects.kinetic), 384, 389

kivy (module), 159

kivy.adapters (module), 251

kivy.adapters.adapter (module), 251, 259

kivy.adapters.args\_converters (module), 253, 261

kivy.adapters.dictadapter (module), 252, 263

kivy.adapters.listadapter (module), 253, 265

kivy.adapters.models (module), 256, 269

kivy.adapters.simplelistadapter (module), 256, 271

kivy.animation (module), 160, 273

kivy.app (module), 170, 285

kivy.atlas (module), 183, 297

kivy.base (module), 207, 301

kivy.cache (module), 186, 305

kivy.clock (module), 187, 307

kivy.compat (module), 192, 313

kivy.config (module), 192, 315

kivy.context (module), 197, 321

kivy.core (module), 323

kivy.core.audio (module), 323, 345

kivy.core.camera (module), 325, 347

kivy.core.clipboard (module), 325, 349

kivy.core.gl (module), 326, 351

kivy.core.image (module), 326, 353

kivy.core.spelling (module), 329, 357

kivy.core.text (module), 330, 359

kivy.core.text.markup (module), 332, 361, 367

kivy.core.text.text\_layout (module), 334, 362, 369

kivy.core.video (module), 336, 373

kivy.core.window (module), 337, 375

kivy.effects (module), 383

kivy.effects.dampedscroll (module), 383, 387

kivy.effects.kinetic (module), 384, 389

kivy.effects.opacityscroll (module), 385, 391

kivy.effects.scroll (module), 385, 393

kivy.event (module), 197, 395

kivy.ext (module), 401

kivy.factory (module), 201, 403

kivy.garden (module), 202, 405

kivy.geometry (module), 203, 407

kivy.gesture (module), 203, 409

kivy.graphics (module), 411

kivy.graphics.compiler (module), 441, 473

kivy.graphics.context (module), 437, 475

kivy.graphics.context\_instructions (module), 434,  
     477

kivy.graphics.fbo (module), 438, 481

kivy.graphics.gl\_instructions (module), 440, 485

kivy.graphics.instructions (module), 429, 487

kivy.graphics.opengl (module), 442, 493

kivy.graphics.opengl\_utils (module), 451, 503

kivy.graphics.shader (module), 452, 505

kivy.graphics.stencil\_instructions (module), 454,  
     507

kivy.graphics.texture (module), 456, 509

kivy.graphics.transformation (module), 461, 515

kivy.graphics.vertex\_instructions (module), 464

kivy.input (module), 519

kivy.input.factory (module), 535, 537

kivy.input.motionevent (module), 531, 539

kivy.input.postproc (module), 521, 545

kivy.input.postproc.calibration (module), 521, 545,  
     549

kivy.input.postproc.dejitter (module), 522, 546, 551

kivy.input.postproc.doubletap (module), 522, 546,  
     553

kivy.input.postproc.ignorelist (module), 522, 546,  
     555

kivy.input.postproc.retaintouch (module), 523,  
     547, 557

kivy.input.postproc.tripletap (module), 523, 547,  
     559

kivy.input.provider (module), 535, 561

kivy.input.providers (module), 523, 563

kivy.input.providers.androidjoystick (module),  
     523, 563, 569

kivy.input.providers.hidinput (module), 525, 565,  
     571

kivy.input.providers.leapfinger (module), 524,  
     564, 573

kivy.input.providers.linuxwacom (module), 526,  
     566, 575

kivy.input.providers.mactouch (module), 526, 566,  
     577

kivy.input.providers.mouse (module), 524, 564,  
     579

kivy.input.providers.mtdev (module), 526, 565,  
     581

kivy.input.providers.probesysfs (module), 523,  
     563, 583

kivy.input.providers.tuio (module), 527, 567, 585

kivy.input.providers.wm\_common (module), 524,  
     564, 587

kivy.input.providers.wm\_pen (module), 527, 566,  
     589

kivy.input.providers.wm\_touch (module), 527,  
     567, 591

kivy.input.recorder (module), 529, 593

kivy.input.shape (module), 536, 597



- kivy.interactive (module), 205, 599
- kivy.lang (module), 210, 603
- kivy.lib (module), 617
- kivy.lib.gstplayer (module), 617, 619
- kivy.loader (module), 181, 621
- kivy.logger (module), 221, 625
- kivy.metrics (module), 222, 627
- kivy.modules (module), 631
- kivy.modules.inspector (module), 632, 637
- kivy.modules.keybinding (module), 633, 639
- kivy.modules.monitor (module), 634, 641
- kivy.modules.recorder (module), 634, 643
- kivy.modules.screen (module), 635, 645
- kivy.modules.touchring (module), 635, 647
- kivy.modules.webdebugger (module), 636, 649
- kivy.multistroke (module), 224, 651
- kivy.network (module), 661
- kivy.network.urlrequest (module), 661, 665
- kivy.parser (module), 232, 669
- kivy.properties (module), 232, 671
- kivy.resources (module), 243, 683
- kivy.storage (module), 685
- kivy.storage.dictstore (module), 688, 691
- kivy.storage.jsonstore (module), 688, 693
- kivy.storage.redisstore (module), 689, 695
- kivy.support (module), 243, 697
- kivy.uix (module), 699
- kivy.uix.abstractview (module), 699, 855
- kivy.uix.accordion (module), 700, 857
- kivy.uix.actionbar (module), 704, 863
- kivy.uix.anchorlayout (module), 708, 867
- kivy.uix.behaviors (module), 709, 869
- kivy.uix.boxlayout (module), 716, 877
- kivy.uix.bubble (module), 717, 879
- kivy.uix.button (module), 721, 883
- kivy.uix.camera (module), 722, 885
- kivy.uix.carousel (module), 723, 887
- kivy.uix.checkbox (module), 725, 891
- kivy.uix.codeinput (module), 726, 893
- kivy.uix.colorpicker (module), 727, 895
- kivy.uix.dropdown (module), 729, 897
- kivy.uix.effectwidget (module), 731, 901
- kivy.uix.filechooser (module), 735, 905
- kivy.uix.floatlayout (module), 740, 911
- kivy.uix.gesturesurface (module), 741, 913
- kivy.uix.gridlayout (module), 745, 917
- kivy.uix.image (module), 748, 923
- kivy.uix.label (module), 751, 927
- kivy.uix.layout (module), 758, 935
- kivy.uix.listview (module), 759, 937
- kivy.uix.modalview (module), 769, 949
- kivy.uix.pagelayout (module), 771, 953
- kivy.uix.popup (module), 772, 955
- kivy.uix.progressbar (module), 774, 959
- kivy.uix.relativelayout (module), 775, 961
- kivy.uix.rst (module), 850, 965
- kivy.uix.sandbox (module), 777, 969
- kivy.uix.scatter (module), 778, 971
- kivy.uix.scatterlayout (module), 781, 975
- kivy.uix.screenmanager (module), 782, 977
- kivy.uix.scrollview (module), 789, 985
- kivy.uix.settings (module), 793, 991
- kivy.uix.slider (module), 800, 999
- kivy.uix.spinner (module), 802, 1003
- kivy.uix.splitter (module), 804, 1005
- kivy.uix.stacklayout (module), 806, 1007
- kivy.uix.stencilview (module), 807, 1011
- kivy.uix.switch (module), 808, 1013
- kivy.uix.tabbedpanel (module), 809, 1015
- kivy.uix.textinput (module), 815, 1023
- kivy.uix.togglebutton (module), 825, 1035
- kivy.uix.treeview (module), 825, 1037
- kivy.uix.video (module), 835, 1043
- kivy.uix.videoplayer (module), 836, 1045
- kivy.uix.vkeyboard (module), 831, 1051
- kivy.uix.widget (module), 841, 1057
- kivy.utils (module), 243, 1067
- kivy.vector (module), 246, 1071
- kivy.weakmethod (module), 249, 1075
- kivy\_base\_dir (in module kivy), 160
- kivy\_config\_fn (in module kivy), 160
- kivy\_configure() (in module kivy), 159
- kivy\_data\_dir (in module kivy), 160
- kivy\_home\_dir (in module kivy), 160
- kivy\_icons\_dir (in module kivy), 160
- kivy\_modules\_dir (in module kivy), 160
- kivy\_options (in module kivy), 159
- kivy\_register\_post\_configuration() (in module kivy), 159
- kivy\_shader\_dir (in module kivy), 160
- kivy\_userexts\_dir (in module kivy), 160
- kivy\_usermodules\_dir (in module kivy), 160
- kv\_directory (kivy.app.App attribute), 178, 293
- kv\_file (kivy.app.App attribute), 178, 293

## L

- Label (class in kivy.uix.label), 753, 929
- label (kivy.core.text.LabelBase attribute), 332, 361
- LabelBase (class in kivy.core.text), 330, 359
- last\_touch (kivy.uix.behaviors.ButtonBehavior attribute), 709, 869
- Layout (class in kivy.uix.layout), 759, 935
- layout (kivy.uix.vkeyboard.VKeyboard attribute), 834, 1054
- layout\_path (kivy.uix.vkeyboard.VKeyboard attribute), 834, 1054
- layout\_text() (in module kivy.core.text.text\_layout), 334, 362, 369

