Vanilla Documentation

Release 0.1

Tal Leming

Sep 15, 2023

Contents

1 Concepts 1

2 Objects 9

3 Indices and tables 153

Python Module Index 155

Index 157

CHAPTER 1

Concepts

1.1 Positioning

Positioning objects on screen is the whole point of vanilla. This document explains the models, syntax and more. First, some terminology:

view Almost every object in vanilla represents a "view." A view is something that appears on screen: a button, a text editor, an image container, a slider, etc.

superview Every view is positioned within a "superview." If the superview moves, the view moves.

subview A view added to another view is a "subview."

window Windows contain a special superview that is positioned edge to edge of the non-title bar area of the window.

There are two ways to position views in vanilla:

Specifying numerical coordinates. This model, named "frame layout", allows you to specify the x and y coordinates and the width and height of views. These values manually adust the frame of the view within the superview. This model offers the most control, but can be cumbersome and the precision makes it difficult to revise complex interface layouts.

Specifying relative positions. This model, known as "auto layout", allows you to describe where controls should be positioned relative to the superview and other views. This model is more verbose, but it is (typically) faster to implement both simple and complex interafces and it makes revisions significantly easier in complex interface layouts.

These models can be mixed and matched within an interface and even within a single superview.

The model that you want to use for positioning a view is indicated with the first argument, named `posSize`, when constructing the view. To indicate that you want to use frame layout, you give a tuple of numbers. To indicate that you want to use auto layout, you use the string `"auto"` and provide rules in a method of the view object.

```
w.button1((10, 10, 100, 20), "Frame Layout")
w.button2("auto", "Auto Layout")
```

1.1.1 Frame Layout

Frame layout is specified with a tuple of four numbers:

1. x position within the superview
2. y position within the superview
3. width
4. height

The `(0, 0)` coordinate is the top left corner of the superview.

Positions relative to the bottom or right of the superview are indicated with negative numbers or zero for the x, y, width and/or height. For example, `(-100, 20, 0, 20)` indicates that the x position is 100 units from the right and the width should align with the right.

Examples

Basic window:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.editor = vanilla.TextEditor((15, 15, -15, -43), "Hello World!")
w.button = vanilla.Button((15, -35, -15, 20), "Done")
w.open()
```

1.1.2 Auto Layout

Auto layout is specified with a series of rules that are added after all subviews have been added to the superview. Auto layout is incredibly powerful, but incredibly complex. The information below is written to cover the interface layouts that are typically created with vanilla. For complete information, read Apple's Auto Layout Guide and documentation on NSLayoutConstraint.

Rules

Auto layout rules specify the relation of one view to another. These rules are added with the superview's `addAutoPosSizeRules` method after the views themselves have been added to the superview.

```
w = vanilla.Window((200, 200))
w.button1 = vanilla.Button("auto", "Button 1")
w.button2 = vanilla.Button("auto", "Button 2")
rules = [
    # see below for documentation
]
w.addAutoPosSizeRules(rules)
```

Rules can be defined as strings formatted in Apple's Visual Format Language or they can be defined as dictionaries with key/value pairs defining view relationships. The dictionary rule form is for advanced edge cases and is defined in detail in the `addAutoPosSizeRules` method documentation. Most interface can be specified with the string rule form, so that form is used for all of the examples that follow.

An optional `metrics` dictionary can be passed to the `addAutoPosSizeRules` method. These allow the string rules to reference constant values, such as spacing values or control sizes, by name. Examples of this are below.

String Rules

Orientations

`H:`	A horizontal rule.
`V:`	A vertical rule.

View References

`|`	Edge of the superview.
`[name]`	An attribute name you used to assign the view to the superview.

Relations

==	Equal.
>=	Greater than or equal.
<=	Less than or equal.

Metrics

-	Standard space.
number (int or float)	A specific number of points.
metric name (string)	A metric defined in the metrics dictionary.

Examples

The following examples use this code, replacing the `rules` and `metrics` as indicated.

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.button = vanilla.Button("auto", "Hello")
rules = []
metrics = {}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

This code will add a button to a window, but it doesn't say anything about where the button should be placed or how big it should be.

Place the button with no space around it:

```
rules = [
    "|[button]|"
]
```

Place the button with standard space around it:

```
rules = [
    "|-[button]-|"
]
```

Place the button with specific space around it:

```
rules = [
    "|-50-[button]-20-|"
]
```

Place the button with a metric defined space around it:

```
rules = [
    "|-padding-[button]-padding-|"
]
metrics = {
    "padding" : 33
}
```

In each of these, the width of the button has been flexible. Define a specific width:

```
rules = [
    "|-[button(75)]-|"
]
```

Define a minimum width:

```
rules = [
    "|-[button(>=75)]-|"
]
```

Define a maximum width:

```
rules = [
    "|-[button(<=100)]-|"
]
```

Define minimum and maximum widths:

```
rules = [
    "|-[button(>=75,<=200)]-|"
]
```

The previous examples all specified horizontal rules. To indicate the direction of a rule, start the rule with `H:` for horizontal and `V:` for vertical. If an orientation isn't specified, as in the examples above, the orientation will be horizontal.

```
rules = [
    # Horizontal
    "H:|-[button]-|",
    # Vertical
    "V:|-[button]-|"
]
```

All of the options shown for specifying values in horizontal orientation also work for specifying values in vertical orientation.

That covers the basics of placing one view in a superview. Placing multiple views uses the same syntax. The following examples use this code, replacing `rules` and `metrics` as indicated.

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.button1 = vanilla.Button("auto", "Hello")
w.button2 = vanilla.Button("auto", "World")
```

(continues on next page)

(continued from previous page)

```
rules = []
metrics = {}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Place the buttons next to each other:

```
rules = [
    # Horizontal
    "H:|-[button1]-[button2]-|",
    # Vertical
    "V:|-[button1]-|",
    "V:|-[button2]-|"
]
```

Place the buttons on top of each other:

```
rules = [
    # Horizontal
    "H:|-[button1]-|",
    "H:|-[button2]-|",
    # Vertical
    "V:|-[button1]-[button2]-|",
]
```

Views can be referenced by other views within rules. To make the buttons have the same width:

```
rules = [
    # Horizontal
    "H:|-[button1]-[button2(==button1)]-|",
    # Vertical
    "V:|-[button1]-|",
    "V:|-[button2]-|"
]
```

Basic window:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.editor = vanilla.TextEditor("auto", "Hello World!")
w.button = vanilla.Button("auto", "Done")
rules = [
    # Horizontal
    "H:|-border-[editor]-border-|",
    "H:|-border-[button]-border-|",
    # Vertical
    "V:|-border-[editor(>=100)]-space-[button]-border-|"
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Stack of views, all with the same width:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.button1 = vanilla.Button("auto", "Button 1")
w.button2 = vanilla.Button("auto", "Button 2")
w.editor = vanilla.TextEditor("auto", "Hello World!")
w.button3 = vanilla.Button("auto", "Button 3")
w.button4 = vanilla.Button("auto", "Button 4")
rules = [
    # Horizontal
    "H:|-border-[button1]-border-|",
    "H:|-border-[button2]-border-|",
    "H:|-border-[editor]-border-|",
    "H:|-border-[button3]-border-|",
    "H:|-border-[button4]-border-|",
    # Vertical
    "V:|-border-[button1]-space-[button2]-space-[editor(>=100)]-space-[button3]-space-
↪[button4]-border-|"
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Stack of views, with different widths:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.button1 = vanilla.Button("auto", "Button 1")
w.button2 = vanilla.Button("auto", "Button 2")
w.button3 = vanilla.Button("auto", "Button 3")
w.button4 = vanilla.Button("auto", "Button 4")
rules = [
    # Horizontal
    "H:|-border-[button1]-border-|",
    "H:|-border-[button2]-space-[button3(==button2)]-border-|",
    "H:|-border-[button4]-border-|",
    # Vertical
    "V:|-border-[button1]-space-[button2]-space-[button4]-border-|",
    "V:|-border-[button1]-space-[button3]-space-[button4]-border-|"
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Flexible views:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.editor1 = vanilla.TextEditor("auto", "Hello World!")
w.editor2 = vanilla.TextEditor("auto", "Hello World!")
w.editor3 = vanilla.TextEditor("auto", "Hello World!")
w.editor4 = vanilla.TextEditor("auto", "Hello World!")
rules = [
    # Horizontal
    "H:|-border-[editor1]-space-[editor2(==editor1)]-border-|",
```

(continues on next page)

(continued from previous page)

```
    "H:|-border-[editor3]-space-[editor4(==editor3)]-border-|",
    # Vertical
    "V:|-border-[editor1]-space-[editor3(==editor1)]-border-|",
    "V:|-border-[editor2]-space-[editor4(==editor2)]-border-|",
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Flexible spaces:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.flex1 = vanilla.Group("auto")
w.flex2 = vanilla.Group("auto")
w.flex3 = vanilla.Group("auto")
w.button1 = vanilla.Button("auto", "Button 1")
w.button2 = vanilla.Button("auto", "Button 2")
w.button3 = vanilla.Button("auto", "Button 3")
rules = [
    # Horizontal
    "H:|-[flex1(>=border)]-[button1]-[flex2(==flex1)]-|",
    "H:|-border-[button2(==100)]-[flex3(>=space)]-[button3(==button2)]-border-|",
    # Vertical
    "V:|-border-[button1]-space-[button2]-border-|",
    "V:|-border-[button1]-space-[button3]-border-|",
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(rules, metrics)
w.open()
```

Nested views:

```
w = vanilla.Window((200, 200), minSize=(100, 100))
w.editor1 = vanilla.TextEditor("auto", "Hello World!")
w.editor2 = vanilla.TextEditor("auto", "Hello World!")
w.nest = vanilla.Group("auto")
w.nest.editor = vanilla.TextEditor("auto", "Hello World!")
w.nest.button = vanilla.Button("auto", "Button")
windowRules = [
    # Horizontal
    "H:|-border-[editor1(>=100)]-space-[editor2(==editor1)]-space-[nest(==100)]-
↪border-|",
    # Vertical
    "V:|-border-[editor1]-border-|",
    "V:|-border-[editor2]-border-|",
    "V:|-border-[nest]-border-|"
]
nestRules = [
    # Horizontal
    "H:|[editor]|",
    "H:|[button]|",
```

(continues on next page)

(continued from previous page)

```
    # Vertical
    "V:|[editor]-space-[button]|"
]
metrics = {
    "border" : 15,
    "space" : 8
}
w.addAutoPosSizeRules(windowRules, metrics)
w.nest.addAutoPosSizeRules(nestRules, metrics)
w.open()
```

Table of views:

CHAPTER 2

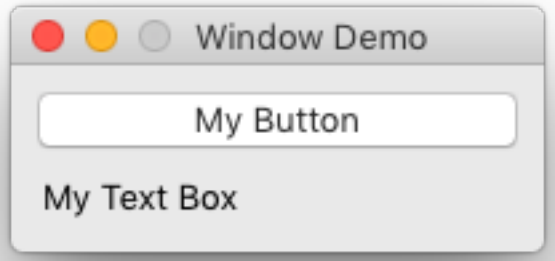
Objects

2.1 Windows

2.1.1 Window

class vanilla.**Window**(*posSize*, *title=''*, *minSize=None*, *maxSize=None*, *textured=False*, *autosaveName=None*, *closable=True*, *miniaturizable=True*, *initiallyVisible=True*, *fullScreenMode=None*, *titleVisible=True*, *fullSizeContentView=False*, *screen=None*)

A window capable of containing controls.



To add a control to a window, simply set it as an attribute of the window.

```
from vanilla import Window, Button, TextBox

class WindowDemo:

    def __init__(self):
        self.w = Window((200, 70), "Window Demo")
        self.w.myButton = Button((10, 10, -10, 20), "My Button")
        self.w.myTextBox = TextBox((10, 40, -10, 17), "My Text Box")
        self.w.open()

WindowDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(left, top, width, height)* representing the position and size of the window. It may also be a tuple of form *(width, height)*. In this case, the window will be positioned on screen automatically.

title The title to be set in the title bar of the window.

minSize Tuple of the form *(width, height)* representing the minimum size that the window can be resized to.

maxSize Tuple of the form *(width, height)* representing the maximum size that the window can be resized to.

textured Boolean value representing if the window should have a textured appearance or not.

autosaveName A string representing a unique name for the window. If given, this name will be used to store the window position and size in the application preferences.

closable Boolean value representing if the window should have a close button in the title bar.

miniaturizable Boolean value representing if the window should have a minimize button in the title bar.

initiallyVisible Boolean value representing if the window will be initially visible. Default is *True*. If *False*, you can show the window later by calling *window.show()*.

fullScreenMode An indication of the full screen mode. These are the options:

None	The window does not allow full screen.
"primary"	Corresponds to NSWindowCollectionBehaviorFullScreenPrimary.
"auxiliary"	Corresponds to NSWindowCollectionBehaviorFullScreenAuxiliary.

titleVisible Boolean value indicating if the window title should be displayed.

fullSizeContentView Boolean value indicating if the content view should be the full size of the window, including the area underneath the titlebar and toolbar.

screen A NSScreen object indicating the screen that the window should be drawn to. When None the window will be drawn to the main screen.

center()
Center the window within the screen.

close()
Close the window.

Once a window has been closed it can not be re-opened.

getNSWindow()
Return the NSWindow that this Vanilla object wraps.

getNSWindowController()
Return an NSWindowController for the NSWindow that this Vanilla object wraps, creating a one if needed.

getPosSize()
A tuple of form *(left, top, width, height)* representing the window's position and size.

getTitle()
The title in the window's title bar.

hide()
Hide the window.

isVisible()
A boolean value representing if the window is visible or not.

makeKey()
Make the window the key window.

makeMain()
Make the window the main window.

open()
Open the window.

select()
Select the window if it is not the currently selected window.

show()
Show the window if it is hidden.

Window.**assignToDocument**(*document*)
Add this window to the list of windows associated with a document.

document should be a *NSDocument* instance.

Window.**setTitle**(*title*)
Set the title in the window's title bar.

title should be a string.

Window.**setPosSize**(*posSize*, *animate=True*)
Set the position and size of the window.

posSize A tuple of form *(left, top, width, height)*.

Window.**move**(*x*, *y*, *animate=True*)
Move the window by *x* units and *y* units.

Window.**resize**(*width*, *height*, *animate=True*)
Change the size of the window to *width* and *height*.

Window.**setDefaultButton**(*button*)
Set the default button in the window.

button will be bound to the Return and Enter keys.

Window.**bind**(*event*, *callback*)
Bind a callback to an event.

event A string representing the desired event. The options are:

"should close"	Called when the user attempts to close the window. This must return a bool indicating if the window should be closed or not.
"close"	Called immediately before the window closes.
"move"	Called immediately after the window is moved.
"resize"	Called immediately after the window is resized.
"became main"	Called immediately after the window has become the main window.
"resigned main"	Called immediately after the window has lost its main window status.
"became key"	Called immediately after the window has become the key window.
"resigned key"	Called immediately after the window has lost its key window status.

For more information about main and key windows, refer to the Cocoa documentation on the subject.

callback The callback that will be called when the event occurs. It should accept a *sender* argument which will be the Window that called the callback.:

```
from vanilla import Window

class WindowBindDemo(object):

    def __init__(self):
        self.w = Window((200, 200))
        self.w.bind("move", self.windowMoved)
        self.w.open()

    def windowMoved(self, sender):
        print("window moved!", sender)

WindowBindDemo()
```

Window.**unbind**(*event*, *callback*)

Unbind a callback from an event.

event A string representing the desired event. Refer to *bind* for the options.

callback The callback that has been bound to the event.

Window.**addToolbar**(*toolbarIdentifier*, *toolbarItems*, *addStandardItems=True*)

Add a toolbar to the window.

toolbarIdentifier A string representing a unique name for the toolbar.

toolbarItems An ordered list of dictionaries containing the following items:

itemIdentifier	A unique string identifier for the item. This is only used internally.
label (optional)	The text label for the item. Defaults to *None*.
paletteLabel (optional)	The text label shown in the customization palette. Defaults to *label*.
toolTip (optional)	The tool tip for the item. Defaults to *label*.
imagePath (optional)	A file path to an image. Defaults to *None*.
imageNamed (optional)	The name of an image already loaded as a NSImage by the application. Defaults to *None*.
imageObject (optional)	A NSImage object. Defaults to *None*.
imageTemplate (optional)	A boolean representing if the image should converted to a template image.
selectable (optional)	A boolean representing if the item is selectable or not. The default value is *False*. For more information on selectable toolbar items, refer to Apple's documentation.
view (optional)	A *NSView* object to be used instead of an image. Defaults to *None*.
visibleByDefault (optional)	If the item should be visible by default pass *True* to this argument. If the item should be added to the toolbar only through the customization palette, use a value of *False*. Defaults to *True*.

addStandardItems A boolean, specifying whether the standard Cocoa toolbar items should be added. Defaults to *True*. If you set it to *False*, you must specify any standard items manually in *toolbarItems*, by using the constants from the AppKit module:

NSToolbarSeparatorItemIdentifier	The Separator item.
NSToolbarSpaceItemIdentifier	The Space item.
NSToolbarFlexibleSpaceItemIdentifier	The Flexible Space item.
NSToolbarShowColorsItemIdentifier	The Colors item. Shows the color panel.
NSToolbarShowFontsItemIdentifier	The Fonts item. Shows the font panel.
NSToolbarCustomizeToolbarItemIdentifier	The Customize item. Shows the customization palette.
NSToolbarPrintItemIdentifier	The Print item. Refer to Apple's NSToolbarItem documentation for more information.

displayMode A string representing the desired display mode for the toolbar.

"default"
"iconLabel"
"icon"
"label"

sizeStyle A string representing the desired size for the toolbar

"default"
"regular"
"small"

Returns a dictionary containing the created toolbar items, mapped by itemIdentifier.

`Window.`**`removeToolbarItem`**(*itemIdentifier*)
Remove a toolbar item by his identifier.

itemIdentifier A unique string identifier for the removed item.

2.1.2 FloatingWindow

class `vanilla.`**`FloatingWindow`**(*posSize*, *title=''*, *minSize=None*, *maxSize=None*, *textured=False*, *autosaveName=None*, *closable=True*, *initiallyVisible=True*, *screen=None*)
A window that floats above all other windows.

To add a control to a window, simply set it as an attribute of the window.