LayoutLine (class in kivy.core.text.text\_layout), 335, 364, 371  
 LayoutWord (class in kivy.core.text.text\_layout), 335, 364, 370  
 length (kivy.core.audio.Sound attribute), 324, 345  
 length() (kivy.vector.Vector method), 248, 1073  
 length2() (kivy.vector.Vector method), 248, 1073  
 level (kivy.uix.treeview.TreeViewNode attribute), 830, 1041  
 lexer (kivy.uix.codeinput.CodeInput attribute), 727, 893  
 limit\_to (kivy.uix.bubble.Bubble attribute), 720, 881  
 Line (class in kivy.graphics), 418  
 Line (class in kivy.graphics.vertex\_instructions), 465  
 line\_height (kivy.uix.label.Label attribute), 755, 931  
 line\_height (kivy.uix.textinput.TextInput attribute), 822, 1030  
 line\_intersection() (kivy.vector.Vector static method), 248, 1073  
 line\_spacing (kivy.uix.textinput.TextInput attribute), 822, 1030  
 linear() (kivy.animation.AnimationTransition static method), 167, 280  
 link() (kivy.properties.Property method), 236, 674  
 list() (kivy.input.factory.MotionEventFactory static method), 535, 537  
 list() (kivy.input.MotionEventFactory static method), 521  
 list\_languages() (kivy.core.spelling.SpellingBase method), 329, 357  
 ListAdapter (class in kivy.adapters.listadapter), 254, 265  
 listdir() (kivy.uix.filechooser.FileSystemAbstract method), 740, 910  
 ListItemButton (class in kivy.uix.listview), 768, 946  
 ListItemLabel (class in kivy.uix.listview), 768, 946  
 ListProperty (class in kivy.properties), 237, 675  
 ListView (class in kivy.uix.listview), 768, 946  
 load() (in module kivy.ext), 401  
 load() (kivy.core.audio.Sound method), 324, 345  
 load() (kivy.core.audio.SoundLoader static method), 324, 346  
 load() (kivy.core.image.Image static method), 327, 354  
 load() (kivy.core.video.VideoBase method), 336, 373  
 load\_config() (kivy.app.App method), 178, 293  
 load\_file() (kivy.lang.BuilderBase method), 220, 614  
 load\_func (kivy.uix.treeview.TreeView attribute), 828, 1039  
 load\_kv() (kivy.app.App method), 179, 293  
 load\_next() (kivy.uix.carousel.Carousel method), 724, 888  
 load\_previous() (kivy.uix.carousel.Carousel method), 724, 888  
 load\_slide() (kivy.uix.carousel.Carousel method), 724, 888  
 load\_string() (kivy.lang.BuilderBase method), 220, 614  
 loaded (kivy.uix.video.Video attribute), 835, 1043  
 LoaderBase (class in kivy.loader), 182, 621  
 LoadIdentity (class in kivy.graphics), 429  
 loading\_image (kivy.loader.LoaderBase attribute), 182, 622  
 Logger (in module kivy.logger), 222, 626  
 LoggerHistory (class in kivy.logger), 222, 626  
 look\_at() (kivy.graphics.transformation.Matrix method), 462, 515  
 loop (kivy.core.audio.Sound attribute), 324, 345  
 loop (kivy.uix.carousel.Carousel attribute), 724, 888

## M

mag\_filter (kivy.graphics.texture.Texture attribute), 460, 513  
 mainthread() (in module kivy.clock), 192, 311  
 manager (kivy.uix.screenmanager.Screen attribute), 784, 979  
 manager (kivy.uix.screenmanager.TransitionBase attribute), 787, 981  
 margin\_hint (kivy.uix.vkeyboard.VKeyboard attribute), 834, 1054  
 markup (kivy.core.text.markup.MarkupLabel attribute), 333, 362, 367  
 markup (kivy.uix.label.Label attribute), 755, 931  
 MarkupLabel (class in kivy.core.text.markup), 333, 362, 367  
 match() (kivy.lang.BuilderBase method), 220, 614  
 match\_candidate() (kivy.multistroke.MultistrokeGesture method), 230, 657  
 Matrix (class in kivy.graphics.transformation), 461, 515  
 matrix (kivy.graphics.context\_instructions.MatrixInstruction attribute), 437, 480  
 matrix (kivy.graphics.MatrixInstruction attribute), 422  
 MatrixInstruction (class in kivy.graphics), 422  
 MatrixInstruction (class in kivy.graphics.context\_instructions), 437, 480  
 max (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
 max (kivy.uix.progressbar.ProgressBar attribute), 774, 959  
 max (kivy.uix.slider.Slider attribute), 801, 999  
 max\_height (kivy.uix.dropdown.DropDown attribute), 731, 899

max\_history (kivy.effects.kinetic.KineticEffect attribute), 384, 389  
 max\_iteration (kivy.clock.ClockBase attribute), 191, 310  
 max\_lines (kivy.uix.label.Label attribute), 755, 931  
 max\_size (kivy.uix.splitter.Splitter attribute), 805, 1006  
 max\_upload\_per\_frame (kivy.loader.LoaderBase attribute), 182, 622  
 menu (kivy.uix.settings.InterfaceWithSidebar attribute), 799, 997  
 merge\_gestures() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 Mesh (class in kivy.graphics), 423  
 Mesh (class in kivy.graphics.vertex\_instructions), 469  
 Metrics (in module kivy.metrics), 223, 628  
 metrics (in module kivy.metrics), 224, 629  
 MetricsBase (class in kivy.metrics), 223, 628  
 min (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
 min (kivy.uix.slider.Slider attribute), 801, 999  
 min\_distance (kivy.effects.kinetic.KineticEffect attribute), 384, 390  
 min\_filter (kivy.graphics.texture.Texture attribute), 460, 513  
 min\_move (kivy.uix.carousel.Carousel attribute), 724, 888  
 min\_overscroll (kivy.effects.dampedscroll.DampedScrollEffect attribute), 383, 387  
 min\_size (kivy.uix.splitter.Splitter attribute), 805, 1006  
 min\_space (kivy.uix.accordion.Accordion attribute), 701, 859  
 min\_space (kivy.uix.accordion.AccordionItem attribute), 703, 860  
 min\_velocity (kivy.effects.kinetic.KineticEffect attribute), 385, 390  
 minimum\_bounding\_circle() (in module kivy.geometry), 203, 407  
 minimum\_height (kivy.uix.gridlayout.GridLayout attribute), 747, 919  
 minimum\_height (kivy.uix.stacklayout.StackLayout attribute), 807, 1008  
 minimum\_height (kivy.uix.textinput.TextInput attribute), 822, 1030  
 minimum\_height (kivy.uix.treeview.TreeView attribute), 828, 1039  
 minimum\_size (kivy.uix.gridlayout.GridLayout attribute), 747, 920  
 minimum\_size (kivy.uix.stacklayout.StackLayout attribute), 807, 1008  
 minimum\_size (kivy.uix.treeview.TreeView attribute), 828, 1039  
 minimum\_width (kivy.uix.actionbar.ActionItem attribute), 704, 864  
 minimum\_width (kivy.uix.gridlayout.GridLayout attribute), 747, 920  
 minimum\_width (kivy.uix.stacklayout.StackLayout attribute), 807, 1008  
 minimum\_width (kivy.uix.treeview.TreeView attribute), 828, 1040  
 mipmap (kivy.graphics.texture.Texture attribute), 460, 514  
 mipmap (kivy.uix.actionbar.ActionItem attribute), 704, 864  
 mipmap (kivy.uix.image.Image attribute), 750, 924  
 mipmap (kivy.uix.label.Label attribute), 755, 931  
 mipmaps (kivy.core.image.ImageData attribute), 329, 356  
 mm() (in module kivy.metrics), 224, 629  
 ModalView (class in kivy.uix.modalview), 770, 950  
 mode (kivy.graphics.Mesh attribute), 423  
 mode (kivy.graphics.vertex\_instructions.Mesh attribute), 470  
 mode (kivy.uix.actionbar.ActionGroup attribute), 705, 865  
 modifiers (kivy.core.window.WindowBase attribute), 340, 378  
 MonochromeEffect (class in kivy.uix.effectwidget), 734, 903  
 MotionEvent (class in kivy.input), 519  
 MotionEvent (class in kivy.input.motionevent), 532, 540  
 MotionEventFactory (class in kivy.input), 520  
 MotionEventFactory (class in kivy.input.factory), 535, 537  
 MotionEventProvider (class in kivy.input), 520  
 MotionEventProvider (class in kivy.input.provider), 535, 561  
 mouse\_pos (kivy.core.window.WindowBase attribute), 340, 378  
 move() (kivy.input.MotionEvent method), 520  
 move() (kivy.input.motionevent.MotionEvent method), 533, 541  
 multiline (kivy.uix.textinput.TextInput attribute), 822, 1030  
 multiply() (kivy.graphics.transformation.Matrix method), 462, 516  
 multiselect (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 714, 875  
 multiselect (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 MultistrokeGesture (class in kivy.multistroke), 229, 656

## N

name (kivy.app.App attribute), 179, 294  
 name (kivy.config.ConfigParser attribute), 196, 318  
 name (kivy.uix.screenmanager.Screen attribute), 784, 979