```
from vanilla import FloatingWindow, Button, TextBox

class FloatingWindowDemo:

    def __init__(self):
        self.w = FloatingWindow((200, 70), "FloatingWindow Demo")
        self.w.myButton = Button((10, 10, -10, 20), "My Button")
        self.w.myTextBox = TextBox((10, 40, -10, 17), "My Text Box")
        self.w.open()

FloatingWindowDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(left, top, width, height)* representing the position and size of the window. It may also be a tuple of form *(width, height)*. In this case, the window will be positioned on screen automatically.

title The title to be set in the title bar of the window.

minSize Tuple of the form *(width, height)* representing the minimum size that the window can be resized to.

maxSize Tuple of the form *(width, height)* representing the maximum size that the window can be resized to.

textured Boolean value representing if the window should have a textured appearance or not.

autosaveName A string representing a unique name for the window. If given, this name will be used to store the window position and size in the application preferences.

closable Boolean value representing if the window should have a close button in the title bar.

screen A NSScreen object indicating the screen that the window should be drawn to. When None the window will be drawn to the main screen.

center()
Center the window within the screen.

close()
Close the window.

Once a window has been closed it can not be re-opened.

getNSWindow()
Return the NSWindow that this Vanilla object wraps.

getNSWindowController()
Return an NSWindowController for the NSWindow that this Vanilla object wraps, creating a one if needed.

getPosSize()
A tuple of form *(left, top, width, height)* representing the window's position and size.

getTitle()
The title in the window's title bar.

hide()
Hide the window.

isVisible()
A boolean value representing if the window is visible or not.

makeKey()
Make the window the key window.

makeMain()
Make the window the main window.

open()
Open the window.

select()
Select the window if it is not the currently selected window.

show()
Show the window if it is hidden.

FloatingWindow.**assignToDocument**(*document*)
Add this window to the list of windows associated with a document.

document should be a *NSDocument* instance.

FloatingWindow.**setTitle**(*title*)
Set the title in the window's title bar.

title should be a string.

FloatingWindow.**setPosSize**(*posSize*, *animate=True*)
Set the position and size of the FloatingWindow.

posSize A tuple of form *(left, top, width, height)*.

FloatingWindow.**move**(*x*, *y*, *animate=True*)
Move the window by *x* units and *y* units.

FloatingWindow.**resize**(*width*, *height*, *animate=True*)
Change the size of the window to *width* and *height*.

FloatingWindow.**setDefaultButton**(*button*)
Set the default button in the FloatingWindow.

button will be bound to the Return and Enter keys.

FloatingWindow.**bind**(*event*, *callback*)
Bind a callback to an event.

event A string representing the desired event. The options are:

"should close"	Called when the user attempts to close the window. This must return a bool indicating if the window should be closed or not.
"close"	Called immediately before the window closes.
"move"	Called immediately after the window is moved.
"resize"	Called immediately after the window is resized.
"became main"	Called immediately after the window has become the main window.
"resigned main"	Called immediately after the window has lost its main window status.
"became key"	Called immediately after the window has become the key window.
"resigned key"	Called immediately after the window has lost its key window status.

For more information about main and key windows, refer to the Cocoa documentation on the subject.

callback The callback that will be called when the event occurs. It should accept a *sender* argument which will be the Window that called the callback.:

```
from vanilla import Window

class WindowBindDemo(object):

    def __init__(self):
        self.w = Window((200, 200))
        self.w.bind("move", self.windowMoved)
        self.w.open()

    def windowMoved(self, sender):
        print("window moved!", sender)

WindowBindDemo()
```

FloatingWindow.**unbind**(*event*, *callback*)
Unbind a callback from an event.

event A string representing the desired event. Refer to *bind* for the options.

callback The callback that has been bound to the event.

FloatingWindow.**addToolbar**(*toolbarIdentifier*, *toolbarItems*, *addStandardItems=True*)
Add a toolbar to the FloatingWindow.

toolbarIdentifier A string representing a unique name for the toolbar.

toolbarItems An ordered list of dictionaries containing the following items:

itemIdentifier	A unique string identifier for the item. This is only used internally.
label (optional)	The text label for the item. Defaults to *None*.
paletteLabel (optional)	The text label shown in the customization palette. Defaults to *label*.
toolTip (optional)	The tool tip for the item. Defaults to *label*.
imagePath (optional)	A file path to an image. Defaults to *None*.
imageNamed (optional)	The name of an image already loaded as a NSImage by the application. Defaults to *None*.
imageObject (optional)	A NSImage object. Defaults to *None*.
imageTemplate (optional)	A boolean representing if the image should converted to a template image.
selectable (optional)	A boolean representing if the item is selectable or not. The default value is *False*. For more information on selectable toolbar items, refer to Apple's documentation.
view (optional)	A *NSView* object to be used instead of an image. Defaults to *None*.
visibleByDefault (optional)	If the item should be visible by default pass *True* to this argument. If the item should be added to the toolbar only through the customization palette, use a value of *False*. Defaults to *True*.

addStandardItems A boolean, specifying whether the standard Cocoa toolbar items should be added. Defaults to *True*. If you set it to *False*, you must specify any standard items manually in *toolbarItems*, by using the constants from the AppKit module:

NSToolbarSeparatorItemIdentifier	The Separator item.
NSToolbarSpaceItemIdentifier	The Space item.
NSToolbarFlexibleSpaceItemIdentifier	The Flexible Space item.
NSToolbarShowColorsItemIdentifier	The Colors item. Shows the color panel.
NSToolbarShowFontsItemIdentifier	The Fonts item. Shows the font panel.
NSToolbarCustomizeToolbarItemIdentifier	The Customize item. Shows the customization palette.
NSToolbarPrintItemIdentifier	The Print item. Refer to Apple's NSToolbarItem documentation for more information.

displayMode A string representing the desired display mode for the toolbar.

"default"
"iconLabel"
"icon"
"label"

sizeStyle A string representing the desired size for the toolbar

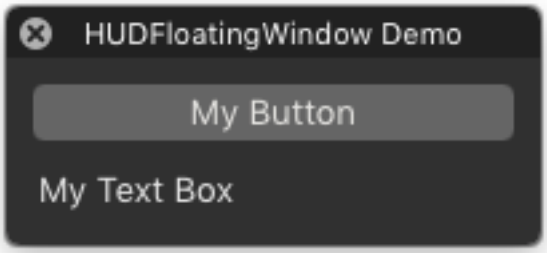
"default"
"regular"
"small"

Returns a dictionary containing the created toolbar items, mapped by itemIdentifier.

`FloatingWindow.`**`removeToolbarItem`**(*itemIdentifier*)
Remove a toolbar item by his identifier.

itemIdentifier A unique string identifier for the removed item.

class `vanilla.`**`HUDFloatingWindow`**(*posSize*, *title=''*, *minSize=None*, *maxSize=None*, *textured=False*, *autosaveName=None*, *closable=True*, *initiallyVisible=True*, *screen=None*)
A window that floats above all other windows and has the HUD appearance.



To add a control to a window, simply set it as an attribute of the window.

```
from vanilla import *

class HUDFloatingWindowDemo:

    def __init__(self):
        self.w = HUDFloatingWindow((200, 70), "HUDFloatingWindow Demo")
        self.w.myButton = Button((10, 10, -10, 20), "My Button")
        self.w.myTextBox = TextBox((10, 40, -10, 17), "My Text Box")
        self.w.open()

HUDFloatingWindowDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(left, top, width, height)* representing the position and size of the window. It may also be a tuple of form *(width, height)*. In this case, the window will be positioned on screen automatically.

title The title to be set in the title bar of the window.

minSize Tuple of the form *(width, height)* representing the minimum size that the window can be resized to.

maxSize Tuple of the form *(width, height)* representing the maximum size that the window can be resized to.

textured Boolean value representing if the window should have a textured appearance or not.

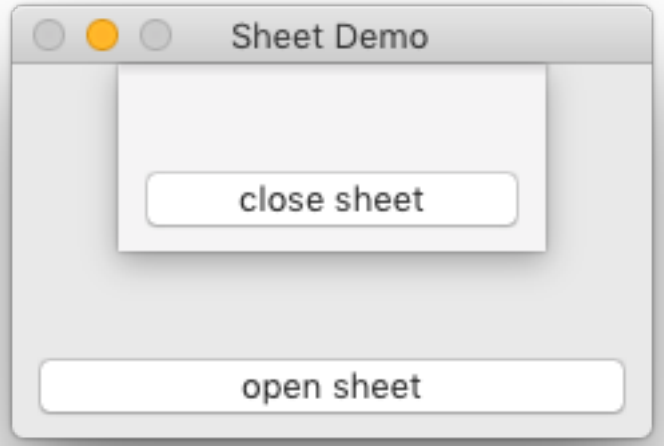
autosaveName A string representing a unique name for the window. If given, this name will be used to store the window position and size in the application preferences.

closable Boolean value representing if the window should have a close button in the title bar.

screen A NSScreen object indicating the screen that the window should be drawn to. When None the window will be drawn to the main screen.

2.1.3 Sheet

class vanilla.**Sheet**(*posSize*, *parentWindow*, *minSize=None*, *maxSize=None*, *autosaveName=None*)
A window that is attached to another window.



To add a control to a sheet, simply set it as an attribute of the sheet.:

```
from vanilla import Window, Sheet, Button

class SheetDemo:

    def __init__(self):
        self.w = Window((240, 140), "Sheet Demo")
        self.w.openSheet = Button((10, -30, -10, 20),
                    "open sheet", callback=self.openSheetCallback)
        self.w.open()

    def openSheetCallback(self, sender):
        self.sheet = Sheet((160, 70), self.w)
        self.sheet.closeSheet = Button((10, -30, -10, 20),
                    "close sheet", callback=self.closeSheetCallback)
        self.sheet.open()

    def closeSheetCallback(self, sender):
        self.sheet.close()
        del self.sheet

SheetDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(width, height)* representing the size of the sheet.

parentWindow The window that the sheet should be attached to.

minSize Tuple of the form *(width, height)* representing the minimum size that the sheet can be resized to.

maxSize Tuple of the form *(width, height)* representing the maximum size that the sheet can be resized to.

autosaveName A string representing a unique name for the sheet. If given, this name will be used to store the sheet size in the application preferences.

center()
Center the window within the screen.

close()
Close the window.

Once a window has been closed it can not be re-opened.

getNSWindow()
Return the NSWindow that this Vanilla object wraps.

getNSWindowController()
Return an NSWindowController for the NSWindow that this Vanilla object wraps, creating a one if needed.

getPosSize()
A tuple of form *(left, top, width, height)* representing the window's position and size.

getTitle()
The title in the window's title bar.

hide()
Hide the window.

isVisible()
A boolean value representing if the window is visible or not.

makeKey()
Make the window the key window.

makeMain()
Make the window the main window.

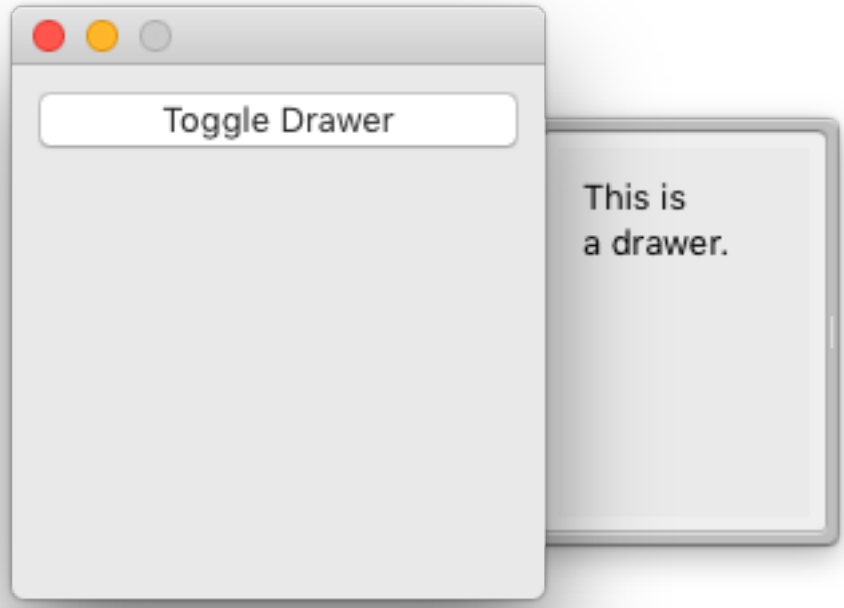
open()
Open the window.

select()
Select the window if it is not the currently selected window.

show()
Show the window if it is hidden.

2.1.4 Drawer

class vanilla.**Drawer**(*size*, *parentWindow*, *minSize=None*, *maxSize=None*, *preferredEdge='left'*, *forceEdge=False*, *leadingOffset=20*, *trailingOffset=20*)
A drawer attached to a window. Drawers are capable of containing controls.



Warning: Drawers are deprecated and should not be used in modern macOS apps.

To add a control to a drawer, simply set it as an attribute of the drawer.

```
from vanilla import Window, Button, Drawer, TextBox

class DrawerDemo:

    def __init__(self):
        self.w = Window((200, 200))
        self.w.button = Button((10, 10, -10, 20), "Toggle Drawer",
                               callback=self.toggleDrawer)
        self.d = Drawer((100, 150), self.w)
        self.d.textBox = TextBox((10, 10, -10, -10),
                               "This is a drawer.")
        self.w.open()
        self.d.open()

    def toggleDrawer(self, sender):
        self.d.toggle()

DrawerDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

size Tuple of form *(width, height)* representing the size of the drawer.

parentWindow The window that the drawer should be attached to.

minSize Tuple of form *(width, height)* representing the minimum size of the drawer.

maxSize Tuple of form *(width, height)* representing the maximum size of the drawer.

preferredEdge The preferred edge of the window that the drawer should be attached to. If the drawer cannot be opened on the preferred edge, it will be opened on the opposite edge. The options are:

"left"
"right"
"top"
"bottom"

forceEdge Boolean representing if the drawer should *always* be opened on the preferred edge.

leadingOffset Distance between the top or left edge of the drawer and the parent window.

trailingOffset Distance between the bottom or right edge of the drawer and the parent window.

`addAutoPosSizeRules`(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

close()
Close the drawer.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSDrawer()
Return the NSDrawer that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isOpen()
Return *True* if the drawer is open, *False* if it is closed.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

open()
Open the drawer.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

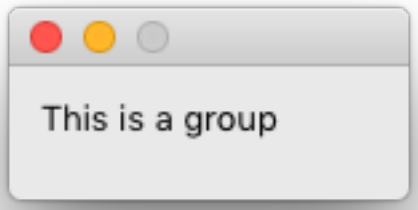
toggle()
Open the drawer if it is closed. Close it if it is open.

2.2 Layout Views

2.2.1 Group

class vanilla.**Group**(*posSize*, *blendingMode=None*, *dropSettings=None*)

An invisible container for controls.



To add a control to a group, simply set it as an attribute of the group.