next() (kivy.uix.screenmanager.ScreenManager method), 785, 980  
 next\_slide (kivy.uix.carousel.Carousel attribute), 725, 888  
 no\_selection (kivy.uix.treeview.TreeViewNode attribute), 830, 1041  
 nocache (kivy.core.image.Image attribute), 327, 354  
 nocache (kivy.uix.image.Image attribute), 750, 924  
 nodes (kivy.uix.treeview.TreeViewNode attribute), 830, 1041  
 NoLanguageSelectedError (class in kivy.core.spelling), 330, 358  
 norm\_image\_size (kivy.uix.image.Image attribute), 750, 924  
 normal\_matrix() (kivy.graphics.transformation.Matrix method), 462, 516  
 normalize() (kivy.gesture.Gesture method), 204, 409  
 normalize() (kivy.vector.Vector method), 248, 1073  
 normalize\_stroke() (kivy.gesture.GestureStroke method), 204, 410  
 NoSuchLangError (class in kivy.core.spelling), 330, 357  
 NoTransition (class in kivy.uix.screenmanager), 788, 983  
 num\_workers (kivy.loader.LoaderBase attribute), 183, 622  
 NumericProperty (class in kivy.properties), 236, 675  
**O**  
 ObjectProperty (class in kivy.properties), 237, 676  
 ObjectWithUid (class in kivy.event), 201, 398  
 Observable (class in kivy.lang), 219, 613  
 odd (kivy.uix.treeview.TreeViewNode attribute), 830, 1042  
 odd\_color (kivy.uix.treeview.TreeViewNode attribute), 830, 1042  
 on\_bring\_to\_front() (kivy.uix.scatter.Scatter method), 780, 973  
 on\_close() (kivy.core.window.WindowBase method), 340, 378  
 on\_config\_change() (kivy.app.App method), 179, 294  
 on\_context\_created() (kivy.uix.sandbox.Sandbox method), 778, 969  
 on\_current\_uid() (kivy.uix.settings.ContentPanel method), 800, 998  
 on\_double\_tap() (kivy.uix.textinput.TextInput method), 822, 1030  
 on\_dropfile() (kivy.core.window.WindowBase method), 340, 378  
 on\_exception() (kivy.uix.sandbox.Sandbox method), 778, 969  
 on\_flip() (kivy.core.window.WindowBase method), 340, 378  
 on\_key\_down() (kivy.core.window.WindowBase method), 341, 378  
 on\_key\_up() (kivy.core.window.WindowBase method), 341, 378  
 on\_keyboard() (kivy.core.window.WindowBase method), 341, 378  
 on\_motion() (kivy.core.window.WindowBase method), 341, 378  
 on\_mouse\_down() (kivy.core.window.WindowBase method), 341, 379  
 on\_mouse\_move() (kivy.core.window.WindowBase method), 341, 379  
 on\_mouse\_up() (kivy.core.window.WindowBase method), 341, 379  
 on\_pause() (kivy.app.App method), 179, 294  
 on\_pause() (kivy.base.EventLoopBase method), 208, 301  
 on\_quad\_touch() (kivy.uix.textinput.TextInput method), 822, 1030  
 on\_request\_close() (kivy.core.window.WindowBase method), 341, 379  
 on\_resize() (kivy.core.window.WindowBase method), 341, 379  
 on\_resume() (kivy.app.App method), 179, 294  
 on\_rotate() (kivy.core.window.WindowBase method), 341, 379  
 on\_selection\_change() (kivy.adapters.listadapter.ListAdapter method), 254, 266  
 on\_start() (kivy.app.App method), 180, 294  
 on\_start() (kivy.base.EventLoopBase method), 208, 301  
 on\_stop() (kivy.app.App method), 180, 294  
 on\_stop() (kivy.base.EventLoopBase method), 208, 302  
 on\_texture() (kivy.core.image.Image method), 327, 354  
 on\_touch\_down() (kivy.core.window.WindowBase method), 341, 379  
 on\_touch\_down() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 on\_touch\_down() (kivy.uix.widget.Widget method), 847, 1062  
 on\_touch\_move() (kivy.core.window.WindowBase method), 341, 379  
 on\_touch\_move() (kivy.uix.gesturesurface.GestureSurface method), 743, 914  
 on\_touch\_move() (kivy.uix.widget.Widget method), 847, 1062  
 on\_touch\_up() (kivy.core.window.WindowBase method), 341, 379  
 on\_touch\_up() (kivy.uix.widget.Widget method), 847, 1062



on\_transform\_with\_touch() (kivy.uix.scatter.Scatter method), 780, 973  
 on\_triple\_tap() (kivy.uix.textinput.TextInput method), 823, 1030  
 opacity (kivy.graphics.Canvas attribute), 414  
 opacity (kivy.graphics.instructions.Canvas attribute), 432, 489  
 opacity (kivy.uix.widget.Widget attribute), 847, 1062  
 OpacityScrollEffect (class in kivy.effects.opacityscroll), 385, 391  
 open() (kivy.uix.dropdown.DropDown method), 731, 899  
 open() (kivy.uix.modalview.ModalView method), 771, 951  
 open\_settings() (kivy.app.App method), 180, 294  
 opos (kivy.input.MotionEvent attribute), 520  
 opos (kivy.input.motionEvent.MotionEvent attribute), 533, 541  
 option\_cls (kivy.uix.spinner.Spinner attribute), 803, 1004  
 OptionProperty (class in kivy.properties), 239, 677  
 options (kivy.app.App attribute), 180, 294  
 options (kivy.properties.OptionProperty attribute), 239, 678  
 options (kivy.uix.settings.SettingOptions attribute), 798, 996  
 options (kivy.uix.video.Video attribute), 835, 1043  
 options (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048  
 order (kivy.uix.effectwidget.ChannelMixEffect attribute), 734, 904  
 orientation (kivy.uix.accordian.Accordian attribute), 701, 859  
 orientation (kivy.uix.accordian.AccordianItem attribute), 703, 860  
 orientation (kivy.uix.boxlayout.BoxLayout attribute), 717, 878  
 orientation (kivy.uix.bubble.Bubble attribute), 720, 881  
 orientation (kivy.uix.slider.Slider attribute), 801, 999  
 orientation (kivy.uix.stacklayout.StackLayout attribute), 807, 1008  
 origin (kivy.graphics.context\_instructions.Rotate attribute), 436, 479  
 origin (kivy.graphics.Rotate attribute), 425  
 osx (kivy.input.motionEvent.MotionEvent attribute), 533, 541  
 osy (kivy.input.motionEvent.MotionEvent attribute), 533, 541  
 osz (kivy.input.motionEvent.MotionEvent attribute), 533, 541  
 out\_back() (kivy.animation.AnimationTransition static method), 168, 281  
 out\_bounce() (kivy.animation.AnimationTransition static method), 168, 281  
 out\_circ() (kivy.animation.AnimationTransition static method), 168, 281  
 out\_cubic() (kivy.animation.AnimationTransition static method), 168, 281  
 out\_elastic() (kivy.animation.AnimationTransition static method), 169, 282  
 out\_expo() (kivy.animation.AnimationTransition static method), 169, 282  
 out\_quad() (kivy.animation.AnimationTransition static method), 169, 282  
 out\_quart() (kivy.animation.AnimationTransition static method), 169, 282  
 out\_quint() (kivy.animation.AnimationTransition static method), 170, 283  
 out\_sine() (kivy.animation.AnimationTransition static method), 170, 283  
 overdraw\_width (kivy.graphics.vertex\_instructions.SmoothLine attribute), 471  
 overflow\_group (kivy.uix.actionbar.ActionView attribute), 706, 865  
 overflow\_image (kivy.uix.actionbar.ActionOverflow attribute), 706, 865  
 overscroll (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
 ox (kivy.input.motionEvent.MotionEvent attribute), 534, 542  
 oy (kivy.input.motionEvent.MotionEvent attribute), 534, 542  
 oz (kivy.input.motionEvent.MotionEvent attribute), 534, 542

## P

padding (kivy.uix.anchorlayout.AnchorLayout attribute), 708, 868  
 padding (kivy.uix.boxlayout.BoxLayout attribute), 717, 878  
 padding (kivy.uix.gridlayout.GridLayout attribute), 747, 920  
 padding (kivy.uix.label.Label attribute), 755, 931  
 padding (kivy.uix.slider.Slider attribute), 801, 999  
 padding (kivy.uix.stacklayout.StackLayout attribute), 807, 1008  
 padding (kivy.uix.textinput.TextInput attribute), 823, 1031  
 padding\_x (kivy.uix.label.Label attribute), 755, 931  
 padding\_x (kivy.uix.textinput.TextInput attribute), 823, 1031  
 padding\_y (kivy.uix.label.Label attribute), 755, 931  
 padding\_y (kivy.uix.textinput.TextInput attribute), 823, 1031  
 page (kivy.uix.pagelayout.PageLayout attribute), 772, 953

page\_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 715, 875  
 PageLayout (class in kivy.uix.pagelayout), 771, 953  
 panel (kivy.uix.settings.SettingItem attribute), 797, 995  
 panels (kivy.uix.settings.ContentPanel attribute), 800, 998  
 parent (kivy.core.window.WindowBase attribute), 342, 379  
 parent (kivy.uix.widget.Widget attribute), 847, 1063  
 parent\_node (kivy.uix.treeview.TreeViewNode attribute), 830, 1042  
 parse() (kivy.lang.Parser method), 221, 614  
 parse\_bool() (in module kivy.parser), 232, 669  
 parse\_color() (in module kivy.parser), 232, 669  
 parse\_filename() (in module kivy.parser), 232, 669  
 parse\_float (in module kivy.parser), 232, 669  
 parse\_float4() (in module kivy.parser), 232, 669  
 parse\_gesture() (kivy.multistroke.Recognizer method), 227, 654  
 parse\_int (in module kivy.parser), 232, 669  
 parse\_int2() (in module kivy.parser), 232, 669  
 parse\_level() (kivy.lang.Parser method), 221, 614  
 parse\_string() (in module kivy.parser), 232, 669  
 Parser (class in kivy.lang), 220, 614  
 ParserException (class in kivy.lang), 221, 615  
 password (kivy.uix.textinput.TextInput attribute), 823, 1031  
 paste() (kivy.uix.textinput.TextInput method), 823, 1031  
 path (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 path (kivy.uix.filechooser.FileChooserProgressBase attribute), 740, 910  
 pause() (kivy.core.video.VideoBase method), 336, 373  
 pause() (kivy.loader.LoaderBase method), 183, 622  
 permute() (kivy.multistroke.MultistrokeGesture method), 230, 657  
 perspective() (kivy.graphics.transformation.Matrix method), 462, 516  
 pixel\_size (kivy.uix.effectwidget.PixelateEffect attribute), 734, 904  
 PixelateEffect (class in kivy.uix.effectwidget), 734, 904  
 pixels (kivy.graphics.Fbo attribute), 417  
 pixels (kivy.graphics.fbo.Fbo attribute), 440, 483  
 pixels (kivy.graphics.texture.Texture attribute), 460, 514  
 platform (in module kivy.utils), 245, 1068  
 play (kivy.input.recorder.Recorder attribute), 530, 594  
 play (kivy.uix.camera.Camera attribute), 723, 885  
 play (kivy.uix.video.Video attribute), 836, 1043  
 play (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048  
 play() (kivy.core.audio.Sound method), 324, 346  
 play() (kivy.core.video.VideoBase method), 337, 373  
 Point (class in kivy.graphics), 423  
 Point (class in kivy.graphics.vertex\_instructions), 469  
 points (kivy.graphics.Bezier attribute), 412  
 points (kivy.graphics.Line attribute), 421  
 points (kivy.graphics.Point attribute), 423  
 points (kivy.graphics.Quad attribute), 424  
 points (kivy.graphics.Triangle attribute), 427  
 points (kivy.graphics.vertex\_instructions.Bezier attribute), 470  
 points (kivy.graphics.vertex\_instructions.Line attribute), 468  
 points (kivy.graphics.vertex\_instructions.Point attribute), 469  
 points (kivy.graphics.vertex\_instructions.Quad attribute), 464  
 points (kivy.graphics.vertex\_instructions.Triangle attribute), 464  
 points\_distance() (kivy.gesture.GestureStroke method), 205, 410  
 pointsize (kivy.graphics.Point attribute), 423  
 pointsize (kivy.graphics.vertex\_instructions.Point attribute), 469  
 pop() (kivy.input.MotionEvent method), 520  
 pop() (kivy.input.motionevent.MotionEvent method), 534, 542  
 PopMatrix (class in kivy.graphics), 424  
 PopMatrix (class in kivy.graphics.context\_instructions), 435, 478  
 PopState (class in kivy.graphics), 429  
 Popup (class in kivy.uix.popup), 773, 956  
 popup (kivy.uix.settings.SettingOptions attribute), 798, 996  
 popup (kivy.uix.settings.SettingPath attribute), 798, 995  
 popup (kivy.uix.settings.SettingString attribute), 797, 995  
 PopupException (class in kivy.uix.popup), 774, 957  
 pos (kivy.graphics.Rectangle attribute), 424  
 pos (kivy.graphics.vertex\_instructions.Rectangle attribute), 465  
 pos (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 pos (kivy.uix.widget.Widget attribute), 847, 1063  
 pos\_hint (kivy.uix.widget.Widget attribute), 847, 1063  
 position (kivy.core.video.VideoBase attribute), 337, 373

position (kivy.uix.video.Video attribute), 836, 1044  
 position (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1048  
 post\_dispatch\_input() (kivy.base.EventLoopBase method), 208, 302  
 ppos (kivy.input.MotionEvent attribute), 520  
 ppos (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 preload() (kivy.uix.rst.RstDocument method), 852, 967  
 prepare() (kivy.multistroke.Candidate method), 232, 659  
 prepare() (kivy.multistroke.UnistrokeTemplate method), 231, 658  
 prepare\_templates() (kivy.multistroke.Recognizer method), 227, 654  
 previous() (kivy.uix.screenmanager.ScreenManager method), 785, 980  
 previous\_image (kivy.uix.actionbar.ActionPrevious attribute), 707, 866  
 previous\_slide (kivy.uix.carousel.Carousel attribute), 725, 888  
 print\_usage() (kivy.cache.Cache static method), 187, 306  
 profile (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 progress (kivy.multistroke.ProgressTracker attribute), 229, 656  
 progress\_cls (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 ProgressBar (class in kivy.uix.progressbar), 774, 959  
 ProgressTracker (class in kivy.multistroke), 228, 655  
 project() (kivy.graphics.transformation.Matrix method), 463, 516  
 propagate\_selection\_to\_data (kivy.adapters.listadapter.ListAdapter attribute), 254, 266  
 properties() (kivy.event.EventDispatcher method), 200, 397  
 Property (class in kivy.properties), 235, 673  
 property() (kivy.event.EventDispatcher method), 200, 398  
 proxy\_ref (kivy.graphics.Instruction attribute), 418  
 proxy\_ref (kivy.graphics.instructions.Instruction attribute), 430, 487  
 proxy\_ref (kivy.uix.widget.Widget attribute), 848, 1063  
 ProxyImage (class in kivy.loader), 183, 623  
 psx (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 psy (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 psz (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 pt() (in module kivy.metrics), 223, 629  
 push() (kivy.input.MotionEvent method), 520  
 push() (kivy.input.motionevent.MotionEvent method), 534, 542  
 push\_attrs\_stack (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 PushMatrix (class in kivy.graphics), 424  
 PushMatrix (class in kivy.graphics.context\_instructions), 435, 478  
 PushState (class in kivy.graphics), 428  
 put() (kivy.storage.AbstractStore method), 688  
 px (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 py (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 PY2 (in module kivy.compat), 192, 313  
 pz (kivy.input.motionevent.MotionEvent attribute), 534, 542

## Q

Quad (class in kivy.graphics), 424  
 Quad (class in kivy.graphics.vertex\_instructions), 464  
 QueryDict (class in kivy.utils), 245, 1068

## R

r (kivy.graphics.ClearColor attribute), 428  
 r (kivy.graphics.Color attribute), 416  
 r (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 r (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485  
 r (kivy.uix.colorpicker.ColorWheel attribute), 729, 896  
 range (kivy.uix.slider.Slider attribute), 801, 1000  
 read() (kivy.config.ConfigParser method), 196, 319  
 read\_pixel() (kivy.core.image.Image method), 327, 354  
 readonly (kivy.uix.textinput.TextInput attribute), 823, 1031  
 recognize() (kivy.multistroke.Recognizer method), 228, 654  
 Recognizer (class in kivy.multistroke), 225, 652  
 record (kivy.input.recorder.Recorder attribute), 530, 594  
 record\_attrs (kivy.input.recorder.Recorder attribute), 530, 594  
 record\_profile\_mask (kivy.input.recorder.Recorder attribute), 531, 595  
 Recorder (class in kivy.input.recorder), 530, 594  
 Rectangle (class in kivy.graphics), 424  
 Rectangle (class in kivy.graphics.vertex\_instructions), 464

rectangle (kivy.graphics.Line attribute), 422  
 rectangle (kivy.graphics.vertex\_instructions.Line attribute), 468  
 RedisStore (class in kivy.storage.redisstore), 689, 695  
 ReferenceListProperty (class in kivy.properties), 239, 678  
 refresh() (kivy.core.text.LabelBase method), 332, 361  
 refresh() (kivy.uix.vkeyboard.VKeyboard method), 834, 1054  
 refresh\_fbo\_setup() (kivy.uix.effectwidget.EffectWidget method), 733, 903  
 refs (kivy.core.text.markup.MarkupLabel attribute), 333, 362, 367  
 refs (kivy.uix.label.Label attribute), 755, 931  
 register() (kivy.cache.Cache static method), 187, 306  
 register() (kivy.core.audio.SoundLoader static method), 324, 346  
 register() (kivy.core.text.LabelBase static method), 332, 361  
 register() (kivy.input.factory.MotionEventFactory static method), 535, 537  
 register() (kivy.input.MotionEventFactory static method), 521  
 register() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 528, 568, 586  
 register\_context() (in module kivy.context), 197, 321  
 register\_event\_type() (kivy.event.EventDispatcher method), 200, 398  
 register\_type() (kivy.uix.settings.Settings method), 796, 994  
 reify (class in kivy.utils), 245, 1068  
 RelativeLayout (class in kivy.uix.relativelayout), 777, 963  
 release() (kivy.core.window.Keyboard method), 338, 376  
 release() (kivy.graphics.Fbo method), 417  
 release() (kivy.graphics.fbo.Fbo method), 440, 483  
 release\_all\_keyboards() (kivy.core.window.WindowBase method), 342, 379  
 release\_keyboard() (kivy.core.window.WindowBase method), 342, 379  
 reload() (kivy.uix.image.Image method), 750, 925  
 remove() (kivy.cache.Cache static method), 187, 306  
 remove() (kivy.graphics.InstructionGroup method), 418  
 remove() (kivy.graphics.instructions.InstructionGroup method), 430, 487  
 remove\_callback() (kivy.config.ConfigParser method), 196, 319  
 remove\_event\_listener() (kivy.base.EventLoopBase method), 208, 302  
 remove\_from\_cache() (kivy.core.image.Image method), 327, 354  
 remove\_group() (kivy.graphics.InstructionGroup method), 418  
 remove\_group() (kivy.graphics.instructions.InstructionGroup method), 430, 487  
 remove\_handler() (kivy.base.ExceptionManagerBase method), 209, 302  
 remove\_input\_provider() (kivy.base.EventLoopBase method), 208, 302  
 remove\_node() (kivy.uix.treeview.TreeView method), 828, 1040  
 remove\_postproc\_module() (kivy.base.EventLoopBase method), 208, 302  
 remove\_reload\_observer() (kivy.graphics.context.Context method), 438, 475  
 remove\_reload\_observer() (kivy.graphics.Fbo method), 417  
 remove\_reload\_observer() (kivy.graphics.fbo.Fbo method), 440, 483  
 remove\_reload\_observer() (kivy.graphics.texture.Texture method), 460, 514  
 remove\_screen() (kivy.uix.screenmanager.TransitionBase method), 787, 982  
 remove\_widget() (kivy.core.window.WindowBase method), 342, 379  
 remove\_widget() (kivy.uix.widget.Widget method), 848, 1063  
 render() (kivy.core.text.LabelBase method), 332, 361  
 render() (kivy.uix.rst.RstDocument method), 852, 967  
 RenderContext (class in kivy.graphics), 424  
 RenderContext (class in kivy.graphics.instructions), 432, 490  
 representing\_cls (kivy.uix.listview.CompositeListItem attribute), 768, 946  
 req\_body (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 req\_headers (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 request\_keyboard() (kivy.core.window.WindowBase method), 342, 380  
 require() (in module kivy), 159  
 rescale\_with\_parent (kivy.uix.splitter.Splitter attribute), 805, 1006  
 reset() (kivy.effects.scroll.ScrollEffect method), 386, 393