```
from vanilla import Window, Group, TextBox

class GroupDemo:

    def __init__(self):
        self.w = Window((150, 50))
        self.w.group = Group((10, 10, -10, -10))
        self.w.group.text = TextBox((0, 0, -0, -0),
                                    "This is a group")
        self.w.open()

GroupDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the group.

blendingMode The blending mode for the window. These are the possible options:

None	No special blending.
"behindWindow"	Blend with the content behind the window.
"withinWindow"	Blend with the content within the window.

dropSettings A drop settings dictionary.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language

- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getDropItemValues(*items*, *pasteboardType=None*)
Get Python objects from the given NSPasteboardItem objects for the given pasteboard type. If this view is registered for only one pasteboard type, None may be given as the pasteboard type.

getNSView()
Return the NSView that this object wraps.

getNSVisualEffectView()
Return the NSVisualEffectView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.2 Tabs

class vanilla.**Tabs**(*posSize, titles=['Tab'], callback=None, sizeStyle='regular', showTabs=True, transitionStyle=None*)
A set of tabs attached to a window. Each tab is capable of containing controls.

To add a control to a tab, simply set it as an attribute of the tab.

```
from vanilla import Window, Tabs, TextBox
```

(continues on next page)

(continued from previous page)

```
class TabsDemo:

    def __init__(self):
        self.w = Window((250, 100))
        self.w.tabs = Tabs((10, 10, -10, -10), ["Tab One", "Tab Two"])
        tab1 = self.w.tabs[0]
        tab1.text = TextBox((10, 10, -10, -10), "This is tab 1")
        tab2 = self.w.tabs[1]
        tab2.text = TextBox((10, 10, -10, -10), "This is tab 2")
        self.w.open()

TabsDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

To retrieve a particular tab, access it by index:

```
myTab = self.w.tabs[0]
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the tabs.

titles An ordered list of tab titles.

callback The method to be called when the user selects a new tab.

sizeStyle A string representing the desired size style of the tabs. The options are:

"regular"
"small"
"mini"

showTabs Boolean representing if the tabview should display tabs.

transitionStyle A string rerpresenting a transition style between tabs. The options are:

None
"crossfade"
"slideUp"
"slideDown"
"slideLeft"
"slideRight"
"slideForward"
"slideBackward"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the index of the selected tab.

getNSTabView()
Return the NSTabView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the selected tab.

value The index of the tab to be selected.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

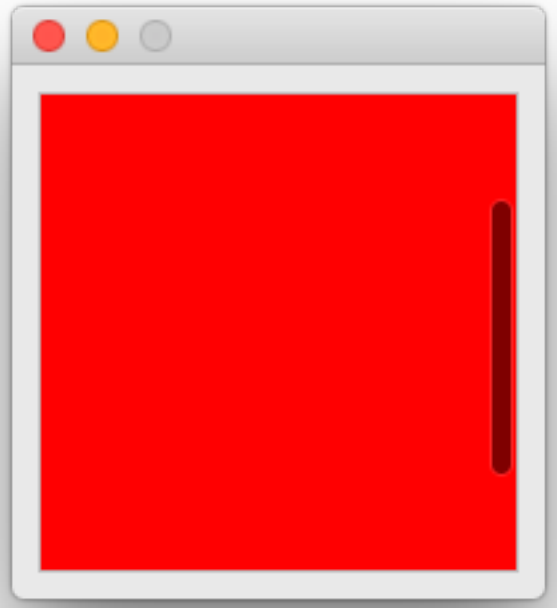
animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.3 ScrollView

class vanilla.**ScrollView**(*posSize*, *nsView*, *hasHorizontalScroller=True*, *hasVerticalScroller=True*, *autohidesScrollers=False*, *backgroundColor=None*, *clipView=None*, *drawsBackground=True*)
A view with scrollers for containing another view.



```
from AppKit import NSView, NSColor, NSRectFill
from vanilla import Window, ScrollView

class DemoView(NSView):

    def drawRect_(self, rect):
        NSColor.redColor().set()
        NSRectFill(self.bounds())

class ScrollViewDemo:

    def __init__(self):
        self.w = Window((200, 200))
        self.view = DemoView.alloc().init()
        self.view.setFrame_(((0, 0), (300, 300)))
        self.w.scrollView = ScrollView((10, 10, -10, -10),
                                       self.view)
        self.w.open()

ScrollViewDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the scroll view.

nsView A NSView object.

hasHorizontalScroller Boolean representing if the scroll view has horizontal scrollers.

hasVerticalScroller Boolean representing if the scroll view has vertical scrollers.

autohidesScrollers Boolean representing if the scroll view auto-hides its scrollers.

backgroundColor A NSColor object representing the background color of the scroll view.

drawsBackground Boolean representing if the background should be drawn.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSScrollView()
Return the NSScrollView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setBackgroundColor(*color*)
Set the background of the scrol view to *color*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

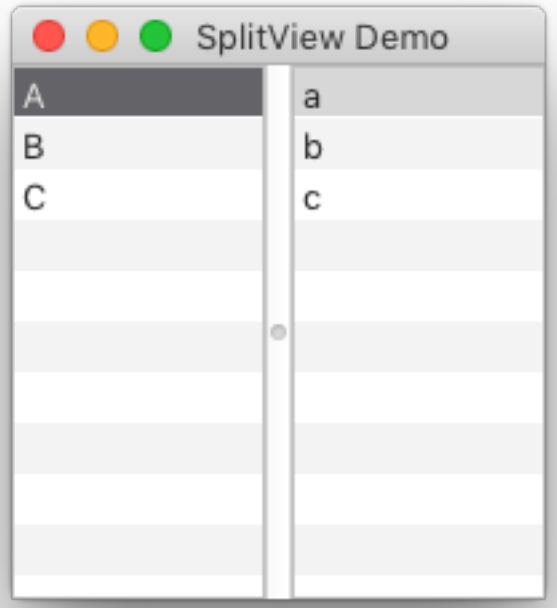
animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.4 SplitView

class vanilla.**SplitView**(*posSize*, *paneDescriptions*, *isVertical=True*, *dividerStyle='splitter'*, *dividerThickness=None*, *dividerColor=None*, *autosaveName=None*, *dividerImage=None*)
View that can be split into two or more subviews with dividers.




```
from vanilla import Window, List, SplitView

class SplitViewDemo:

    def __init__(self):
        self.w = Window((200, 200), "SplitView Demo", minSize=(100, 100))
        list1 = List((0, 0, -0, -0), ["A", "B", "C"])
        list2 = List((0, 0, -0, -0), ["a", "b", "c"])
        paneDescriptors = [
            dict(view=list1, identifier="pane1"),
            dict(view=list2, identifier="pane2"),
        ]
        self.w.splitView = SplitView((0, 0, -0, -0), paneDescriptors)
        self.w.open()

SplitViewDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the split view.

paneDescriptions An ordered list of dictionaries describing the subviews, or "panes". Those dictionaries can have the following keys:

view	A view, either a Vanilla object or a NSView. Required.
"identifier"	A string identifying the pane. Required.
"size"	The initial size of the pane. Optional.
"minSize"	The minimum size of the pane. Optional. The default is 0.
"maxSize"	The maximum size of the pane. Optional. The default is no maximum size.
"canCollapse"	Boolean indicating if the pane can collapse. Optional. The default is *True*.
"resizeFlexibility"	Boolean indicating if the pane can adjust its size automatically when the SplitView size changes. Optional. The default is *True* unless the pane has a fixed size.

isVertical Boolean representing if the split view is vertical. Default is *True*.

dividerStyle String representing the style of the divider. These are the options:

splitter
thin
thick
None

dividerThickness An integer representing the desired thickness of the divider.

dividerColor A NSColor that should be used to paint the divider.

autosaveName The autosave name for the SplitView.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSSplitView()
Return the NSSplitView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isPaneVisible(*identifier*)
Returns a boolean indicating if the pane with *identifier* is visible or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setDividerDrawingFunction(*function*)
Set a function that will draw the contents of the divider. This can be *None* or a function that accepts the following arguments:

splitView	The SplitView calling the function.
rect	The rectangle containing the divider.

The function must use the Cocoa drawing API.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

showPane(*identifier*, *onOff*, *animate=False*)
Set the visibility of the pane with *identifier*.

onOff should be a boolean indicating the desired visibility of the pane. If *animate* is True, the pane will expand or collapse with an animation.

togglePane(*identifier*, *animate=False*)
Toggle the visibility of the pane with *identifier*. If *animate* is True, the pane will expand or collapse with and animation.

2.2.5 Box

class vanilla.**Box**(*posSize*, *title=None*, *fillColor=None*, *borderColor=None*, *borderWidth=None*, *cornerRadius=None*, *margins=None*)
A bordered container for other controls.

To add a control to a box, simply set it as an attribute of the box.

```
from vanilla import Window, Box, TextBox

class BoxDemo:

    def __init__(self):
        self.w = Window((150, 70))
        self.w.box = Box((10, 10, -10, -10))
        self.w.box.text = TextBox((10, 10, -10, -10), "This is a box")
```

(continues on next page)

(continued from previous page)

```
        self.w.open()

BoxDemo()
```

No special naming is required for the attributes. However, each attribute must have a unique name.

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the box.

title The title to be displayed above the box. Pass *None* if no title is desired.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSBox()
Return the NSBox that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the title of the box.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setBorderColor(*color*)
Set the border color of the box.

setBorderWidth(*value*)
Set the border width of the box.

setCornerRadius(*value*)
Set the corner radius of the box.

setFillColor(*color*)
Set the fill color of the box.

setMargins(*value*)
Set the x, y margins for the content.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the title of the box.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.6 Popover

class vanilla.**Popover**(*size*, *parentView=None*, *preferredEdge='top'*, *behavior='semitransient'*)
A popover capable of containing controls.


```
from vanilla import Window, List, Popover, TextBox

class PopoverExample:

    def __init__(self):
        self.w = Window((120, 120))
        self.w.list = List((0, 0, -0, -0),
                ['A', 'B', 'C'],
                selectionCallback=self.showPopoverCallback)
        self.w.open()

    def showPopoverCallback(self, sender):
        selection = sender.getSelection()
        if not selection:
            return
        index = sender.getSelection()[0]
        relativeRect = sender.getNSTableView().rectOfRow_(index)
        self.pop = Popover((140, 80))
        self.pop.text = TextBox((10, 10, -10, -10), 'This is a popover.')
        self.pop.open(parentView=sender.getNSTableView(), preferredEdge='right',␣
↪relativeRect=relativeRect)

PopoverExample()
```

size Tuple of form *(width, height)* representing the size of the content in the popover.

parentView The parent view that the popover should pop out from. This can be either a vanilla object or an instance of NSView or *NSView_* subclass.

preferredEdge The edge of the parent view that you want the popover to pop out from. These are the options:

"left"
"right"
"top"
"bottom"

behavior The desired behavior of the popover. These are the options:

"applicationDefined"	Corresponds to NSPopoverBehaviorApplicationDefined.
"transient"	Corresponds to NSPopoverBehaviorTransient.
"semitransient"	Corresponds to NSPopoverBehaviorSemitransient.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must by a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*event*, *callback*)
Bind a callback to an event.

event A string representing the desired event. The options are:

"will show"	Called immediately before the popover shows.
"did show"	Called immediately after the popover shows.
"will close"	Called immediately before the popover closes.
"did close"	Called immediately after the popover closes.

close()
Close the popover.

Once a popover has been closed it can not be re-opened.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

open(*parentView=None*, *preferredEdge=None*, *relativeRect=None*)
Open the popover. If desired, the *parentView* may be specified. If not, the values assigned during init will be used.

Additionally, a rect of form *(x, y, width, height)* may be specified to indicate where the popover should pop out from. If not provided, the parent view's bounds will be used.

resize(*width*, *height*)
Change the size of the popover to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

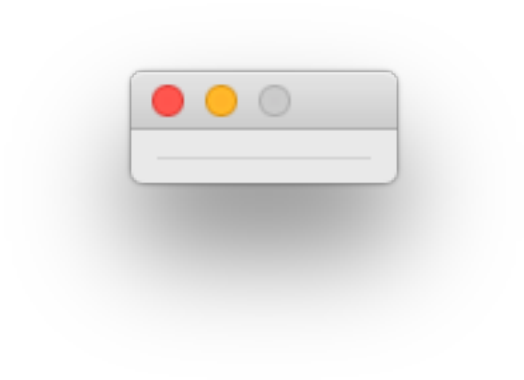
unbind(*event*, *callback*)
Unbind a callback from an event.

event A string representing the desired event. Refer to *Popover.bind()* for the options.

callback The callback that has been bound to the event.

2.2.7 HorizontalLine

class vanilla.**HorizontalLine**(*posSize*)
A horizontal line.



```
from vanilla import Window, HorizontalLine

class HorizontalLineDemo:

    def __init__(self):
        self.w = Window((100, 20))
        self.w.line = HorizontalLine((10, 10, -10, 1))
        self.w.open()

HorizontalLineDemo()
```

posSize Tuple of form *(left, top, width, height)* representing the position and size of the line.

Standard Dimensions	
H	1

addAutoPosSizeRules(*rules*, *metrics=None*)
Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSBox()
Return the NSBox that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the title of the box.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setBorderColor(*color*)
Set the border color of the box.

setBorderWidth(*value*)
Set the border width of the box.

setCornerRadius(*value*)
Set the corner radius of the box.

setFillColor(*color*)
Set the fill color of the box.

setMargins(*value*)
Set the x, y margins for the content.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

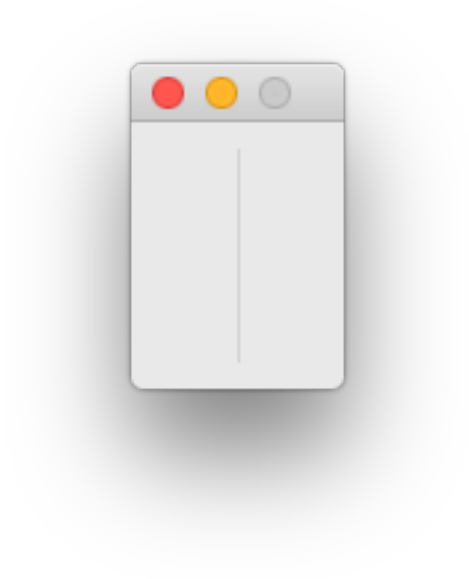
setTitle(*title*)
Set the title of the box.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.8 VerticalLine

class vanilla.**VerticalLine**(*posSize*)
A vertical line.



```
from vanilla import Window, VerticalLine

class VerticalLineDemo:

    def __init__(self):
        self.w = Window((80, 100))
        self.w.line = VerticalLine((40, 10, 1, -10))
        self.w.open()

VerticalLineDemo()
```

posSize Tuple of form *(left, top, width, height)* representing the position and size of the line.

Standard Dimensions	
V	1

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSBox()
Return the NSBox that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the title of the box.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setBorderColor(*color*)
Set the border color of the box.

setBorderWidth(*value*)
Set the border width of the box.

setCornerRadius(*value*)
Set the corner radius of the box.

setFillColor(*color*)
Set the fill color of the box.

setMargins(*value*)
Set the x, y margins for the content.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the title of the box.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.9 GridView

class vanilla.**GridView**(*posSize*, *contents*, *columnWidth=None*, *columnSpacing=0*, *columnPadding=(0, 0)*, *columnPlacement='leading'*, *rowHeight=None*, *rowSpacing=0*, *rowPadding=(0, 0)*, *rowPlacement='top'*, *rowAlignment='firstBaseline'*, *columnDescriptions=None*)
A view that allows the placement of other views within a grid.


```
from vanilla import Button, GridView, Window

class GridViewExample:
    def __init__(self):
        self.w = Window((120, 90))

        self.button1 = Button("auto", "one")
        self.button2 = Button("auto", "two")
        self.button3 = Button("auto", "three")
        self.button4 = Button("auto", "four")

        self.w.gridView = GridView(
            (0, 0, 0, 0),
            contents=[
                dict(
                    cells=[
                        dict(view=self.button1),
                        dict(view=self.button2),
                    ]
                ),
                dict(
                    cells=[
                        dict(view=self.button3),
                        dict(view=self.button4),
                    ]
                ),
            ],
            columnPadding=(4, 4),
            rowPadding=(4, 4),
        )

        self.w.open()


GridViewExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the grid view.

contents The contents to display within the grid. See below for structure.

columnWidth The width for columns.

columnSpacing The amount of spacing between columns.

columnPadding The (left, right) padding for columns.

columnPlacement The horizontal placement of content within columns. Options:

- "leading"
- "center"
- "trailing"
- "fill"

rowHeight The height for rows.

rowSpacing The amount of spacing between rows.

rowPadding The (top, bottom) padding for rows.

rowPlacement The vertical placement of content within rows. Options:

- "top"
- "center"
- "bottom"
- "fill"

rowAlignment The alignment of the row. Options:

- "firstBaseline"
- "lastBaseline"
- "none"

columnDescriptions An optional list of dictionaries defining specific attributes for the columns. Options:

- "width"
- "columnPadding"
- "columnPlacement"

Contents Definition Structure

Contents are defined as with a list of row definitions. A row definition is a list of cell definitions or a dictionary with this structure:

- **cells** A list of cell definitions.
- **rowHeight** (optional) A height for the row that overrides the GridView level row height.
- **rowPadding** (optional) A (top, bottom) padding definition for the row that overrides the GridView level row padding.
- **rowPlacement** (optional) A placement for the row that overrides the GridView level row placement.
- **rowAlignment** (optional) An alignment for the row that overrides the GridView level row placement.

Cells are defined with either a Vanilla object, a NSView (or NSView subclass) object, None, or a dictionary with this structure:

- **view** A Vanilla object or NSView (or NSView subclass) object.

- **width** (optional) A width to apply to the view.
- **height** (optional) A height to apply to the view.
- **columnPlacement** (optional) A horizontal placement for the cell that overrides the row level or GridView level placement.
- **rowPlacement** (optional) A vertical placement for the cell that overrides the row level or GridView level placement.
- **rowAlignment** (optional) A row alignment for the cell that overrides the row level or GridView level alignment.

If a cell is defined as None, the cell will be merged with the first cell directly above that has content.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

appendColumn(*cells*, *columnWidth=None*, *columnPadding=None*, *columnPlacement=None*)
Append a column and populate it with a list of cells. The cells must have the same structure as defined in *__init__*.

appendRow(*cells*, *rowHeight=None*, *rowPadding=None*, *rowPlacement=None*, *rowAlignment=None*)
Append a row and populate it with a list of cells. The cells must have the same structure as defined in *__init__*.

Merging is not possible with this method.

columnIsVisible(*index*)
Get the visibility of column at *index*.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getColumnCount()
Get the number of columns.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getRowCount()
Get the number of rows.

insertColumn(*index*, *cells*, *columnWidth=None*, *columnPadding=None*, *columnPlacement=None*)
Insert a column at *index* and populate it with a list of cells. The cells must have the same structure as defined in *__init__*.

insertRow(*index*, *cells*, *rowHeight=None*, *rowPadding=None*, *rowPlacement=None*, *rowAlignment=None*)
Insert a row at *index* and populate it with a list of cells. The cells definition must have the same structure as defined in *__init__*.

Merging is not possible with this method.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

moveColumn(*fromIndex*, *toIndex*)
Move column at *fromIndex* to *toIndex*.

moveRow(*fromIndex*, *toIndex*)
Move row at *fromIndex* to *toIndex*.

removeColumn(*index*)
Remove column at *index*.

removeRow (*index*)
Remove row at *index*.

resize (*width*, *height*)
Change the size of the object to *width* and *height*.

rowIsVisible (*index*)
Get the visibility of row at *index*.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show (*onOff*)
Show or hide the object.

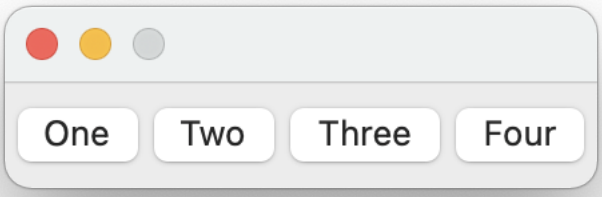
onOff A boolean value representing if the object should be shown or not.

showColumn (*index*, *value*)
Set the visibility of column at *index*.

showRow (*index*, *value*)
Set the visibility of row at *index*.

2.2.10 HorizontalStackView

class vanilla.**HorizontalStackView** (*posSize*, *views*, *spacing=0*, *alignment='center'*, *distribution='fillEqually'*, *edgeInsets=(0, 0, 0, 0)*)
A view that allows the creation of a stack of views.




```
from vanilla import Button, HorizontalStackView, Window

class HorizontalStackViewExample:
    def __init__(self):
        self.w = Window((300, 40))

        self.button1 = Button("auto", "One")
        self.button2 = Button("auto", "Two")
        self.button3 = Button("auto", "Three")
```

(continues on next page)

(continued from previous page)

```
        self.button4 = Button("auto", "Four")

        self.w.horizontalStack = HorizontalStackView(
            (0, 0, 0, 0),
            views=[
                dict(view=self.button1),
                dict(view=self.button2),
                dict(view=self.button3),
                dict(view=self.button4)
            ],
            spacing=4,
            edgeInsets=(4, 4, 4, 4),
        )

        self.w.open()


HorizontalStackViewExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the stack view.

views The views to display in the stack. See below for structure.

spacing Space to insert between views.

alignment The alignment of the views. Options:

- "leading"
- "center"
- "trailing"
- One of the NSLayoutAttribute options.

distribution The distribution of the views. Options:

- "equalCentering"
- "equalSpacing"
- "fill"
- "fillEqually"
- "fillProportionally"
- "gravity"

edgeInsets Tuple of four numbers (left, top, right, bottom) indicating the amount to inset the views.

View Definition Structure

Views are defined with either a Vanilla object, a NSView (or NSView subclass) object or a dictionary with this structure:

- **view** A vanilla object or an instance of NSView.
- **width** A number, string (using the size syntax below) or None (indicating auto width) defining the width.
- **height** A number, string (using the size syntax below) or None (indicating auto height) defining the height.
- **gravity** The gravity that this view should be attracted to.
- **spacing** A number defining custom spacing after the view.

Size Constants:

- "fill" Stretch the view to fit the width or height of the stack view.
- "fit" Fit the view to the size needed to contain its contents.

Size Syntax:

- "==value" where value can be coerced to an integer or float
- "<=value" where value can be coerced to an integer or float
- ">=value" where value can be coerced to an integer or float

Up to two are allowed. Separate with `,` (comma).

Horizontal Gravity Options:

- "leading"
- "center"
- "trailing"
- One of the NSStackViewGravity options.

Vertical Gravity Options:

- "top"
- "center"
- "bottom"
- One of the NSStackViewGravity options.

`addAutoPosSizeRules`(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

appendView(*view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Append a view.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

insertView(*index*, *view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Insert a view.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

removeView(*view*)
Remove a view.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setEdgeInsets(*value*)
Set the edge insets.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

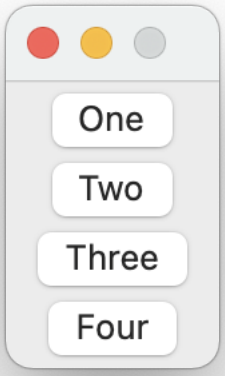
animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.2.11 VerticalStackView

class vanilla.**VerticalStackView**(*posSize*, *views*, *spacing=0*, *alignment='center'*, *distribution='fillEqually'*, *edgeInsets=(0, 0, 0, 0)*)
A view that allows the creation of a stack of views.




```
from vanilla import Button, VerticalStackView, Window


class VerticalStackViewExample:
    def __init__(self):
        self.w = Window((80, 300))

        self.button1 = Button("auto", "One")
        self.button2 = Button("auto", "Two")
        self.button3 = Button("auto", "Three")
        self.button4 = Button("auto", "Four")

        self.w.horizontalStack = VerticalStackView(
            (0, 0, 0, 0),
            views=[
                dict(view=self.button1),
                dict(view=self.button2),
                dict(view=self.button3),
                dict(view=self.button4)
```

(continues on next page)

(continued from previous page)

```
            ],
            spacing=4,
            edgeInsets=(4, 4, 4, 4),
        )

        self.w.open()


VerticalStackViewExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the stack view.

views The views to display in the stack. See below for structure.

spacing Space to insert between views.

alignment The alignment of the views. Options:

- "leading"
- "center"
- "trailing"
- One of the NSLayoutAttribute options.

distribution The distribution of the views. Options:

- "equalCentering"
- "equalSpacing"
- "fill"
- "fillEqually"
- "fillProportionally"
- "gravity"

edgeInsets Tuple of four numbers (left, top, right, bottom) indicating the amount to inset the views.

View Definition Structure

Views are defined with either a Vanilla object, a NSView (or NSView subclass) object or a dictionary with this structure:

- **view** A vanilla object or an instance of NSView.
- **width** A number, string (using the size syntax below) or None (indicating auto width) defining the width.
- **height** A number, string (using the size syntax below) or None (indicating auto height) defining the height.
- **gravity** The gravity that this view should be attracted to.
- **spacing** A number defining custom spacing after the view.

Size Constants:

- "fill" Stretch the view to fit the width or height of the stack view.
- "fit" Fit the view to the size needed to contain its contents.

Size Syntax:

- "==value" where value can be coerced to an integer or float

- "<=value" where value can be coerced to an integer or float
- ">=value" where value can be coerced to an integer or float

Up to two are allowed. Separate with `,` (comma).

Horizontal Gravity Options:

- "leading"
- "center"
- "trailing"
- One of the NSStackViewGravity options.

Vertical Gravity Options:

- "top"
- "center"
- "bottom"
- One of the NSStackViewGravity options.

`addAutoPosSizeRules`(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

appendView(*view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Append a view.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

insertView(*index*, *view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Insert a view.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

removeView(*view*)
Remove a view.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setEdgeInsets(*value*)
Set the edge insets.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.3 Data Views

2.3.1 List2

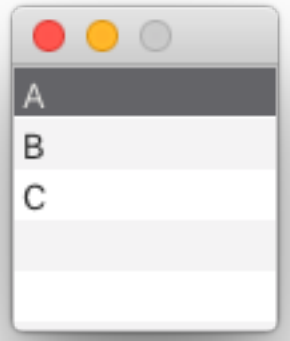
2.3.2 List2 Item Cells

2.3.3 List

class vanilla.**List**(*posSize*, *items*, *dataSource=None*, *columnDescriptions=None*, *showColumnTitles=True*, *selectionCallback=None*, *doubleClickCallback=None*, *editCallback=None*, *menuCallback=None*, *enableDelete=False*, *enableTypingSensitivity=False*, *allowsMultipleSelection=True*, *allowsEmptySelection=True*, *allowsSorting=True*, *drawVerticalLines=False*, *drawHorizontalLines=False*, *autohidesScrollers=True*, *drawFocusRing=True*, *rowHeight=17.0*, *selfDropSettings=None*, *selfWindowDropSettings=None*, *selfDocumentDropSettings=None*, *selfApplicationDropSettings=None*, *otherApplicationDropSettings=None*, *dragSettings=None*)

A control that shows a list of items. These lists can contain one or more columns.

A single column example:




```
from vanilla import Window, List

class ListDemo:
```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0), ["A", "B", "C"],
                        selectionCallback=self.selectionCallback)
        self.w.open()

    def selectionCallback(self, sender):
        print(sender.getSelection())

ListDemo()
```

A multiple column example:


```
from vanilla import Window, List

class ListDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0),
                        [{"One": "A", "Two": "a"}, {"One": "B", "Two": "b"}],
                        columnDescriptions=[{"title": "One"}, {"title": "Two"}],
                        selectionCallback=self.selectionCallback)
        self.w.open()

    def selectionCallback(self, sender):
        print(sender.getSelection())

ListDemo()
```

List objects behave like standard Python lists. For example, given this List:

```
self.w.myList = List((10, 10, 200, 100), ["A", "B", "C"])
```

The following Python list methods work:

```
# Getting the length of the List.
>>> len(self.w.myList)
3

# Retrieving an item or items from a List.
>>> self.w.myList[1]
"B"
>>> self.w.myList[:2]
["A", "B"]

# Setting an item in a List.
>>> self.w.myList[1] = "XYZ"
>>> self.w.myList.get()
["A", "XYZ", "C"]

# Deleting an item at an index in a List.
>>> del self.w.myList[1]
>>> self.w.myList.get()
["A", "C"]

# Appending an item to a List.
>>> self.w.myList.append("Z")
>>> self.w.myList.get()
["A", "B", "C", "Z"]

# Removing the first occurance of an item in a List.
>>> self.w.myList.remove("A")
>>> self.w.myList.get()
["B", "C"]

# Getting the index for the first occurance of an item in a List.
>>> self.w.myList.index("B")
1

# Inserting an item into a List.
>>> self.w.myList.insert(1, "XYZ")
>>> self.w.myList.get()
["A", "XYZ", "B", "C"]

# Extending a List.
>>> self.w.myList.extend(["X", "Y", "Z"])
>>> self.w.myList.get()
["A", "B", "C", "X", "Y", "Z"]

# Iterating over a List.
>>> for i in self.w.myList:
>>>     i
"A"
"B"
"C"
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the list.

items The items to be displayed in the list. In the case of multiple column lists, this should be a list of dictionaries with the data for each column keyed by the column key as defined in *columnDescriptions*. If you intend to use a dataSource, *items* must be *None*.

dataSource A Cocoa object supporting the NSTableViewDataSource protocol. If *dataSource* is given, *items* must be *None*.

columnDescriptions An ordered list of dictionaries describing the columns. This is only necessary for multiple column lists.

"title"	The title to appear in the column header.
"key" (optional)	The key from which this column should get its data from each dictionary in *items*. If nothing is given, the key will be the string given in *title*.
"formatter" (optional)	An NSFormatter for controlling the display and input of the column's cells.
"cell" (optional)	A cell type to be displayed in the column. If nothing is given, a text cell is used.
"editable" (optional)	Enable or disable editing in the column. If nothing is given, it will follow the editability of the rest of the list.
"width" (optional)	The width of the column.
"minWidth" (optional)	The minimum width of the column. The fallback is *width*.
"maxWidth" (optional)	The maximum width of the column. The fallback is *width*.
"allowsSorting" (optional)	A boolean representing that this column allows the user to sort the table by clicking the column's header. The fallback is *True*. If a List is set to disallow sorting the column level settings will be ignored.
"typingSensitive" (optional)	A boolean representing that this column should be the column that responds to user key input. Only one column can be flagged as *True*. If no column is flagged, the first column will automatically be flagged.
binding (optional)	A string indicating which binding object the column's cell should be bound to. By default, this is "value". You should only override this in very specific cases.

showColumnTitles Boolean representing if the column titles should be shown or not. Column titles will not be shown in single column lists.

selectionCallback Callback to be called when the selection in the list changes.

doubleClickCallback Callback to be called when an item is double clicked.

editCallback Callback to be called after an item has been edited.

menuCallback Callback to be called when a contextual menu is requested.

enableDelete A boolean representing if items in the list can be deleted via the interface.

enableTypingSensitivity A boolean representing if typing in the list will jump to the closest match as the entered keystrokes. *Available only in single column lists.*

allowsMultipleSelection A boolean representing if the list allows more than one item to be selected.

allowsEmptySelection A boolean representing if the list allows zero items to be selected.

allowsSorting A boolean indicating if the list allows user sorting by clicking column headers.

drawVerticalLines Boolean representing if vertical lines should be drawn in the list.

drawHorizontalLines Boolean representing if horizontal lines should be drawn in the list.

drawFocusRing Boolean representing if the standard focus ring should be drawn when the list is selected.

rowHeight The height of the rows in the list.

autohidesScrollers Boolean representing if scrollbars should automatically be hidden if possible.

selfDropSettings A dictionary defining the drop settings when the source of the drop is this list. The dictionary form is described below.

selfWindowDropSettings A dictionary defining the drop settings when the source of the drop is contained the same window as this list. The dictionary form is described below.

selfDocumentDropSettings A dictionary defining the drop settings when the source of the drop is contained the same document as this list. The dictionary form is described below.

selfApplicationDropSettings A dictionary defining the drop settings when the source of the drop is contained the same application as this list. The dictionary form is described below.

otherApplicationDropSettings A dictionary defining the drop settings when the source of the drop is contained an application other than the one that contains this list. The dictionary form is described below.

The drop settings dictionaries should be of this form:

type	A single drop type indicating what drop types the list accepts. For example, "NSFilenamesPboardType" or "MyCustomPboardType".
operation (optional)	A drag operation that the list accepts. The default is *NSDragOperationCopy*.
allowDropBetweenRows (optional)	A boolean indicating if the list accepts drops between rows. The default is *True*.
allowDropOnRow (optional)	A boolean indicating if the list accepts drops on rows. The default is *False*.
callback	Callback to be called when a drop is proposed and when a drop is to occur. This method should return a boolean representing if the drop is acceptable or not. This method must accept *sender* and *dropInfo* arguments. The *dropInfo* will be a dictionary as described below.

The *dropInfo* dictionary passed to drop callbacks will be of this form:

data	The data proposed for the drop. This data will be of the type specified by *dropDataFormat*.
rowIndex	The row where the drop is proposed.
source	The source from which items are being dragged. If this object is wrapped by Vanilla, the Vanilla object will be passed as the source.
dropOnRow	A boolean representing if the row is being dropped on. If this is *False*, the drop should occur between rows.
isProposal	A boolean representing if this call is simply proposing the drop or if it is time to accept the drop.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. **onOff** should be a boolean.

get()
Get the list of items in the list.

getNSScrollView()
Return the NSScrollView that this object wraps.

getNSTableView()
Return the NSTableView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getSelection()
Get a list of indexes of selected items in the list.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

scrollToSelection()
Scroll the selected rows to visible.

set(*items*)
Set the items in the list.

items should follow the same format as described in the constructor.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setSelection(*selection*)
Set the selected items in the list.

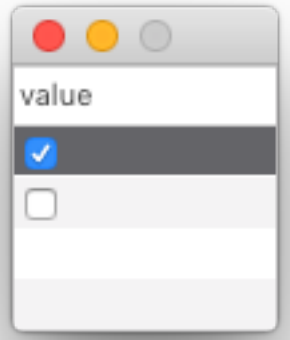
selection should be a list of indexes.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.3.4 List Item Cells

vanilla.**CheckBoxListCell**(*title=None*)
An object that displays a check box in a List column.



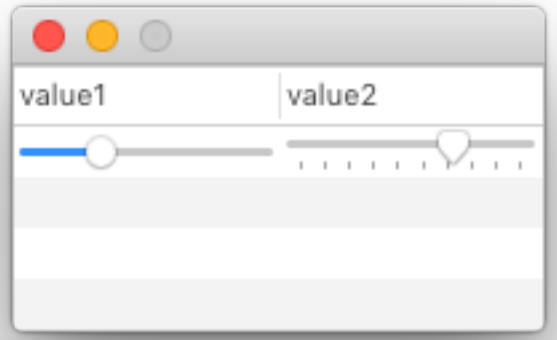
Note: This object should only be used in the *columnDescriptions* argument during the construction of a List.