reset\_context (kivy.graphics.Callback attribute), 413  
 reset\_context (kivy.graphics.instructions.Callback attribute), 434, 491  
 reset\_undo() (kivy.uix.textinput.TextInput method), 823, 1031  
 resolution (kivy.core.camera.CameraBase attribute), 325, 347  
 resolution (kivy.uix.camera.Camera attribute), 723, 885  
 resolve\_path() (kivy.uix.rst.RstDocument method), 852, 967  
 resource\_add\_path() (in module kivy.resources), 243, 683  
 resource\_find() (in module kivy.resources), 243, 683  
 resource\_remove\_path() (in module kivy.resources), 243, 683  
 resp\_headers (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 resp\_status (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 result (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 resume() (kivy.loader.LoaderBase method), 183, 622  
 rgb (kivy.graphics.ClearColor attribute), 428  
 rgb (kivy.graphics.Color attribute), 416  
 rgb (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 rgb (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485  
 rgba (kivy.graphics.ClearColor attribute), 428  
 rgba (kivy.graphics.Color attribute), 416  
 rgba (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 rgba (kivy.graphics.gl\_instructions.ClearColor attribute), 441, 485  
 right (kivy.uix.widget.Widget attribute), 848, 1063  
 right\_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 715, 875  
 RiseInTransition (class in kivy.uix.screenmanager), 788, 983  
 root (kivy.app.App attribute), 180, 294  
 root (kivy.uix.treeview.TreeView attribute), 828, 1040  
 root\_options (kivy.uix.treeview.TreeView attribute), 829, 1040  
 root\_window (kivy.app.App attribute), 180, 294  
 rootpath (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 Rotate (class in kivy.graphics), 425  
 Rotate (class in kivy.graphics.context\_instructions), 436, 479  
 rotate() (kivy.graphics.transformation.Matrix method), 463, 516  
 rotate() (kivy.vector.Vector method), 248, 1073  
 rotation (kivy.core.window.WindowBase attribute), 342, 380  
 rotation (kivy.uix.scatter.Scatter attribute), 781, 973  
 round\_value (kivy.effects.dampedscroll.DampedScrollEffect attribute), 383, 387  
 rounded\_rectangle (kivy.graphics.Line attribute), 422  
 rounded\_rectangle (kivy.graphics.vertex\_instructions.Line attribute), 469  
 row\_default\_height (kivy.uix.gridlayout.GridLayout attribute), 748, 920  
 row\_force\_default (kivy.uix.gridlayout.GridLayout attribute), 748, 920  
 row\_height (kivy.uix.listview.ListView attribute), 769, 947  
 rows (kivy.uix.gridlayout.GridLayout attribute), 748, 920  
 rows\_minimum (kivy.uix.gridlayout.GridLayout attribute), 748, 920  
 RstDocument (class in kivy.uix.rst), 851, 966  
 run() (kivy.app.App method), 180, 295  
 run() (kivy.base.EventLoopBase method), 208, 302  
 run() (kivy.loader.LoaderBase method), 183, 623  
 runTouchApp() (in module kivy.base), 209, 302

## S

s (kivy.graphics.Color attribute), 416  
 s (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 safeIn() (kivy.interactive.SafeMembrane method), 207, 601  
 SafeList (class in kivy.utils), 244, 1067  
 SafeMembrane (class in kivy.interactive), 207, 601  
 safeOut() (kivy.interactive.SafeMembrane method), 207, 601  
 Sandbox (class in kivy.uix.sandbox), 778, 969  
 save() (kivy.core.image.Image method), 328, 355  
 save() (kivy.graphics.texture.Texture method), 461, 514  
 Scale (class in kivy.graphics), 425  
 Scale (class in kivy.graphics.context\_instructions), 436, 479  
 scale (kivy.graphics.context\_instructions.Scale attribute), 436, 479  
 scale (kivy.graphics.Scale attribute), 425  
 scale (kivy.uix.scatter.Scatter attribute), 781, 974  
 scale() (kivy.graphics.transformation.Matrix method), 463, 516  
 scale\_for\_screen() (kivy.input.MotionEvent method), 520  
 scale\_for\_screen() (kivy.input.motionevent.MotionEvent method), 534, 542

scale\_max (kivy.uix.scatter.Scatter attribute), 781, 974  
 scale\_min (kivy.uix.scatter.Scatter attribute), 781, 974  
 scale\_stroke() (kivy.gesture.GestureStroke method), 205, 410  
 ScanlinesEffect (class in kivy.uix.effectwidget), 734, 904  
 Scatter (class in kivy.uix.scatter), 779, 972  
 ScatterLayout (class in kivy.uix.scatterlayout), 782, 975  
 ScatterPlane (class in kivy.uix.scatter), 781, 974  
 schedule\_interval() (kivy.clock.ClockBase method), 191, 311  
 schedule\_once() (kivy.clock.ClockBase method), 191, 311  
 Screen (class in kivy.uix.screenmanager), 784, 979  
 screen\_in (kivy.uix.screenmanager.TransitionBase attribute), 787, 982  
 screen\_names (kivy.uix.screenmanager.ScreenManager attribute), 785, 980  
 screen\_out (kivy.uix.screenmanager.TransitionBase attribute), 787, 982  
 ScreenManager (class in kivy.uix.screenmanager), 785, 980  
 ScreenManagerException (class in kivy.uix.screenmanager), 786, 981  
 screens (kivy.uix.screenmanager.ScreenManager attribute), 785, 980  
 screenshot() (kivy.core.window.WindowBase method), 342, 380  
 scroll (kivy.effects.scroll.ScrollEffect attribute), 386, 393  
 scroll\_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 715, 875  
 scroll\_distance (kivy.uix.carousel.Carousel attribute), 725, 889  
 scroll\_distance (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_timeout (kivy.uix.carousel.Carousel attribute), 725, 889  
 scroll\_timeout (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_type (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_wheel\_distance (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_x (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_x (kivy.uix.textinput.TextInput attribute), 823, 1031  
 scroll\_y (kivy.uix.scrollview.ScrollView attribute), 792, 988  
 scroll\_y (kivy.uix.textinput.TextInput attribute), 824, 1031  
 ScrollEffect (class in kivy.effects.scroll), 386, 393  
 scrolling (kivy.uix.listview.ListView attribute), 769, 947  
 ScrollView (class in kivy.uix.scrollview), 790, 986  
 section (kivy.uix.settings.SettingItem attribute), 797, 995  
 seek() (kivy.core.audio.Sound method), 324, 346  
 seek() (kivy.core.video.VideoBase method), 337, 373  
 seek() (kivy.uix.video.Video method), 836, 1044  
 seek() (kivy.uix.videoplayer.VideoPlayer method), 840, 1048  
 segment\_intersection() (kivy.vector.Vector static method), 248, 1074  
 segments (kivy.graphics.Bezier attribute), 412  
 segments (kivy.graphics.Ellipse attribute), 416  
 segments (kivy.graphics.vertex\_instructions.Bezier attribute), 471  
 segments (kivy.graphics.vertex\_instructions.Ellipse attribute), 465  
 select() (kivy.uix.dropdown.DropDown method), 731, 899  
 select() (kivy.uix.listview.SelectableView method), 768, 946  
 select\_all() (kivy.uix.textinput.TextInput method), 824, 1031  
 select\_language() (kivy.core.spelling.SpellingBase method), 329, 357  
 select\_list() (kivy.adapters.listadapter.ListAdapter method), 255, 266  
 select\_node() (kivy.uix.behaviors.CompoundSelectionBehavior method), 715, 875  
 select\_node() (kivy.uix.treeview.TreeView method), 829, 1040  
 select\_text() (kivy.uix.textinput.TextInput method), 824, 1032  
 select\_with\_key\_down() (kivy.uix.behaviors.CompoundSelectionBehavior method), 715, 875  
 select\_with\_key\_up() (kivy.uix.behaviors.CompoundSelectionBehavior method), 715, 875  
 select\_with\_touch() (kivy.uix.behaviors.CompoundSelectionBehavior method), 715, 876  
 SelectableDataItem (class in kivy.adapters.models), 256, 269  
 SelectableView (class in kivy.uix.listview), 767, 945  
 selected\_alpha (kivy.uix.settings.SettingItem attribute), 797, 995  
 selected\_color (kivy.uix.listview.CompositeListItem attribute), 768, 946  
 selected\_color (kivy.uix.listview.ListItemButton attribute), 768, 946  
 selected\_node (kivy.uix.treeview.TreeView at-