```
from vanilla import Window, List, CheckBoxListCell

class CheckBoxListCellDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0),
                     [{"value": True}, {"value": False}],
                     columnDescriptions=[{"title": "value", "cell":␣
↪CheckBoxListCell()}],
                     editCallback=self.editCallback)
        self.w.open()

    def editCallback(self, sender):
        print(sender.get())

CheckBoxListCellDemo()
```

title The title to be set in *all* items in the List column.

vanilla.**SliderListCell**(*minValue=0*, *maxValue=100*, *tickMarkCount=None*, *stopOnTickMarks=False*)

An object that displays a slider in a List column.



Note: This object should only be used in the *columnDescriptions* argument during the construction of a List.

```
from vanilla import Window, List, SliderListCell

class SliderListCellDemo:

    def __init__(self):
        self.w = Window((200, 100))
        self.w.myList = List((0, 0, -0, -0),
                    [{"value1": 30, "value2": 70}],
                    columnDescriptions=[
                        {"title": "value1", "cell": SliderListCell()},
                        {"title": "value2", "cell": SliderListCell(tickMarkCount=10)},
                    ],
                    editCallback=self.editCallback)
        self.w.open()

    def editCallback(self, sender):
        print(sender.get())

SliderListCellDemo()
```

minValue The minimum value for the slider.

maxValue The maximum value for the slider.

tickMarkCount The number of tick marcks to be displayed on the slider. If *None* is given, no tick marks will be displayed.

stopOnTickMarks Boolean representing if the slider knob should only stop on the tick marks.

`vanilla.`**`PopUpButtonListCell`**(*items*)

An object that displays a pop up list in a List column.

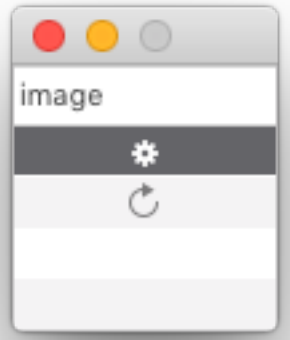
Note: When using this cell in a List, the *binding* in the column description must be set to *selectedValue*.

```
from vanilla import Window, List, PopUpButtonListCell

class PopUpButtonListCellDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0),
                    [{"value": "A"}, {"value": "B"}],
                    columnDescriptions=[{
                        "title": "value",
                        "cell": PopUpButtonListCell(["A", "B", "C"]),
                        "binding": "selectedValue"
                    }],
                    editCallback=self.editCallback)
        self.w.open()

    def editCallback(self, sender):
        print(sender.get())

PopUpButtonListCellDemo()
```

items The items that should appear in the pop up list.

vanilla.**ImageListCell**(*horizontalAlignment='center'*, *verticalAlignment='center'*, *scale='proportional'*)

An object that displays an image in a List column.




```
from AppKit import NSImage
from vanilla import Window, List, ImageListCell

class ImageListCellDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0),
                    [
                        {"image": NSImage.imageNamed_("NSActionTemplate")},
                        {"image": NSImage.imageNamed_("NSRefreshTemplate")}
                    ],
                    columnDescriptions=[
                        {"title": "image", "cell": ImageListCell()}
                    ])
        self.w.open()

ImageListCellDemo()
```

horizontalAlignment A string representing the desired horizontal alignment of the image in the view. The options are:

“left”	Image is aligned left.
“right”	Image is aligned right.
“center”	Image is centered.

verticalAlignment A string representing the desired vertical alignment of the image in the view. The options are:

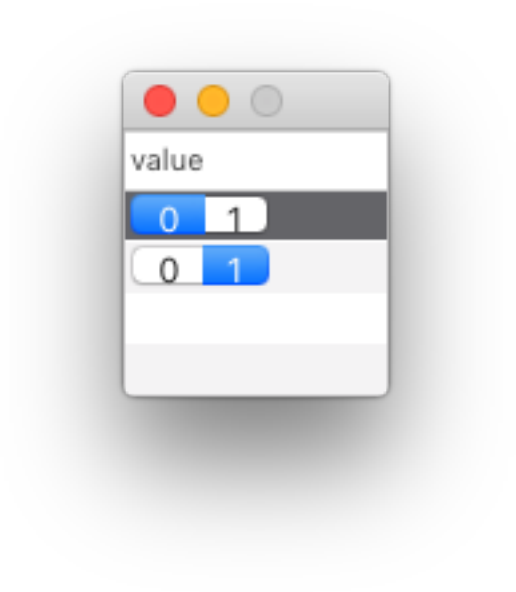
“top”	Image is aligned top.
“bottom”	Image is aligned bottom.
“center”	Image is centered.

scale A string representing the desired scale style of the image in the view. The options are:

“proportional”	Proportionally scale the image to fit in the view if it is larger than the view.
“fit”	Distort the proportions of the image until it fits exactly in the view.
“none”	Do not scale the image.

vanilla.**SegmentedButtonListCell**(*segmentDescriptions*)

An object that displays a segmented button in a List column.



Note: When using this cell in a List, the *binding* in the column description must be set to *selectedIndex*.

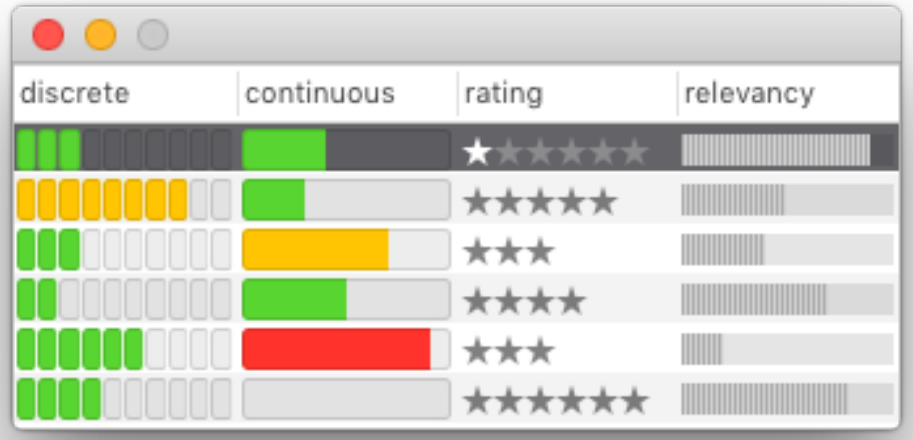
```
from vanilla import Window, List, SegmentedButtonListCell

class SegmentedButtonListCellDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.myList = List((0, 0, -0, -0),
                        [{"value": 0}, {"value": 1}],
                        columnDescriptions=[
                            {
                                "title": "value",
                                "cell": SegmentedButtonListCell([dict(title="0"),␣
↪dict(title="1")]),
                                "binding": "selectedIndex"
                            }
                        ],
                        editCallback=self.editCallback)
        self.w.open()

    def editCallback(self, sender):
        print(sender.get())

SegmentedButtonListCellDemo()
```

segmentDescriptions An ordered list of dictionaries describing the segments.

title (optional)	The title of the segment.
imagePath (optional)	A file path to an image to display in the segment.
imageNamed (optional)	The name of an image already loaded as a NSImage by the application to display in the segment.
imageObject (optional)	A NSImage object to display in the segment.

vanilla.**LevelIndicatorListCell**(*style='discrete'*, *minValue=0*, *maxValue=10*, *warningValue=None*, *criticalValue=None*, *imagePath=None*, *imageNamed=None*, *imageObject=None*)

An object that displays a level indicator in a List column.



Note: This object should only be used in the *columnDescriptions* argument during the construction of a List.

```
from vanilla import Window, List, LevelIndicatorListCell

class LevelIndicatorListCellDemo:

    def __init__(self):
        self.w = Window((340, 140))
        items = [
            {"discrete": 3, "continuous": 4, "rating": 1, "relevancy": 9},
            {"discrete": 8, "continuous": 3, "rating": 5, "relevancy": 5},
            {"discrete": 3, "continuous": 7, "rating": 3, "relevancy": 4},
            {"discrete": 2, "continuous": 5, "rating": 4, "relevancy": 7},
            {"discrete": 6, "continuous": 9, "rating": 3, "relevancy": 2},
```

(continues on next page)

(continued from previous page)

```
            {"discrete": 4, "continuous": 0, "rating": 6, "relevancy": 8},
        ]
        columnDescriptions = [
            {"title": "discrete",
            "cell": LevelIndicatorListCell(style="discrete", warningValue=7,␣
↪criticalValue=9)},
            {"title": "continuous",
            "cell": LevelIndicatorListCell(style="continuous", warningValue=7,␣
↪criticalValue=9)},
            {"title": "rating",
            "cell": LevelIndicatorListCell(style="rating", maxValue=6)},
            {"title": "relevancy",
            "cell": LevelIndicatorListCell(style="relevancy")},
        ]
        self.w.list = List((0, 0, -0, -0), items=items,
                           columnDescriptions=columnDescriptions)
        self.w.open()

LevelIndicatorListCellDemo()
```

style The style of the level indicator. The options are:

"continuous"	A continuous bar.
"discrete"	A segmented bar.
"rating"	A row of stars. Similar to the rating indicator in iTunes.
"relevancy"	A row of lines. Similar to the search result relevancy indicator in Mail.

minValue The minimum value allowed by the level indicator.

maxValue The maximum value allowed by the level indicator.

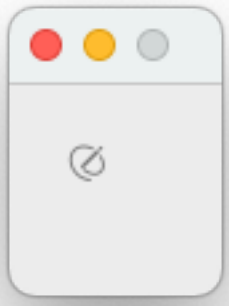
warningValue The value at which the filled portions of the level indicator should display the warning color. Applies only to discrete and continuous level indicators.

criticalValue The value at which the filled portions of the level indicator should display the critical color. Applies only to discrete and continuous level indicators.

2.3.5 ImageView

class vanilla.**ImageView**(*posSize*, *horizontalAlignment='center'*, *verticalAlignment='center'*, *scale='proportional'*)

A view that displays an image.



```
import AppKit
from vanilla import ImageView, Window

class ImageViewExample:
    def __init__(self):
        self.w = Window((80, 80))
        self.w.imageView = ImageView(
            (10, 10, 40, 40),
            horizontalAlignment="center",
            verticalAlignment="center",
            scale="proportional"
        )
        image = AppKit.NSImage.imageWithSystemSymbolName_accessibilityDescription_
↪("pencil.and.outline", "")
        self.w.imageView.setImage(imageObject=image)
        self.w.open()


ImageViewExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the view.

horizontalAlignment A string representing the desired horizontal alignment of the image in the view. The options are:

"left"	Image is aligned left.
"right"	Image is aligned right.
"center"	Image is centered.

verticalAlignment A string representing the desired vertical alignment of the image in the view. The options are:

"top"	Image is aligned top.
"bottom"	Image is aligned bottom.
"center"	Image is centered.

scale A string representing the desired scale style of the image in the view. The options are:

"proportional"	Proportionally scale the image to fit in the view if it is larger than the view.
"fit"	Distort the proportions of the image until it fits exactly in the view.
"none"	Do not scale the image.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSImageView()
Return the NSImageView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setImage(*imagePath=None*, *imageNamed=None*, *imageObject=None*)
Set the image in the view.

imagePath A file path to an image.

imageNamed The name of an image already load as a NSImage by the application.

imageObject A NSImage object.

Note: Only one of *imagePath*, *imageNamed*, *imageObject* should be set.

---

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

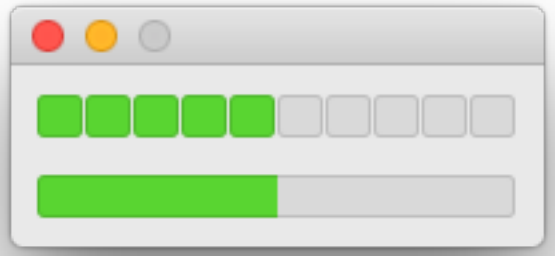
animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.3.6 LevelIndicator

class vanilla.**LevelIndicator**(*posSize*, *style='discrete'*, *value=5*, *minValue=0*, *maxValue=10*, *warningValue=None*, *criticalValue=None*, *tickMarkPosition=None*, *minorTickMarkCount=None*, *majorTickMarkCount=None*, *callback=None*)
A control which shows a value on a linear scale.



```
from vanilla import Window, LevelIndicator

class LevelIndicatorDemo:

    def __init__(self):
        self.w = Window((200, 68))
        self.w.discreteIndicator = LevelIndicator(
                    (10, 10, -10, 18), callback=self.levelIndicatorCallback)
        self.w.continuousIndicator = LevelIndicator(
                    (10, 40, -10, 18), style="continuous",
                    callback=self.levelIndicatorCallback)
        self.w.open()

    def levelIndicatorCallback(self, sender):
        print("level indicator edit!", sender.get())

LevelIndicatorDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the level indicator.

Standard Dimensions		
discrete without ticks	H	18
discrete with minor ticks	H	22
discrete with major ticks	H	25
continuous without ticks	H	16
continuous with minor ticks	H	20
continuous with major ticks	H	23

style The style of the level indicator. The options are:

"continuous"	A continuous bar.
"discrete"	A segmented bar.

value The initial value of the level indicator.

minValue The minimum value allowed by the level indicator.

maxValue The maximum value allowed by the level indicator.

warningValue The value at which the filled portions of the level indicator should display the warning color.

criticalValue The value at which the filled portions of the level indicator should display the critical color.

tickMarkPosition The position of the tick marks in relation to the level indicator. The options are:

"above"
"below"

minorTickMarkCount The number of minor tick marcks to be displayed on the level indicator. If *None* is given, no minor tick marks will be displayed.

majorTickMarkCount The number of major tick marcks to be displayed on the level indicator. If *None* is given, no major tick marks will be displayed.

callback The method to be called when the level indicator has been edited. If no callback is given, the level indicator will not be editable.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the value of the level indicator.

getCriticalValue(*value*)
Get the critical value of the level indicator.

getMaxValue()
Get the maximum of the level indicator.

getMinValue()
Get the minimum value of the level indicator.

getNSLevelIndicator()
Return the NSLevelIndicator that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

getWarningValue(*value*)
Get the warning value of the level indicator.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the value of the level indicator.

setCriticalValue(*value*)
Set the critical value of the level indicator.

setMaxValue(*value*)
Set the maximum of the level indicator.

setMinValue (*value*)
Set the minimum value of the level indicator.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle (*title*)
Set the control title.

title A string representing the title.

setWarningValue (*value*)
Set the warning value of the level indicator.

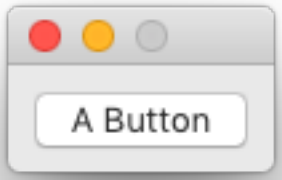
show (*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4 Buttons

2.4.1 Button

class vanilla.**Button** (*posSize*, *title*, *callback=None*, *sizeStyle='regular'*)
A standard button.




```
from vanilla import Window, Button

class ButtonDemo:

    def __init__(self):
        self.w = Window((100, 40))
        self.w.button = Button((10, 10, -10, 20), "A Button",
                               callback=self.buttonCallback)
        self.w.open()

    def buttonCallback(self, sender):
```

(continues on next page)

(continued from previous page)

```
        print("button hit!")

ButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the button. The size of the button should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	20
Small	H	17
Mini	H	14

title The text to be displayed on the button. Pass *None* if no title is desired.

callback The method to be called when the user presses the button.

sizeStyle A string representing the desired size style of the button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*key*, *modifiers*)

Bind a key to the button.

key A single character or one of the following:

"help"
"home"
"end"
"pageup"
"pagedown"
"forwarddelete"
"leftarrow"
"rightarrow"
"uparrow"
"downarrow"

modifiers A list containing nothing or as many of the following as desired:

"command"
"control"
"option"
"shift"
"capslock"

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSButton()
Return the NSButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

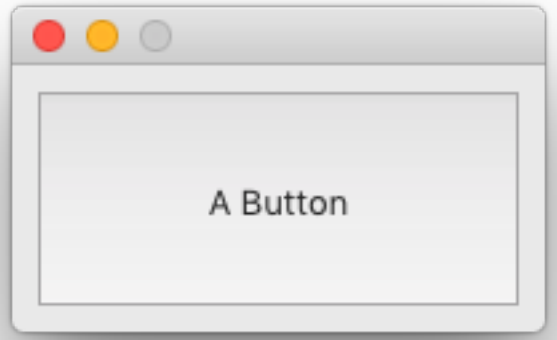
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4.2 SquareButton

class vanilla.**SquareButton**(*posSize*, *title*, *callback=None*, *sizeStyle='regular'*)
A standard square button.




```
from vanilla import *

class SquareButtonDemo:

    def __init__(self):
        self.w = Window((200, 100))
        self.w.button = SquareButton((10, 10, -10, -10), "A Button",
                                     callback=self.buttonCallback)
        self.w.open()

    def buttonCallback(self, sender):
        print("button hit!")

SquareButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the button.

title The text to be displayed on the button. Pass *None* if no title is desired.

callback The method to be called when the user presses the button.

sizeStyle A string representing the desired size style of the button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*key*, *modifiers*)

Bind a key to the button.

key A single character or one of the following:

“help”
“home”
“end”
“pageup”
“pagedown”
“forwarddelete”
“leftarrow”
“rightarrow”
“uparrow”
“downarrow”

modifiers A list containing nothing or as many of the following as desired:

“command”
“control”
“option”
“shift”
“capslock”

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSButton()
Return the NSButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4.3 ImageButton

class vanilla.**ImageButton**(*posSize*, *imagePath=None*, *imageNamed=None*, *imageObject=None*, *title=None*, *bordered=True*, *imagePosition='top'*, *callback=None*, *sizeStyle='regular'*)

A button with an image.

```
import AppKit
from vanilla import Window, ImageButton

class ImageButtonDemo:

    def __init__(self):
        self.w = Window((50, 50))
        self.w.button = ImageButton(
            (10, 10, 30, 30),
            imageNamed=AppKit.NSImageNameRefreshTemplate,
           callback=self.buttonCallback
           )
        self.w.open()

    def buttonCallback(self, sender):
        print("button hit!")

ImageButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the button.

title The text to be displayed on the button. Pass *None* if no title is desired.

bordered Boolean representing if the button should be bordered.

imagePath A file path to an image.

imageNamed The name of an image already loaded as a NSImage by the application.

imageObject A NSImage object.

Note: Only one of *imagePath*, *imageNamed*, *imageObject* should be set.

---

imagePosition The position of the image relative to the title. The options are:

"top"
"bottom"
"left"
"right"

callback The method to be called when the user presses the button.

sizeStyle A string representing the desired size style of the button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*key*, *modifiers*)
Bind a key to the button.

key A single character or one of the following:

"help"
"home"
"end"
"pageup"
"pagedown"
"forwarddelete"
"leftarrow"
"rightarrow"
"uparrow"
"downarrow"

modifiers A list containing nothing or as many of the following as desired:

"command"
"control"
"option"
"shift"
"capslock"

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSButton()
Return the NSButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setImage(*imagePath=None*, *imageNamed=None*, *imageObject=None*)
Set the image in the button.

imagePath A file path to an image.

imageNamed The name of an image already loaded as an NSImage by the application.

imageObject A NSImage object.

Note: Only one of *imagePath*, *imageNamed*, *imageObject* should be set.

---

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

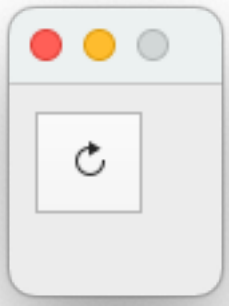
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4.4 GradientButton

class vanilla.**GradientButton**(*posSize*, *imagePath=None*, *imageNamed=None*, *imageObject=None*, *title=None*, *bordered=True*, *imagePosition='top'*, *callback=None*, *sizeStyle='regular'*)
An image button that initiates an immediate action related to a view.



```
import AppKit
from vanilla import GradientButton, Window


class GradientButtonExample:
    def __init__(self):
        self.w = Window((80, 80))
        self.w.gradientButton = GradientButton(
            (10, 10, 40, 40),
            imageNamed=AppKit.NSImageNameRefreshTemplate,
            imagePosition="top",
            callback=self.gradientButtonCallback,
            sizeStyle="regular",
        )

        self.w.open()

    def gradientButtonCallback(self, sender):
        print("gradient button hit!")


GradientButtonExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the button.

title The text to be displayed on the button. Pass *None* if no title is desired.

bordered Boolean representing if the button should be bordered.

imagePath A file path to an image.

imageNamed The name of an image already loaded as a NSImage by the application.

imageObject A NSImage object.

Note: Only one of *imagePath*, *imageNamed*, *imageObject* should be set.

---

imagePosition The position of the image relative to the title. The options are:

“top”
“bottom”
“left”
“right”

callback The method to be called when the user presses the button.

sizeStyle A string representing the desired size style of the button. The options are:

“regular”
“small”
“mini”

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
“view1”	The vanilla wrapped view for the left side of the rule.
“attribute1”	The attribute of the view for the left side of the rule. See below for options.
“relation” (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *“==”*.
“view2”	The vanilla wrapped view for the right side of the rule.
“attribute2”	The attribute of the view for the right side of the rule. See below for options.
“multiplier” (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
“constant” (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
“left”	NSLayoutAttributeLeft
“right”	NSLayoutAttributeRight
“top”	NSLayoutAttributeTop
“bottom”	NSLayoutAttributeBottom
“leading”	NSLayoutAttributeLeading
“trailing”	NSLayoutAttributeTrailing
“width”	NSLayoutAttributeWidth
“height”	NSLayoutAttributeHeight
“centerX”	NSLayoutAttributeCenterX
“centerY”	NSLayoutAttributeCenterY
“baseline”	NSLayoutAttributeBaseline
“lastBaseline”	NSLayoutAttributeLastBaseline
“firstBaseline”	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*key*, *modifiers*)
Bind a key to the button.

key A single character or one of the following:

"help"
"home"
"end"
"pageup"
"pagedown"
"forwarddelete"
"leftarrow"
"rightarrow"
"uparrow"
"downarrow"

modifiers A list containing nothing or as many of the following as desired:

"command"
"control"
"option"
"shift"
"capslock"

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSButton()
Return the NSButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize (*width*, *height*)
Change the size of the object to *width* and *height*.

setImage (*imagePath=None*, *imageNamed=None*, *imageObject=None*)
Set the image in the button.

imagePath A file path to an image.

imageNamed The name of an image already loaded as an NSImage by the application.

imageObject A NSImage object.

Note: Only one of *imagePath*, *imageNamed*, *imageObject* should be set.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle (*title*)
Set the control title.

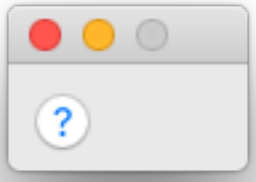
title A string representing the title.

show (*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4.5 HelpButton

class vanilla.**HelpButton** (*posSize*, *callback=None*, *page=None*, *anchor=None*)
A standard help button.



```
from vanilla import Window, HelpButton

class HelpButtonDemo:

    def __init__(self):
        self.w = Window((90, 40))
```

(continues on next page)

(continued from previous page)

```
        self.w.button = HelpButton((10, 10, 21, 20),
                                   callback=self.buttonCallback)
        self.w.open()

    def buttonCallback(self, sender):
        print("help button hit!")

HelpButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the button. The size of the button should match the standard dimensions.

Standard Dimensions	
Width	21
Height	20

callback The method to be called when the user presses the button.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

bind(*key*, *modifiers*)

Bind a key to the button.

key A single character or one of the following:

"help"
"home"
"end"
"pageup"
"pagedown"
"forwarddelete"
"leftarrow"
"rightarrow"
"uparrow"
"downarrow"

modifiers A list containing nothing or as many of the following as desired:

"command"
"control"
"option"
"shift"
"capslock"

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSButton()
Return the NSButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

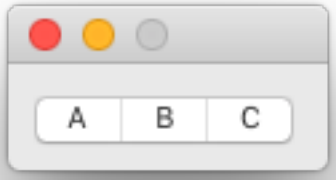
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.4.6 SegmentedButton

class vanilla.**SegmentedButton**(*posSize*, *segmentDescriptions*, *callback=None*, *selectionStyle='one'*, *sizeStyle='small'*)
A standard segmented button.




```
from vanilla import Window, SegmentedButton

class SegmentedButtonDemo:

    def __init__(self):
        self.w = Window((120, 40))
        self.w.button = SegmentedButton((10, 10, -10, 20),
            [dict(title="A"), dict(title="B"), dict(title="C")],
           callback=self.buttonCallback)
        self.w.open()

    def buttonCallback(self, sender):
        print("button hit!")

SegmentedButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the segmented button. The size of the segmented button should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	21
Small	H	18
Mini	H	15

segmentDescriptions An ordered list of dictionaries describing the segments.

width (optional)	The desired width of the segment.
title (optional)	The title of the segment.
enabled (optional)	The enabled state of the segment. The default is *True*.
imagePath (optional)	A file path to an image to display in the segment.
imageNamed (optional)	The name of an image already loaded as a NSImage by the application to display in the segment.
imageObject (optional)	A NSImage object to display in the segment.
imageTemplate (optional)	A boolean representing if the image should converted to a template image.

callback The method to be called when the user presses the segmented button.

selectionStyle The selection style in the segmented button.

one	Only one segment may be selected.
any	Any number of segments may be selected.
momentary	A segmented is only selected when tracking.

sizeStyle A string representing the desired size style of the segmented button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. **onOff** should be a boolean.

get()
Get the selected segment. If this control is set to *any* mode, the returned value will be a list of integers. Otherwise the returned value will be a single integer or *None* if no segment is selected.

getNSSegmentedButton()
Return the NSSegmentedControl that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the selected segment. If this control is set to *any* mode, *value* should be a list of integers. Otherwise *value* should be a single integer.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

title A string representing the title.

show(*onOff*)
Show or hide the object.

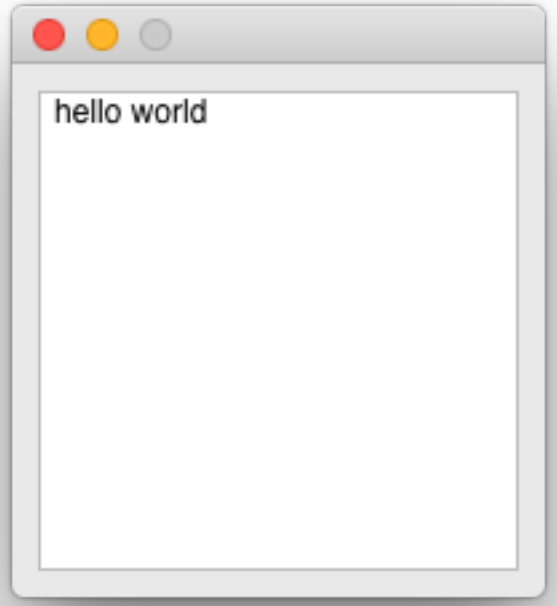
onOff A boolean value representing if the object should be shown or not.

2.5 Inputs

2.5.1 TextEditor

class vanilla.**TextEditor**(*posSize*, *text=''*, *callback=None*, *readOnly=False*, *checksSpelling=False*)

Standard long text entry control.




```
from vanilla import Window, TextEditor

class TextEditorDemo:

    def __init__(self):
        self.w = Window((200, 200))
        self.w.textEditor = TextEditor((10, 10, -10, -10),
                            text='hello world',
                            callback=self.textEditorCallback)
        self.w.open()

    def textEditorCallback(self, sender):
        print("text entry!", sender.get())

TextEditorDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the text entry control.

text The text to be displayed in the text entry control.

callback The method to be called when the user presses the text entry control.

readOnly Boolean representing if the text can be edited or not.

checksSpelling Boolean representing if spelling should be automatically checked or not.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the text entry control.

getNSScrollView()
Return the NSScrollView that this object wraps.

getNSTextView()
Return the NSTextView that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

selectAll()
Select all text in the text entry control.

set(*value*)
Set the contents of the text box.

value A string representing the contents of the text box.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.2 TextBox

class vanilla.**TextBox**(*posSize*, *text=''*, *alignment='natural'*, *selectable=False*, *sizeStyle='regular'*)
A rectangle containing static text.


```
from vanilla import Window, TextBox

class TextBoxDemo:

    def __init__(self):
        self.w = Window((100, 37))
        self.w.textBox = TextBox((10, 10, -10, 17), "A TextBox")
        self.w.open()

TextBoxDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the text box.

Standard Dimensions		
Regular	H	17
Small	H	14
Mini	H	12

text The text to be displayed in the text box. If the object is a NSAttributedString, the attributes will be used for display.

alignment A string representing the desired visual alignment of the text in the text box. The options are:

"left"	Text is aligned left.
"right"	Text is aligned right.
"center"	Text is centered.
"justified"	Text is justified.
"natural"	Follows the natural alignment of the text's script.

selectable Boolean representing if the text is selectable or not.

sizeStyle A string representing the desired size style of the button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)
Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the text box.

getNSTextField()
Return the NSTextField that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the contents of the text box.

value A string representing the contents of the text box.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

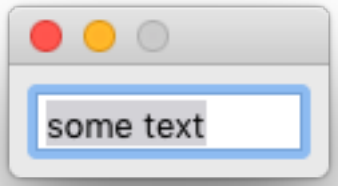
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.3 EditText

class vanilla.**EditText**(*posSize*, *text=''*, *callback=None*, *continuous=True*, *readOnly=False*, *formatter=None*, *placeholder=None*, *sizeStyle='regular'*)
Standard short text entry control.




```
from vanilla import Window, EditText

class EditTextDemo:

    def __init__(self):
        self.w = Window((120, 42))
        self.w.editText = EditText((10, 10, -10, 22),
                                   text='some text',
                                   callback=self.editTextCallback)
        self.w.open()

    def editTextCallback(self, sender):
        print("text entry!", sender.get())

EditTextDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the text entry control.

Standard Dimensions		
Regular	H	22
Small	H	19
Mini	H	16

text An object representing the contents of the text entry control. If no formatter has been assigned to the control, this should be a string. If a formatter has been assigned, this should be an object of the type that the formatter expects.

callback The method to be called when the user enters text.

continuous If *True*, the callback (if any) will be called upon each keystroke, if *False*, only call the callback when editing finishes. Default is *True*.

readOnly Boolean representing if the text can be edited or not.

formatter A NSFormatter for controlling the display and input of the text entry.

placeholder A placeholder string to be shown when the text entry control is empty.

sizeStyle A string representing the desired size style of the text entry control. The options are:

“regular”
“small”
“mini”

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
“view1”	The vanilla wrapped view for the left side of the rule.
“attribute1”	The attribute of the view for the left side of the rule. See below for options.
“relation” (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *“==”*.
“view2”	The vanilla wrapped view for the right side of the rule.
“attribute2”	The attribute of the view for the right side of the rule. See below for options.
“multiplier” (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
“constant” (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
“left”	NSLayoutAttributeLeft
“right”	NSLayoutAttributeRight
“top”	NSLayoutAttributeTop
“bottom”	NSLayoutAttributeBottom
“leading”	NSLayoutAttributeLeading
“trailing”	NSLayoutAttributeTrailing
“width”	NSLayoutAttributeWidth
“height”	NSLayoutAttributeHeight
“centerX”	NSLayoutAttributeCenterX
“centerY”	NSLayoutAttributeCenterY
“baseline”	NSLayoutAttributeBaseline
“lastBaseline”	NSLayoutAttributeLastBaseline
“firstBaseline”	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
“<=”	NSLayoutRelationLessThanOrEqual
“==”	NSLayoutRelationEqual
“>=”	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the text entry control.

If no formatter has been assigned to the control, this returns a string. If a formatter has been assigned, this returns an object which has been translated by the formatter.

getNSTextField()
Return the NSTextField that this object wraps.

getPlaceholder()
Get the placeholder string displayed in the control.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

selectAll()
Select all text in the text entry control.

set(*value*)
Set the contents of the text entry control.

value An object representing the contents of the text entry control. If no formatter has been assigned to the control, this should be a string. If a formatter has been assigned, this should be an object of the type that the formatter expects.

setPlaceholder(*value*)
Set *value* as the placeholder string to be displayed in the control. *value* must be a string.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

title A string representing the title.

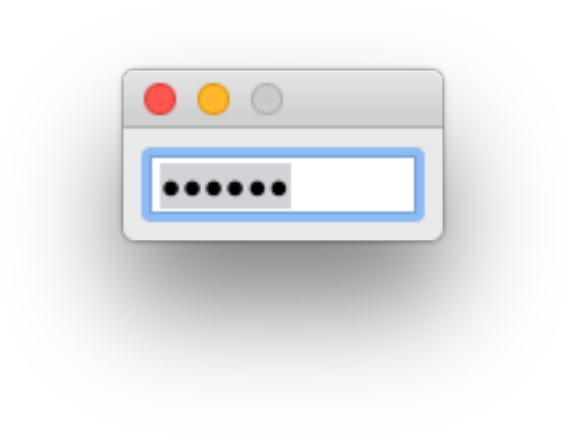
show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.4 SecureEditText

class vanilla.**SecureEditText**(*posSize*, *text=''*, *callback=None*, *continuous=True*, *readOnly=False*, *formatter=None*, *placeholder=None*, *sizeStyle='regular'*)

Standard secure text entry control.



```
from vanilla import Window, SecureEditText

class SecureEditTextDemo:

    def __init__(self):
        self.w = Window((120, 42))
        self.w.secureEditText = SecureEditText((10, 10, -10, 22),
                            text='abc123',
                            callback=self.secureEditTextCallback)
        self.w.open()

    def secureEditTextCallback(self, sender):
        print("text entry!", sender.get())

SecureEditTextDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the text entry control.

Standard Dimensions		
Regular	H	22
Small	H	19
Mini	H	16

text An object representing the contents of the text entry control. If no formatter has been assigned to the control, this should be a string. If a formatter has been assigned, this should be an object of the type that the formatter expects.

callback The method to be called when the user enters text.

continuous If *True*, the callback (if any) will be called upon each keystroke, if *False*, only call the callback when editing finishes. Default is *True*.

readOnly Boolean representing if the text can be edited or not.

formatter A NSFormatter for controlling the display and input of the text entry.

placeholder A placeholder string to be shown when the text entry control is empty.

sizeStyle A string representing the desired size style of the text entry control. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the text entry control.

If no formatter has been assigned to the control, this returns a string. If a formatter has been assigned, this returns an object which has been translated by the formatter.

getNSSecureTextField()
Return the NSSecureTextField that this object wraps.

getNSTextField()
Return the NSTextField that this object wraps.

getPlaceholder()
Get the placeholder string displayed in the control.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

selectAll()
Select all text in the text entry control.

set(*value*)
Set the contents of the text entry control.

value An object representing the contents of the text entry control. If no formatter has been assigned to the control, this should be a string. If a formatter has been assigned, this should be an object of the type that the formatter expects.

setPlaceholder(*value*)
Set *value* as the placeholder string to be displayed in the control. *value* must be a string.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

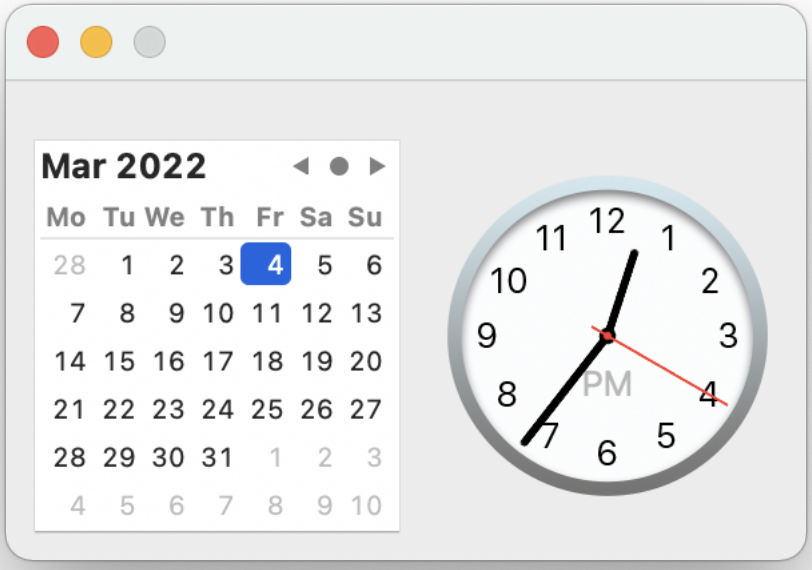
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.5 DatePicker

class vanilla.**DatePicker**(*posSize*, *date=None*, *minDate=None*, *maxDate=None*, *showStepper=True*, *mode='text'*, *timeDisplay='hourMinuteSecond'*, *dateDisplay='yearMonthDay'*, *callback=None*, *sizeStyle='regular'*)
A control for setting date objects




```
from vanilla import DatePicker, Window

class DatePickerExample:
    def __init__(self):
        self.w = Window((300, 180))
        self.w.datePicker = DatePicker(
            (10, 10, -10, -10),
```

(continues on next page)

(continued from previous page)

```
            showStepper=True,
            mode="graphical",
            timeDisplay="hourMinuteSecond",
            dateDisplay="yearMonthDay",
            callback=self.datePickerCallback,
            sizeStyle="regular",
        )

        self.w.open()

    def datePickerCallback(self, sender):
        print(sender.get())


DatePickerExample()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the date picker control.

Standard Dimensions - Text Mode		
Regular	H	22
Small	H	19
Mini	H	16

Standard Dimensions - Graphical Mode	
Calendar and Clock	227w 148h
Calendar	139w 148h
Clock	122w 123h

date A NSDate object representing the date and time that should be set in the control.

minDate A NSDate object representing the lowest date and time that can be set in the control.

maxDate A NSDate object representing the highest date and time that can be set in the control.

showStepper A boolean indicating if the thumb stepper should be shown in text mode.

mode A string representing the desired mode for the date picker control. The options are:

"text"
"graphical"

timeDisplay A string representing the desired time units that should be displayed in the date picker control. The options are:

None	Do not display time.
"hourMinute"	Display hour and minute.
"hourMinuteSecond"	Display hour, minute, second.

dateDisplay A string representing the desired date units that should be displayed in the date picker control. The options are:

None	Do not display date.
“yearMonth”	Display year and month.
“yearMonthDay”	Display year, month and day.

sizeStyle A string representing the desired size style of the date picker control. This only applies in text mode. The options are:

“regular”
“small”
“mini”

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
“view1”	The vanilla wrapped view for the left side of the rule.
“attribute1”	The attribute of the view for the left side of the rule. See below for options.
“relation” (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *“==”*.
“view2”	The vanilla wrapped view for the right side of the rule.
“attribute2”	The attribute of the view for the right side of the rule. See below for options.
“multiplier” (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
“constant” (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
“left”	NSLayoutAttributeLeft
“right”	NSLayoutAttributeRight
“top”	NSLayoutAttributeTop
“bottom”	NSLayoutAttributeBottom
“leading”	NSLayoutAttributeLeading
“trailing”	NSLayoutAttributeTrailing
“width”	NSLayoutAttributeWidth
“height”	NSLayoutAttributeHeight
“centerX”	NSLayoutAttributeCenterX
“centerY”	NSLayoutAttributeCenterY
“baseline”	NSLayoutAttributeBaseline
“lastBaseline”	NSLayoutAttributeLastBaseline
“firstBaseline”	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the date picker control.

getNSDatePicker()
Return the NSDatePicker that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the contents of the date picker control.

value A NSDate object.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

title A string representing the title.

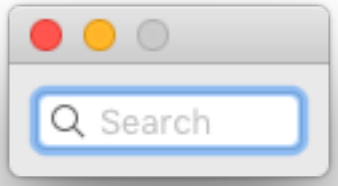
show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.6 SearchBox

class vanilla.**SearchBox**(*posSize*, *text=''*, *callback=None*, *formatter=None*, *placeholder=None*, *sizeStyle='regular'*)

A text entry field similar to the search field in Safari.




```
from vanilla import Window, SearchBox

class SearchBoxDemo:

    def __init__(self):
        self.w = Window((120, 42))
        self.w.searchBox = SearchBox((10, 10, -10, 22),
                                     callback=self.searchBoxCallback)
        self.w.open()

    def searchBoxCallback(self, sender):
        print("search box entry!", sender.get())

SearchBoxDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the search box.

Standard Dimensions		
Regular	H	22
Small	H	19
Mini	H	15

text The text to be displayed in the search box.

callback The method to be called when the user presses the search box.

formatter A NSFormatter for controlling the display and input of the text entry.

placeholder A placeholder string to be shown when the text entry control is empty.

sizeStyle A string representing the desired size style of the search box. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)
Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the contents of the search box.

getNSSearchField()
Return the NSSearchField that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the contents of the search box.

value A string representing the contents of the search box.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

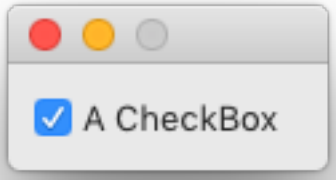
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.7 CheckBox

class vanilla.**CheckBox**(*posSize*, *title*, *callback=None*, *value=False*, *sizeStyle='regular'*)
A standard check box.




```
from vanilla import Window, CheckBox

class CheckBoxDemo:

    def __init__(self):
        self.w = Window((120, 40))
        self.w.checkBox = CheckBox((10, 10, -10, 20), "A CheckBox",
                           callback=self.checkBoxCallback, value=True)
        self.w.open()

    def checkBoxCallback(self, sender):
        print("check box state change!", sender.get())

CheckBoxDemo()
```

posSize Tuple of form (left, top, width, height) or *"auto"* representing the position and size of the check box. The size of the check box should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	22
Small	H	18
Mini	H	10

title The text to be displayed next to the check box. Pass *None* if no title is desired.

callback The method to be called when the user changes the state of the check box.

value A boolean representing the state of the check box.

sizeStyle A string representing the desired size style of the check box. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language

- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. **onOff** should be a boolean.

get()
Get the state of the check box.

getNSButton()
Return the NSButton that this object wraps.

This is currently not implemented for CheckBox.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the state of the check box.

value A boolean representing the state of the check box.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

title A string representing the title.

show(*onOff*)
Show or hide the object.

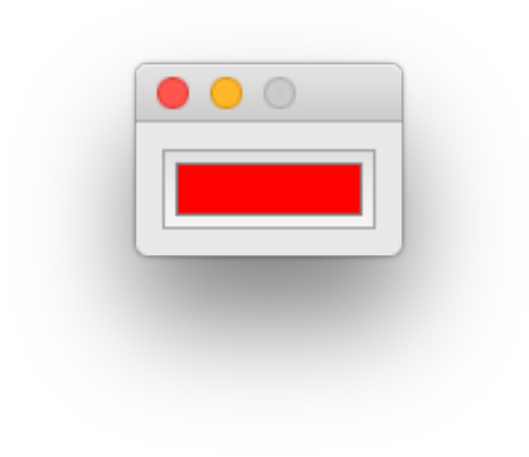
onOff A boolean value representing if the object should be shown or not.

toggle()
Toggle the state of the check box.

If the check box is on, turn it off. If the check box is off, turn it on.

2.5.8 ColorWell

class vanilla.**ColorWell**(*posSize*, *callback=None*, *color=None*, *colorWellStyle=None*)
A control that allows for showing and choosing a color value.



ColorWell objects handle NSColor objects.

```
from AppKit import NSColor
from vanilla import Window, ColorWell

class ColorWellDemo:

    def __init__(self):
        self.w = Window((100, 50))
        self.w.colorWell = ColorWell((10, 10, -10, -10),
                            callback=self.colorWellEdit,
                            color=NSColor.redColor())
        self.w.open()

    def colorWellEdit(self, sender):
        print("color well edit!", sender.get())

ColorWellDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the color well.

callback The method to be caled when the user selects a new color.

color A NSColor object. If *None* is given, the color shown will be white.

colorWellStyle A string representing the desired color well style. The options are:

"default"
"minimal"
"expanded"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the NSColor object representing the current color in the color well.

getNSColorWell()
Return the NSColorWell that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*color*)
Set the color in the color well.

color A NSColor object representing the color to be displayed in the color well.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

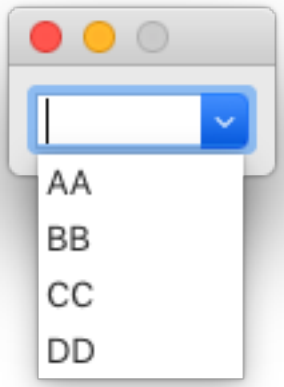
animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.9 ComboBox

class vanilla.**ComboBox**(*posSize*, *items*, *completes=True*, *continuous=False*, *callback=None*, *formatter=None*, *sizeStyle='regular'*)
A text entry control that allows direct text entry or selection for a list of options.




```
from vanilla import Window, ComboBox

class ComboBoxDemo:

    def __init__(self):
        self.w = Window((100, 41))
        self.w.comboBox = ComboBox((10, 10, -10, 21),
                                   ["AA", "BB", "CC", "DD"],
                                   callback=self.comboBoxCallback)
```

(continues on next page)

(continued from previous page)

```
        self.w.open()

    def comboBoxCallback(self, sender):
        print("combo box entry!", sender.get())

ComboBoxDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the combo box control. The size of the combo box should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	21
Small	H	17
Mini	H	14

items The items to be displayed in the combo box.

completes Boolean representing if the combo box auto completes entered text.

continuous If *True*, the callback (if any) will be called upon each keystroke, if *False*, only call the callback when editing finishes or after item selection. Default is *False*.

callback The method to be called when the user enters text.

formatter A NSFormatter for controlling the display and input of the text entry.

sizeStyle A string representing the desired size style of the combo box. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the text entered in the combo box.

getItems()
Get the items in the combo box as a list.

getNSComboBox()
Return the NSComboBox that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set (*value*)
Set the text in the text field of the combo box.

value A string to set in the combo box.

setItems (*items*)
Set the items in the combo box list.

items A list of strings to set in the combo box list.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle (*title*)
Set the control title.

title A string representing the title.

show (*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.10 PopUpButton

class vanilla.**PopUpButton** (*posSize*, *items*, *callback=None*, *sizeStyle='regular'*)
A button which, when selected, displays a list of items for the user to choose from.


```
from vanilla import Window, PopUpButton

class PopUpButtonDemo:

    def __init__(self):
        self.w = Window((100, 40))
        self.w.popUpButton = PopUpButton((10, 10, -10, 20),
                                         ["A", "B", "C"],
                                         callback=self.popUpButtonCallback)
        self.w.open()
```

(continues on next page)

(continued from previous page)

```
    def popUpButtonCallback(self, sender):
        print("pop up button selection!", sender.get())

PopUpButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the pop up button. The size of the button should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	20
Small	H	17
Mini	H	15

items A list of items to appear in the pop up list.

callback The method to be called when the user selects an item in the pop up list.

sizeStyle A string representing the desired size style of the pop up button. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the index selected item in the pop up list.

getItem()
Get the selected item title in the pop up list.

getItems()
Get the list of items that appear in the pop up list.

getNSPopUpButton()
Return the NSPopUpButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize (*width*, *height*)
Change the size of the object to *width* and *height*.

set (*value*)
Set the index of the selected item in the pop up list.

setItem (*item*)
Set the selected item title in the pop up list.

setItems (*items*)
Set the items to appear in the pop up list.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle (*title*)
Set the control title.

title A string representing the title.

show (*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.11 ActionButton

class vanilla.**ActionButton** (*posSize*, *items*, *sizeStyle='regular'*, *bordered=True*)
An action button with a menu.

```
from vanilla import Window, ActionButton

class ActionPopUpButtonDemo:

    def __init__(self):
        self.w = Window((100, 40))
        items = [
                dict(title="first", callback=self.firstCallback),
                dict(title="second", callback=self.secondCallback),
```

(continues on next page)

(continued from previous page)

```
                dict(title="third", items=[
                        dict(title="sub first", callback=self.subFirstCallback)
                    ])
                ]
        self.w.actionPopUpButton = ActionButton((10, 10, 30, 20), items)
        self.w.open()

    def firstCallback(self, sender):
        print("first")

    def secondCallback(self, sender):
        print("second")

    def subFirstCallback(self, sender):
        print("sub first")

ActionPopUpButtonDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the pop up button. The size of the button should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	20
Small	H	17
Mini	H	15

items A list of items to appear in the pop up list as dictionaries. Optionally an item could be a NSMenuItem. When an item is set to `----`, it will be a menu item separator.

"title"	The title of the item.
"callback"	The callback of the item.
"items"	Each item could have sub menus, as list of dictionaries with the same format.

sizeStyle A string representing the desired size style of the pop up button. The options are:

"regular"
"small"
"mini"

bordered Boolean representing if the button should be bordered.

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the index selected item in the pop up list.

getItem()
Get the selected item title in the pop up list.

getItems()
Get the list of items that appear in the pop up list.

getNSPopUpButton()
Return the NSPopUpButton that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the index of the selected item in the pop up list.

setItem(*item*)
Set the selected item title in the pop up list.

setItems(*items*)
Set the items to appear in the pop up list.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTitle(*title*)
Set the control title.

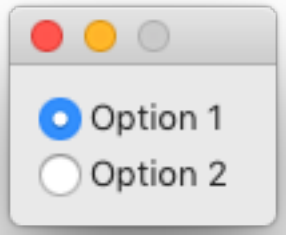
title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.12 RadioGroup

class vanilla.**RadioGroup**(*posSize*, *titles*, *isVertical=True*, *callback=None*, *sizeStyle='regular'*)
A collection of radio buttons.



Note: This should be used only for frame layout.