tribute), 829, 1040  
 selected\_nodes (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 716, 876  
 selection (kivy.adapters.listadapter.ListAdapter attribute), 255, 266  
 selection (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 selection\_color (kivy.uix.textinput.TextInput attribute), 824, 1032  
 selection\_from (kivy.uix.textinput.TextInput attribute), 824, 1032  
 selection\_limit (kivy.adapters.listadapter.ListAdapter attribute), 255, 267  
 selection\_mode (kivy.adapters.listadapter.ListAdapter attribute), 255, 267  
 selection\_text (kivy.uix.textinput.TextInput attribute), 824, 1032  
 selection\_to (kivy.uix.textinput.TextInput attribute), 824, 1032  
 separator\_color (kivy.uix.popup.Popup attribute), 773, 956  
 separator\_height (kivy.uix.popup.Popup attribute), 774, 956  
 separator\_image (kivy.uix.actionbar.ActionGroup attribute), 706, 865  
 separator\_width (kivy.uix.actionbar.ActionGroup attribute), 706, 865  
 set() (kivy.config.ConfigParser method), 196, 319  
 set() (kivy.graphics.context\_instructions.Rotate method), 436, 479  
 set() (kivy.graphics.Rotate method), 425  
 set() (kivy.properties.Property method), 236, 675  
 set\_config() (kivy.properties.ConfigParserProperty method), 242, 681  
 set\_fbo\_shader() (kivy.uix.effectwidget.EffectBase method), 733, 903  
 set\_icon() (kivy.core.window.WindowBase method), 342, 380  
 set\_max() (kivy.properties.BoundedNumericProperty method), 239, 677  
 set\_min() (kivy.properties.BoundedNumericProperty method), 239, 677  
 set\_title() (kivy.core.window.WindowBase method), 342, 380  
 set\_vkeyboard\_class() (kivy.core.window.WindowBase method), 343, 380  
 set\_window() (kivy.base.EventLoopBase method), 208, 302  
 setall() (kivy.config.ConfigParser method), 196, 319  
 setdefault() (kivy.config.ConfigParser method), 196, 319  
 setdefaults() (kivy.config.ConfigParser method), 197, 319  
 setter() (kivy.event.EventDispatcher method), 200, 368  
 SettingBoolean (class in kivy.uix.settings), 798, 996  
 SettingItem (class in kivy.uix.settings), 796, 994  
 SettingNumeric (class in kivy.uix.settings), 798, 996  
 SettingOptions (class in kivy.uix.settings), 798, 996  
 SettingPath (class in kivy.uix.settings), 797, 995  
 Settings (class in kivy.uix.settings), 795, 993  
 settings (kivy.uix.settings.SettingsPanel attribute), 796, 994  
 settings\_cls (kivy.app.App attribute), 180, 295  
 SettingsPanel (class in kivy.uix.settings), 796, 994  
 SettingString (class in kivy.uix.settings), 797, 995  
 SettingsWithNoMenu (class in kivy.uix.settings), 799, 997  
 SettingsWithSidebar (class in kivy.uix.settings), 798, 996  
 SettingsWithSpinner (class in kivy.uix.settings), 799, 996  
 SettingsWithTabbedPanel (class in kivy.uix.settings), 799, 997  
 SettingTitle (class in kivy.uix.settings), 798, 996  
 setup\_mode() (kivy.uix.vkeyboard.VKeyboard method), 834, 1055  
 setup\_mode\_dock() (kivy.uix.vkeyboard.VKeyboard method), 834, 1055  
 setup\_mode\_free() (kivy.uix.vkeyboard.VKeyboard method), 834, 1055  
 Shader (class in kivy.graphics.shader), 453, 506  
 shader (kivy.graphics.instructions.RenderContext attribute), 432, 490  
 shader (kivy.graphics.RenderContext attribute), 424  
 ShaderTransition (class in kivy.uix.screenmanager), 787, 982  
 Shape (class in kivy.input.shape), 536, 597  
 shape (kivy.input.motionevent.MotionEvent attribute), 534, 542  
 ShapeRect (class in kivy.input.shape), 536, 597  
 shorten (kivy.uix.label.Label attribute), 756, 932  
 shorten() (kivy.core.text.LabelBase method), 332, 361  
 shorten\_from (kivy.uix.label.Label attribute), 756, 932  
 shorten\_post() (kivy.core.text.markup.MarkupLabel method), 333, 362, 368  
 show\_arrow (kivy.uix.bubble.Bubble attribute), 720, 881  
 show\_errors (kivy.uix.rst.RstDocument attribute), 852, 967  
 show\_hidden (kivy.uix.filechooser.FileChooserController attribute), 739, 909  
 show\_keyboard() (kivy.uix.textinput.TextInput method), 824, 1032

SimpleListAdapter (class in kivy.adapters.simplelistadapter), 257, 271

single\_points\_test() (kivy.uix.gesturesurface.GestureContainer method), 744, 915

sizable\_from (kivy.uix.splitter.Splitter attribute), 805, 1006

size (kivy.core.image.Image attribute), 328, 355

size (kivy.core.image.ImageData attribute), 329, 356

size (kivy.core.window.WindowBase attribute), 343, 380

size (kivy.graphics.Fbo attribute), 417

size (kivy.graphics.fbo.Fbo attribute), 440, 483

size (kivy.graphics.Rectangle attribute), 424

size (kivy.graphics.texture.Texture attribute), 461, 514

size (kivy.graphics.vertex\_instructions.Rectangle attribute), 465

size (kivy.uix.effectwidget.HorizontalBlurEffect attribute), 734, 904

size (kivy.uix.effectwidget.VerticalBlurEffect attribute), 735, 904

size (kivy.uix.widget.Widget attribute), 848, 1063

size\_hint (kivy.uix.widget.Widget attribute), 848, 1063

size\_hint\_x (kivy.uix.widget.Widget attribute), 848, 1064

size\_hint\_y (kivy.uix.widget.Widget attribute), 848, 1064

Slider (class in kivy.uix.slider), 801, 999

slides (kivy.uix.carousel.Carousel attribute), 725, 889

SlideTransition (class in kivy.uix.screenmanager), 788, 983

SmoothLine (class in kivy.graphics.vertex\_instructions), 471

softinput\_mode (kivy.core.window.WindowBase attribute), 343, 380

sort\_func (kivy.uix.filechooser.FileChooserController attribute), 739, 909

sorted\_keys (kivy.adapters.dictadapter.DictAdapter attribute), 253, 263

Sound (class in kivy.core.audio), 323, 345

SoundLoader (class in kivy.core.audio), 324, 346

source (kivy.core.audio.Sound attribute), 324, 346

source (kivy.core.image.ImageData attribute), 329, 356

source (kivy.graphics.BindTexture attribute), 413

source (kivy.graphics.context\_instructions.BindTexture attribute), 435, 478

source (kivy.graphics.instructions.VertexInstruction attribute), 430, 488

source (kivy.graphics.shader.Shader attribute), 453, 506

source (kivy.graphics.VertexInstruction attribute), 427

source (kivy.uix.effectwidget.EffectBase attribute), 733, 903

source (kivy.uix.image.Image attribute), 750, 925

source (kivy.uix.rst.RstDocument attribute), 853, 967

source (kivy.uix.videoplayer.VideoPlayer attribute), 840, 1049

source\_encoding (kivy.uix.rst.RstDocument attribute), 853, 967

source\_error (kivy.uix.rst.RstDocument attribute), 853, 967

source\_stack (kivy.graphics.ApplyContextMatrix attribute), 429

sp() (in module kivy.metrics), 224, 629

spacing (kivy.uix.boxlayout.BoxLayout attribute), 717, 878

spacing (kivy.uix.gridlayout.GridLayout attribute), 748, 920

spacing (kivy.uix.stacklayout.StackLayout attribute), 807, 1009

SpellingBase (class in kivy.core.spelling), 329, 357

Spinner (class in kivy.uix.spinner), 803, 1004

SpinnerOption (class in kivy.uix.spinner), 803, 1004

split\_str (kivy.uix.label.Label attribute), 756, 932

Splitter (class in kivy.uix.splitter), 804, 1005

spos (kivy.input.MotionEvent attribute), 520

spos (kivy.input.motionevent.MotionEvent attribute), 534, 542

spring\_constant (kivy.effects.dampedscroll.DampedScrollEffect attribute), 383, 387

stack (kivy.graphics.context\_instructions.MatrixInstruction attribute), 437, 480

stack (kivy.graphics.context\_instructions.PopMatrix attribute), 436, 478

stack (kivy.graphics.context\_instructions.PushMatrix attribute), 435, 478

stack (kivy.graphics.LoadIdentity attribute), 429

stack (kivy.graphics.MatrixInstruction attribute), 422

stack (kivy.graphics.PopMatrix attribute), 424

stack (kivy.graphics.PushMatrix attribute), 424

StackLayout (class in kivy.uix.stacklayout), 806, 1008

start (kivy.uix.videoplayer.VideoPlayerAnnotation attribute), 841, 1049

start() (kivy.animation.Animation method), 162, 275

start() (kivy.base.EventLoopBase method), 208, 302

start() (kivy.core.camera.CameraBase method), 325, 347

start() (kivy.effects.kinetic.KineticEffect method),



[385, 390](#)  
 start() (kivy.input.MotionEventProvider method), [520](#)  
 start() (kivy.input.provider.MotionEventProvider method), [536, 561](#)  
 start() (kivy.input.providers.tuio.TuioMotionEventProvider method), [528, 568, 586](#)  
 start() (kivy.loader.LoaderBase method), [183, 623](#)  
 start() (kivy.uix.screenmanager.TransitionBase method), [787, 982](#)  
 state (kivy.core.audio.Sound attribute), [324, 346](#)  
 state (kivy.core.video.VideoBase attribute), [337, 373](#)  
 state (kivy.uix.behaviors.ButtonBehavior attribute), [709, 869](#)  
 state (kivy.uix.video.Video attribute), [836, 1044](#)  
 state (kivy.uix.videoplayer.VideoPlayer attribute), [840, 1049](#)  
 status (kivy.core.audio.Sound attribute), [324, 346](#)  
 StencilPop (class in kivy.graphics), [426](#)  
 StencilPop (class in kivy.graphics.stencil\_instructions), [455, 508](#)  
 StencilPush (class in kivy.graphics), [426](#)  
 StencilPush (class in kivy.graphics.stencil\_instructions), [455, 508](#)  
 StencilUnUse (class in kivy.graphics), [426](#)  
 StencilUnUse (class in kivy.graphics.stencil\_instructions), [455, 508](#)  
 StencilUse (class in kivy.graphics), [426](#)  
 StencilUse (class in kivy.graphics.stencil\_instructions), [455, 508](#)  
 StencilView (class in kivy.uix.stencilview), [808, 1011](#)  
 step (kivy.uix.slider.Slider attribute), [801, 1000](#)  
 stop() (in module kivy.modules.inspector), [633, 637](#)  
 stop() (kivy.animation.Animation method), [162, 275](#)  
 stop() (kivy.app.App method), [180, 295](#)  
 stop() (kivy.base.EventLoopBase method), [208, 302](#)  
 stop() (kivy.core.audio.Sound method), [324, 346](#)  
 stop() (kivy.core.camera.CameraBase method), [325, 347](#)  
 stop() (kivy.core.video.VideoBase method), [337, 374](#)  
 stop() (kivy.effects.kinetic.KineticEffect method), [385, 390](#)  
 stop() (kivy.input.MotionEventProvider method), [520](#)  
 stop() (kivy.input.provider.MotionEventProvider method), [536, 561](#)  
 stop() (kivy.input.providers.tuio.TuioMotionEventProvider method), [528, 568, 586](#)  
 stop() (kivy.loader.LoaderBase method), [183, 623](#)  
 stop() (kivy.multistroke.ProgressTracker method), [229, 656](#)  
 stop() (kivy.uix.screenmanager.TransitionBase method), [787, 982](#)  
 stop\_all() (kivy.animation.Animation static method), [162, 275](#)  
 stop\_property() (kivy.animation.Animation method), [162, 275](#)  
 stopTouchApp() (in module kivy.base), [209, 303](#)  
 str\_to\_gesture() (kivy.gesture.GestureDatabase method), [204, 410](#)  
 string\_to\_keycode() (kivy.core.window.Keyboard method), [338, 376](#)  
 string\_types (in module kivy.compat), [192, 313](#)  
 StringProperty (class in kivy.properties), [237, 675](#)  
 strip (kivy.uix.label.Label attribute), [757, 932](#)  
 strip\_border (kivy.uix.tabbedpanel.TabbedPanel attribute), [813, 1019](#)  
 strip\_cls (kivy.uix.splitter.Splitter attribute), [805, 1006](#)  
 strip\_comments() (kivy.lang.Parser method), [221, 614](#)  
 strip\_image (kivy.uix.tabbedpanel.TabbedPanel attribute), [813, 1019](#)  
 strip\_size (kivy.uix.splitter.Splitter attribute), [805, 1006](#)  
 StripLayout (class in kivy.uix.tabbedpanel), [812, 1018](#)  
 stroke\_length() (kivy.gesture.GestureStroke method), [205, 410](#)  
 strtotuple() (in module kivy.utils), [244, 1067](#)  
 success (kivy.graphics.shader.Shader attribute), [454, 506](#)  
 suggest() (kivy.core.spelling.SpellingBase method), [330, 357](#)  
 SwapTransition (class in kivy.uix.screenmanager), [788, 983](#)  
 swipe\_threshold (kivy.uix.pagelayout.PageLayout attribute), [772, 953](#)  
 Switch (class in kivy.uix.switch), [808, 1013](#)  
 switch\_to() (kivy.uix.screenmanager.ScreenManager method), [786, 980](#)  
 switch\_to() (kivy.uix.tabbedpanel.TabbedPanel method), [814, 1019](#)  
 sx (kivy.input.motionevent.MotionEvent attribute), [534, 542](#)  
 sy (kivy.input.motionevent.MotionEvent attribute), [534, 542](#)  
 sync() (kivy.lang.BuilderBase method), [220, 614](#)  
 system\_size (kivy.core.window.WindowBase attribute), [343, 381](#)  
 sz (kivy.input.motionevent.MotionEvent attribute), [534, 542](#)

tribute), 534, 542

## T

- tab\_height (kivy.uix.tabbedpanel.TabbedPanel attribute), 814, 1019
- tab\_list (kivy.uix.tabbedpanel.TabbedPanel attribute), 814, 1020
- tab\_pos (kivy.uix.tabbedpanel.TabbedPanel attribute), 814, 1020
- tab\_width (kivy.uix.tabbedpanel.TabbedPanel attribute), 814, 1020
- tab\_width (kivy.uix.textinput.TextInput attribute), 824, 1032
- tabbed\_panel (kivy.uix.tabbedpanel.TabbedPanelStrip attribute), 815, 1020
- TabbedPanel (class in kivy.uix.tabbedpanel), 812, 1018
- TabbedPanelContent (class in kivy.uix.tabbedpanel), 814, 1020
- TabbedPanelException (class in kivy.uix.tabbedpanel), 815, 1020
- TabbedPanelHeader (class in kivy.uix.tabbedpanel), 814, 1020
- TabbedPanelItem (class in kivy.uix.tabbedpanel), 814, 1020
- TabbedPanelStrip (class in kivy.uix.tabbedpanel), 814, 1020
- target (kivy.core.window.Keyboard attribute), 338, 376
- target (kivy.graphics.texture.Texture attribute), 461, 514
- target (kivy.uix.vkeyboard.VKeyboard attribute), 835, 1055
- target\_stack (kivy.graphics.ApplyContextMatrix attribute), 429
- target\_widget (kivy.effects.scroll.ScrollEffect attribute), 386, 393
- template (kivy.adapters.adapter.Adapter attribute), 252, 260
- template() (kivy.lang.BuilderBase method), 220, 614
- tex\_coords (kivy.graphics.instructions.VertexInstruction attribute), 431, 488
- tex\_coords (kivy.graphics.texture.Texture attribute), 461, 514
- tex\_coords (kivy.graphics.VertexInstruction attribute), 427
- text (kivy.core.text.LabelBase attribute), 332, 361
- text (kivy.uix.label.Label attribute), 757, 933
- text (kivy.uix.rst.RstDocument attribute), 853, 967
- text (kivy.uix.textinput.TextInput attribute), 824, 1032
- text\_size (kivy.core.text.LabelBase attribute), 332, 361
- text\_size (kivy.uix.label.Label attribute), 757, 933
- TextInput (class in kivy.uix.textinput), 817, 1025
- textinput (kivy.uix.settings.SettingPath attribute), 798, 996
- textinput (kivy.uix.settings.SettingString attribute), 797, 995
- Texture (class in kivy.graphics.texture), 458, 512
- texture (kivy.core.camera.CameraBase attribute), 325, 347
- texture (kivy.core.image.Image attribute), 328, 355
- texture (kivy.core.video.VideoBase attribute), 337, 374
- texture (kivy.graphics.Fbo attribute), 418
- texture (kivy.graphics.fbo.Fbo attribute), 440, 483
- texture (kivy.graphics.instructions.VertexInstruction attribute), 431, 488
- texture (kivy.graphics.VertexInstruction attribute), 428
- texture (kivy.uix.effectwidget.EffectWidget attribute), 733, 903
- texture (kivy.uix.image.Image attribute), 750, 925
- texture (kivy.uix.label.Label attribute), 757, 933
- texture\_size (kivy.uix.image.Image attribute), 750, 925
- texture\_size (kivy.uix.label.Label attribute), 758, 933
- texture\_update() (kivy.uix.label.Label method), 758, 934
- TextureRegion (class in kivy.graphics.texture), 461, 514
- textures (kivy.atlas.Atlas attribute), 186, 299
- thumbnail (kivy.uix.videoplayer.VideoPlayer attribute), 841, 1049
- tick() (kivy.clock.ClockBase method), 191, 311
- tick\_draw() (kivy.clock.ClockBase method), 191, 311
- time\_end (kivy.input.motionevent.MotionEvent attribute), 534, 542
- time\_start (kivy.input.motionevent.MotionEvent attribute), 535, 543
- time\_update (kivy.input.motionevent.MotionEvent attribute), 535, 543
- title (kivy.app.App attribute), 180, 295
- title (kivy.uix.accordion.AccordionItem attribute), 703, 860
- title (kivy.uix.actionbar.ActionPrevious attribute), 707, 866
- title (kivy.uix.popup.Popup attribute), 774, 957
- title (kivy.uix.rst.RstDocument attribute), 853, 967
- title (kivy.uix.settings.SettingItem attribute), 797, 995
- title (kivy.uix.settings.SettingsPanel attribute), 796, 994
- title\_args (kivy.uix.accordion.AccordionItem attribute), 703, 860
- title\_color (kivy.uix.popup.Popup attribute), 774,

957  
 title\_size (kivy.uix.popup.Popup attribute), 774, 957  
 title\_template (kivy.uix.accordion.AccordionItem attribute), 703, 860  
 to\_local() (kivy.uix.widget.Widget method), 848, 1064  
 to\_parent() (kivy.uix.widget.Widget method), 848, 1064  
 to\_widget() (kivy.uix.widget.Widget method), 849, 1064  
 to\_window() (kivy.uix.widget.Widget method), 849, 1064  
 toctrees (kivy.uix.rst.RstDocument attribute), 853, 967  
 toggle\_fullscreen() (kivy.core.window.WindowBase method), 343, 381  
 toggle\_node() (kivy.uix.treeview.TreeView method), 829, 1040  
 ToggleButton (class in kivy.uix.togglebutton), 825, 1035  
 ToggleButtonBehavior (class in kivy.uix.behaviors), 709, 869  
 top (kivy.uix.widget.Widget attribute), 849, 1064  
 total (kivy.uix.filechooser.FileChooserProgressBase attribute), 740, 910  
 touch\_control (kivy.uix.switch.Switch attribute), 809, 1014  
 touch\_distance (kivy.uix.switch.Switch attribute), 809, 1014  
 touch\_multiselect (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 535, 543  
 touches (kivy.base.EventLoopBase attribute), 209, 302  
 transfer\_gesture() (kivy.multistroke.Recognizer method), 228, 655  
 transform (kivy.uix.scatter.Scatter attribute), 781, 974  
 transform\_inv (kivy.uix.scatter.Scatter attribute), 781, 974  
 transition (kivy.animation.Animation attribute), 162, 275  
 transition (kivy.uix.screenmanager.ScreenManager attribute), 786, 981  
 transition\_progress (kivy.uix.screenmanager.Screen attribute), 785, 979  
 transition\_state (kivy.uix.screenmanager.Screen attribute), 785, 979  
 TransitionBase (class in kivy.uix.screenmanager), 786, 981  
 Translate (class in kivy.graphics), 426  
 Translate (class in kivy.graphics.context\_instructions), 437, 480  
 translate() (kivy.graphics.transformation.Matrix method), 463, 516  
 translation\_touches (kivy.uix.scatter.Scatter attribute), 781, 974  
 transpose() (kivy.graphics.transformation.Matrix method), 463, 517  
 TreeView (class in kivy.uix.treeview), 827, 1038  
 TreeViewException (class in kivy.uix.treeview), 829, 1040  
 TreeViewLabel (class in kivy.uix.treeview), 829, 1040  
 TreeViewNode (class in kivy.uix.treeview), 829, 1041  
 Triangle (class in kivy.graphics), 427  
 Triangle (class in kivy.graphics.vertex\_instructions), 464  
 trigger\_action() (kivy.uix.behaviors.ButtonBehavior method), 709, 869  
 trim\_left\_of\_sel() (kivy.adapters.dictadapter.DictAdapter method), 253, 263  
 trim\_left\_of\_sel() (kivy.adapters.listadapter.ListAdapter method), 255, 267  
 trim\_right\_of\_sel() (kivy.adapters.dictadapter.DictAdapter method), 253, 263  
 trim\_right\_of\_sel() (kivy.adapters.listadapter.ListAdapter method), 256, 267  
 trim\_to\_sel() (kivy.adapters.dictadapter.DictAdapter method), 253, 263  
 trim\_to\_sel() (kivy.adapters.listadapter.ListAdapter method), 256, 267  
 triple\_tap\_time (kivy.input.motionevent.MotionEvent attribute), 535, 543  
 TuiO2dCurMotionEvent (class in kivy.input.providers.tuio), 529, 568, 586  
 TuiO2dObjMotionEvent (class in kivy.input.providers.tuio), 529, 568, 586  
 TuiOMotionEventProvider (class in kivy.input.providers.tuio), 528, 567, 585  
**U**  
 ud (kivy.input.motionevent.MotionEvent attribute), 535, 543  
 uid (kivy.input.motionevent.MotionEvent attribute), 535, 543  
 unbind() (kivy.event.EventDispatcher method), 201, 398  
 unbind() (kivy.properties.Property method), 236, 675  
 unbind\_widget() (kivy.lang.BuilderBase method), 220, 614  
 underline\_color (kivy.uix.rst.RstDocument attribute), 853, 967  
 ungrab() (kivy.input.MotionEvent method), 520

ungrab() (kivy.input.motionEvent.MotionEvent method), 535, 543  
 uniforms (kivy.uix.effectwidget.AdvancedEffectBase attribute), 734, 903  
 uninstall\_twisted\_reactor() (in module kivy.support), 243, 697  
 UnistrokeTemplate (class in kivy.multistroke), 231, 658  
 unload() (kivy.core.audio.Sound method), 324, 346  
 unload() (kivy.core.video.VideoBase method), 337, 374  
 unload() (kivy.uix.video.Video method), 836, 1044  
 unload\_file() (kivy.lang.BuilderBase method), 220, 614  
 unregister() (kivy.input.providers.tuio.TuioMotionEventProvider static method), 528, 568, 586  
 unregister\_event\_types() (kivy.event.EventDispatcher method), 201, 398  
 unschedule() (kivy.clock.ClockBase method), 191, 311  
 unzip\_extensions() (in module kivy.ext), 402  
 up\_count (kivy.uix.behaviors.CompoundSelectionBehavior attribute), 716, 876  
 update() (kivy.effects.kinetic.KineticEffect method), 385, 390  
 update() (kivy.input.MotionEventProvider method), 520  
 update() (kivy.input.provider.MotionEventProvider method), 536, 561  
 update() (kivy.input.providers.tuio.TuioMotionEventProvider method), 528, 568, 586  
 update\_bbox() (kivy.uix.gesturesurface.GestureContainer method), 744, 915  
 update\_config() (kivy.config.ConfigParser method), 197, 319  
 update\_from\_scroll() (kivy.uix.scrollview.ScrollView method), 792, 989  
 update\_velocity() (kivy.effects.kinetic.KineticEffect method), 385, 390  
 UpdateNormalMatrix (class in kivy.graphics), 429  
 url (kivy.network.urlrequest.UrlRequest attribute), 663, 667  
 URLRequest (class in kivy.network.urlrequest), 662, 665  
 use\_bubble (kivy.uix.textinput.TextInput attribute), 825, 1032  
 use\_handles (kivy.uix.textinput.TextInput attribute), 825, 1033  
 use\_kivy\_settings (kivy.app.App attribute), 181, 295  
 use\_parent\_modelview (kivy.graphics.instructions.RenderContext attribute), 432, 490  
 use\_parent\_modelview (kivy.graphics.RenderContext attribute), 424  
 use\_parent\_projection (kivy.graphics.instructions.RenderContext attribute), 433, 490  
 use\_parent\_projection (kivy.graphics.RenderContext attribute), 425  
 use\_separator (kivy.uix.actionbar.ActionGroup attribute), 706, 865  
 use\_separator (kivy.uix.actionbar.ActionView attribute), 706, 865  
 user\_data\_dir (kivy.app.App attribute), 181, 295  
 use\_size (kivy.core.text.LabelBase attribute), 332, 361  
 uvpos (kivy.graphics.texture.Texture attribute), 461, 514  
 uvsize (kivy.graphics.texture.Texture attribute), 461, 514

## V

value (kivy.graphics.Color attribute), 416  
 v (kivy.graphics.context\_instructions.Color attribute), 435, 478  
 valign (kivy.uix.label.Label attribute), 758, 934  
 value (kivy.effects.kinetic.KineticEffect attribute), 385, 390  
 value (kivy.uix.progressbar.ProgressBar attribute), 775, 959  
 value (kivy.uix.settings.SettingItem attribute), 797, 995  
 value (kivy.uix.slider.Slider attribute), 801, 1000  
 value\_normalized (kivy.uix.progressbar.ProgressBar attribute), 775, 959  
 value\_normalized (kivy.uix.slider.Slider attribute), 801, 1000  
 value\_pos (kivy.uix.slider.Slider attribute), 802, 1000  
 values (kivy.uix.settings.SettingBoolean attribute), 798, 996  
 values (kivy.uix.spinner.Spinner attribute), 803, 1004  
 VariableListProperty (class in kivy.properties), 240, 679  
 vbar (kivy.uix.scrollview.ScrollView attribute), 793, 989  
 Vector (class in kivy.vector), 247, 1072  
 velocity (kivy.effects.kinetic.KineticEffect attribute), 385, 390  
 VertexInstruction (class in kivy.graphics), 427  
 VertexInstruction (class in kivy.graphics.instructions), 430, 488  
 VerticalBlurEffect (class in kivy.uix.effectwidget), 734, 904

vertices (kivy.graphics.Mesh attribute), 423  
 vertices (kivy.graphics.vertex\_instructions.Mesh attribute), 470  
 Video (class in kivy.uix.video), 835, 1043  
 VideoBase (class in kivy.core.video), 336, 373  
 VideoPlayer (class in kivy.uix.videoplayer), 838, 1047  
 VideoPlayerAnnotation (class in kivy.uix.videoplayer), 841, 1049  
 view\_clip() (kivy.graphics.transformation.Matrix method), 463, 517  
 viewport\_size (kivy.uix.scrollview.ScrollView attribute), 793, 989  
 VKeyboard (class in kivy.uix.vkeyboard), 832, 1053  
 volume (kivy.core.audio.Sound attribute), 324, 346  
 volume (kivy.core.video.VideoBase attribute), 337, 374  
 volume (kivy.uix.video.Video attribute), 836, 1044  
 volume (kivy.uix.videoplayer.VideoPlayer attribute), 841, 1049  
 vs (kivy.graphics.shader.Shader attribute), 454, 506  
 vs (kivy.uix.screenmanager.ShaderTransition attribute), 788, 982

## W

wait() (kivy.network.urlrequest.UrlRequest method), 663, 667  
 walk() (kivy.uix.widget.Widget method), 849, 1064  
 walk\_reverse() (kivy.uix.widget.Widget method), 849, 1065  
 WeakMethod (class in kivy.weakmethod), 249, 1075  
 wheel (kivy.uix.colorpicker.ColorPicker attribute), 728, 895  
 Widget (class in kivy.uix.widget), 843, 1059  
 widget (kivy.core.window.Keyboard attribute), 338, 376  
 WidgetException (class in kivy.uix.widget), 850, 1066  
 width (kivy.core.image.Image attribute), 328, 355  
 width (kivy.core.image.ImageData attribute), 329, 356  
 width (kivy.core.window.WindowBase attribute), 343, 381  
 width (kivy.graphics.Line attribute), 422  
 width (kivy.graphics.texture.Texture attribute), 461, 514  
 width (kivy.graphics.vertex\_instructions.Line attribute), 469  
 width (kivy.input.shape.ShapeRect attribute), 536, 597  
 width (kivy.uix.widget.Widget attribute), 850, 1066  
 Window (in module kivy.core.window), 343, 381  
 window (kivy.core.window.Keyboard attribute), 338, 376  
 window (kivy.input.recorder.Recorder attribute), 531, 595  
 WindowBase (class in kivy.core.window), 338, 376  
 WipeTransition (class in kivy.uix.screenmanager), 788, 983  
 with\_previous (kivy.uix.actionbar.ActionPrevious attribute), 707, 866  
 WM\_MotionEvent (class in kivy.input.providers.wm\_touch), 527, 567, 591  
 WM\_Pen (class in kivy.input.providers.wm\_pen), 527, 566, 589  
 wrap (kivy.graphics.texture.Texture attribute), 461, 514  
 write() (kivy.config.ConfigParser method), 197, 319

## X

x (kivy.graphics.context\_instructions.Scale attribute), 436, 479  
 x (kivy.graphics.context\_instructions.Translate attribute), 437, 480  
 x (kivy.graphics.Scale attribute), 426  
 x (kivy.graphics.Translate attribute), 426  
 x (kivy.input.motionevent.MotionEvent attribute), 535, 543  
 x (kivy.uix.widget.Widget attribute), 850, 1066  
 x (kivy.vector.Vector attribute), 249, 1074  
 xy (kivy.graphics.context\_instructions.Translate attribute), 437, 480  
 xy (kivy.graphics.Translate attribute), 426  
 xyz (kivy.graphics.context\_instructions.Scale attribute), 436, 479  
 xyz (kivy.graphics.context\_instructions.Translate attribute), 437, 480  
 xyz (kivy.graphics.Scale attribute), 426  
 xyz (kivy.graphics.Translate attribute), 427

## Y

y (kivy.graphics.context\_instructions.Scale attribute), 437, 479  
 y (kivy.graphics.context\_instructions.Translate attribute), 437, 480  
 y (kivy.graphics.Scale attribute), 426  
 y (kivy.graphics.Translate attribute), 427  
 y (kivy.input.motionevent.MotionEvent attribute), 535, 543  
 y (kivy.uix.widget.Widget attribute), 850, 1066  
 y (kivy.vector.Vector attribute), 249, 1074

## Z

z (kivy.graphics.context\_instructions.Scale attribute), 437, 480

z (kivy.graphics.context\_instructions.Translate attribute), [437](#), [480](#)  
z (kivy.graphics.Scale attribute), [426](#)  
z (kivy.graphics.Translate attribute), [427](#)  
z (kivy.input.motionevent.MotionEvent attribute), [535](#), [543](#)