```
from vanilla import Window, RadioGroup

class RadioGroupDemo:

    def __init__(self):
        self.w = Window((100, 60))
        self.w.radioGroup = RadioGroup((10, 10, -10, 40),
                                       ["Option 1", "Option 2"],
                                       callback=self.radioGroupCallback)
        self.w.radioGroup.set(0)
        self.w.open()

    def radioGroupCallback(self, sender):
        print("radio group edit!", sender.get())

RadioGroupDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the radio group.

titles A list of titles to be shown next to the radio buttons.

isVertical Boolean representing if the radio group is vertical or horizontal.

callback The method to be caled when a radio button is selected.

sizeStyle A string representing the desired size style of the radio group. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language

- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

enableRadioButton(*index*, *onOff*=*True*)
Enable or disable a RadioGroup button specified by its index.

get()
Get the index of the selected radio button.

getNSMatrix()
Return the NSMatrix that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*index*)
Set the index of the selected radio button.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

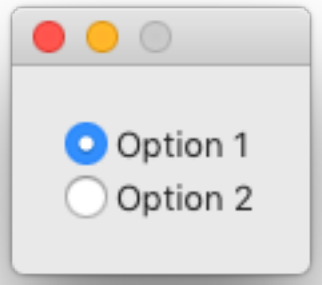
setTitle(*title*)
Set the control title.

title A string representing the title.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

class vanilla.**VerticalRadioGroup**(*posSize*, *titles*, *callback=None*, *sizeStyle='regular'*)
A vertical collection of radio buttons.



```
from vanilla import Window, VerticalRadioGroup

class VerticalRadioGroupDemo:

    def __init__(self):
        self.w = Window((100, 100))
        self.w.radioGroup = VerticalRadioGroup(
            "auto",
            ["Option 1", "Option 2"],
            callback=self.radioGroupCallback
        )
        self.w.radioGroup.set(0)
        rules = [
            "H:|-[radioGroup]-|",
            "V:|-[radioGroup(==%d)]-|" % self.w.radioGroup.getFittingHeight()
        ]
        self.w.addAutoPosSizeRules(rules)
        self.w.open()

    def radioGroupCallback(self, sender):
        print("radio group edit!", sender.get())

VerticalRadioGroupDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the radio group.

titles A list of titles to be shown next to the radio buttons.

callback The method to be caled when a radio button is selected.

sizeStyle A string representing the desired size style of the radio group. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

appendView(*view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Append a view.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the index of the selected radio button.

getFittingHeight()
Get the fitting height for all buttons in the group.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

insertView(*index*, *view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Insert a view.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

removeView(*view*)
Remove a view.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*index*)
Set the index of the selected radio button.

setEdgeInsets(*value*)
Set the edge insets.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

class vanilla.**HorizontalRadioGroup**(*posSize*, *titles*, *callback=None*, *sizeStyle='regular'*)
A horizontal collection of radio buttons.

```
from vanilla import Window, HorizontalRadioGroup

class RadioGroupDemo:

    def __init__(self):
        self.w = Window((300, 100))
        self.w.radioGroup = HorizontalRadioGroup(
            "auto",
            ["Option 1", "Option 2"],
            callback=self.radioGroupCallback
        )
        self.w.radioGroup.set(0)
        rules = [
            "H:|-[radioGroup]-|",
            "V:|-[radioGroup(==%d)]-|" % self.w.radioGroup.getFittingHeight()
        ]
        self.w.addAutoPosSizeRules(rules)
        self.w.open()

    def radioGroupCallback(self, sender):
        print("radio group edit!", sender.get())

RadioGroupDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the radio group.

titles A list of titles to be shown next to the radio buttons.

callback The method to be caled when a radio button is selected.

sizeStyle A string representing the desired size style of the radio group. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

appendView(*view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)

Append a view.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the index of the selected radio button.

getFittingHeight()
Get the fitting height for all buttons in the group.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

insertView(*index*, *view*, *width=None*, *height=None*, *spacing=None*, *gravity='center'*)
Insert a view.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

removeView(*view*)
Remove a view.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*index*)
Set the index of the selected radio button.

setEdgeInsets(*value*)
Set the edge insets.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.5.13 Slider

class vanilla.**Slider**(*posSize*, *minValue=0*, *maxValue=100*, *value=50*, *tickMarkCount=None*, *stopOnTickMarks=False*, *continuous=True*, *callback=None*, *sizeStyle='regular'*)
A standard slider control. Sliders can be vertical or horizontal and they can show tick marks or not.

```
from vanilla import Window, Slider

class SliderDemo:

    def __init__(self):
        self.w = Window((200, 43))
        self.w.slider = Slider((10, 10, -10, 23),
                               tickMarkCount=10,
                               callback=self.sliderCallback)
        self.w.open()

    def sliderCallback(self, sender):
        print("slider edit!", sender.get())

SliderDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the slider. The size of the slider should match the appropriate value for the given *sizeStyle*.

Standard Dimensions				
without ticks				
Regular	W	15	H	15
Small	W	12	H	11
Mini	W	10	H	10
with ticks				
Regular	W	24	H	23
Small	W	17	H	17
Mini	W	16	H	16

minValue The minimum value allowed by the slider.

maxValue The maximum value allowed by the slider.

value The initial value of the slider.

tickMarkCount The number of tick marcks to be displayed on the slider. If *None* is given, no tick marks will be displayed.

stopOnTickMarks Boolean representing if the slider knob should only stop on the tick marks.

continuous Boolean representing if the assigned callback should be called during slider editing. If *False* is given, the callback will be called after the editing has finished.

callback The method to be called when the slider has been edited.

sizeStyle A string representing the desired size style of the slider. The options are:

"regular"
"small"
"mini"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the value of the slider.

getNSSlider()
Return the NSSlider that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

getTitle()
Get the control title.

isEnabled()
Return a bool indicating if the object is enable or not.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set(*value*)
Set the value of the slider.

setMaxValue(*value*)
Set the maximum value allowed by the slider.

setMinValue(*value*)
Set the minimum value allowed by the slider.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

setTickMarkCount(*value*)
Set the number of tick marks on the slider.

setTickMarkPosition(*value*)
Set the position of the tick marks on the slider.

For vertical sliders, the options are:

"left"
"right"

For horizontal sliders, the options are:

"top"
"bottom"

setTitle(*title*)
Set the control title.

title A string representing the title.

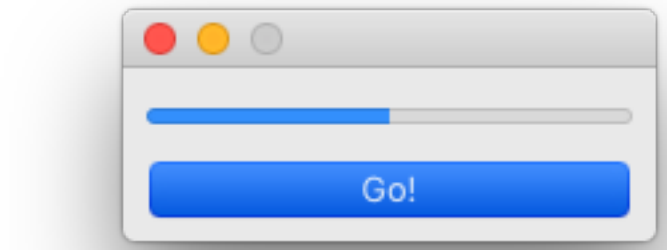
show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

2.6 Progress Indicators

2.6.1 ProgressBar

class vanilla.**ProgressBar**(*posSize*, *minValue=0*, *maxValue=100*, *isIndeterminate=False*, *sizeStyle='regular'*, *progressStyle='bar'*)
A standard progress bar.




```
from vanilla import Window, ProgressBar, Button

class ProgressBarDemo:

    def __init__(self):
        self.w = Window((200, 65))
        self.w.bar = ProgressBar((10, 10, -10, 16))
        self.w.button = Button((10, 35, -10, 20), "Go!",
                               callback=self.showProgress)
```

(continues on next page)

(continued from previous page)

```
        self.w.open()

    def showProgress(self, sender):
        import time
        self.w.bar.set(0)
        for i in range(10):
            self.w.bar.increment(10)
            time.sleep(.2)

ProgressBarDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the progress bar. The height of the progress bar should match the appropriate value for the given *sizeStyle*.

Standard Dimensions		
Regular	H	16
Small	H	10

minValue The minimum value of the progress bar.

maxValue The maximum value of the progress bar.

isIndeterminate Boolean representing if the progress bar is indeterminate. Determinate progress bars show how much of the task has been completed. Indeterminate progress bars simply show that the application is busy.

sizeStyle A string representing the desired size style of the pregress bar. The options are:

"regular"
"small"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

get()
Get the current value of the progress bar.

Only available in determinate progress bars.

getNSProgressIndicator()
Return the NSProgressIndicator that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

increment(*value=1*)
Increment the progress bar by **value**.

Only available in determinate progress bars.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

set (*value*)
Set the value of the progress bar to **value**.

Only available in determinate progress bars.

setPosSize (*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show (*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

start ()
Start the animation.

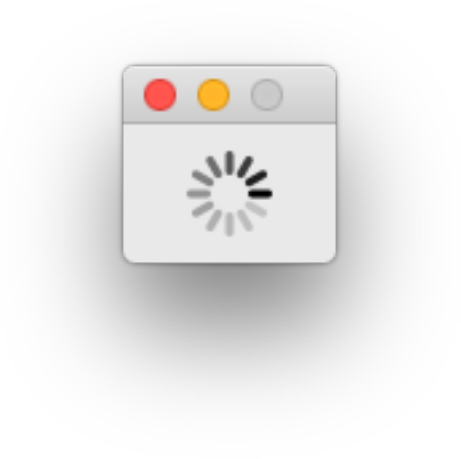
Only available in indeterminate progress bars.

stop ()
Stop the animation.

Only available in indeterminate progress bars.

2.6.2 ProgressSpinner

class vanilla.**ProgressSpinner** (*posSize*, *displayWhenStopped=False*, *sizeStyle='regular'*)
An animated, spinning progress indicator.



```
from vanilla import Window, ProgressSpinner

class ProgressSpinnerDemo:

    def __init__(self):
        self.w = Window((80, 52))
        self.w.spinner = ProgressSpinner((24, 10, 32, 32),
                                displayWhenStopped=True)
        self.w.spinner.start()
        self.w.open()
```

(continues on next page)

(continued from previous page)

```
ProgressSpinnerDemo()
```

posSize Tuple of form *(left, top, width, height)* or *"auto"* representing the position and size of the spinner. The size of the spinner should match the appropriate value for the given *sizeStyle*.

Standard Dimensions				
Regular	W	32	H	32
Small	W	16	H	16

displayWhenStopped Boolean representing if the spinner should be displayed when it is not spinning.

sizeStyle A string representing the desired size style of the spinner. The options are:

"regular"
"small"

addAutoPosSizeRules(*rules*, *metrics=None*)

Add auto layout rules for controls/view in this view.

rules must be a list of rule definitions. Rule definitions may take two forms:

- strings that follow the Visual Format Language
- dictionaries with the following key/value pairs:

key	value
"view1"	The vanilla wrapped view for the left side of the rule.
"attribute1"	The attribute of the view for the left side of the rule. See below for options.
"relation" (optional)	The relationship between the left side of the rule and the right side of the rule. See below for options. The default value is *"=="*.
"view2"	The vanilla wrapped view for the right side of the rule.
"attribute2"	The attribute of the view for the right side of the rule. See below for options.
"multiplier" (optional)	The constant multiplied with the attribute on the right side of the rule as part of getting the modified attribute. The default value is *1*.
"constant" (optional)	The constant added to the multiplied attribute value on the right side of the rule to yield the final modified attribute. The default value is *0*.

The *attribute1* and *attribute2* options are:

value	AppKit equivalent
"left"	NSLayoutAttributeLeft
"right"	NSLayoutAttributeRight
"top"	NSLayoutAttributeTop
"bottom"	NSLayoutAttributeBottom
"leading"	NSLayoutAttributeLeading
"trailing"	NSLayoutAttributeTrailing
"width"	NSLayoutAttributeWidth
"height"	NSLayoutAttributeHeight
"centerX"	NSLayoutAttributeCenterX
"centerY"	NSLayoutAttributeCenterY
"baseline"	NSLayoutAttributeBaseline
"lastBaseline"	NSLayoutAttributeLastBaseline
"firstBaseline"	NSLayoutAttributeFirstBaseline

Refer to the NSLayoutAttribute documentation for the information about what each of these do.

The *relation* options are:

value	AppKit equivalent
"<="	NSLayoutRelationLessThanOrEqual
"=="	NSLayoutRelationEqual
">="	NSLayoutRelationGreaterThanOrEqual

Refer to the NSLayoutRelation documentation for the information about what each of these do.

metrics may be either *None* or a dict containing key value pairs representing metrics keywords used in the rules defined with strings.

enable(*onOff*)
Enable or disable the object. *onOff* should be a boolean.

getNSProgressIndicator()
Return the NSProgressIndicator that this object wraps.

getPosSize()
The position and size of the object as a tuple of form *(left, top, width, height)*.

isVisible()
Return a bool indicating if the object is visible or not.

move(*x*, *y*)
Move the object by *x* units and *y* units.

resize(*width*, *height*)
Change the size of the object to *width* and *height*.

setPosSize(*posSize*, *animate=False*)
Set the position and size of the object.

posSize A tuple of form *(left, top, width, height)*.

animate A boolean flag telling to animate the transition. Off by default.

show(*onOff*)
Show or hide the object.

onOff A boolean value representing if the object should be shown or not.

start()
Start the animation.

stop()
Stop the animation.

CHAPTER 3

Indices and tables

- genindex

Python Module Index

v

vanilla, 9
vanilla2, 60

Index

A

ActionButton (*class in vanilla*), 131
addAutoPosSizeRules() (*vanilla.ActionButton method*), 132
addAutoPosSizeRules() (*vanilla.Box method*), 37
addAutoPosSizeRules() (*vanilla.Button method*), 81
addAutoPosSizeRules() (*vanilla.CheckBox method*), 120
addAutoPosSizeRules() (*vanilla.ColorWell method*), 123
addAutoPosSizeRules() (*vanilla.ComboBox method*), 126
addAutoPosSizeRules() (*vanilla.DatePicker method*), 115
addAutoPosSizeRules() (*vanilla.Drawer method*), 22
addAutoPosSizeRules() (*vanilla.EditText method*), 108
addAutoPosSizeRules() (*vanilla.GradientButton method*), 92
addAutoPosSizeRules() (*vanilla.GridView method*), 50
addAutoPosSizeRules() (*vanilla.Group method*), 24
addAutoPosSizeRules() (*vanilla.HelpButton method*), 95
addAutoPosSizeRules() (*vanilla.HorizontalLine method*), 42
addAutoPosSizeRules() (*vanilla.HorizontalRadioGroup method*), 140
addAutoPosSizeRules() (*vanilla.HorizontalStackView method*), 54
addAutoPosSizeRules() (*vanilla.ImageButton method*), 88
addAutoPosSizeRules() (*vanilla.ImageView method*), 75
addAutoPosSizeRules() (*vanilla.LevelIndicator method*), 78
addAutoPosSizeRules() (*vanilla.List method*), 64
addAutoPosSizeRules() (*vanilla.Popover method*), 40
addAutoPosSizeRules() (*vanilla.PopUpButton method*), 129
addAutoPosSizeRules() (*vanilla.ProgressBar method*), 147
addAutoPosSizeRules() (*vanilla.ProgressSpinner method*), 150
addAutoPosSizeRules() (*vanilla.RadioGroup method*), 135
addAutoPosSizeRules() (*vanilla.ScrollView method*), 31
addAutoPosSizeRules() (*vanilla.SearchBox method*), 118
addAutoPosSizeRules() (*vanilla.SecureEditText method*), 111
addAutoPosSizeRules() (*vanilla.SegmentedButton method*), 99
addAutoPosSizeRules() (*vanilla.Slider method*), 144
addAutoPosSizeRules() (*vanilla.SplitView method*), 34
addAutoPosSizeRules() (*vanilla.SquareButton method*), 84
addAutoPosSizeRules() (*vanilla.Tabs method*), 27
addAutoPosSizeRules() (*vanilla.TextBox method*), 104
addAutoPosSizeRules() (*vanilla.TextEditor method*), 102
addAutoPosSizeRules() (*vanilla.VerticalLine method*), 45
addAutoPosSizeRules() (*vanilla.VerticalRadioGroup method*), 138
addAutoPosSizeRules() (*vanilla.VerticalStackView method*), 58
addToolbar() (*vanilla.FloatingWindow method*), 16
addToolbar() (*vanilla.Window method*), 12
appendColumn() (*vanilla.GridView method*), 51

appendRow() (*vanilla.GridView method*), 51
appendView() (*vanilla.HorizontalRadioGroup method*), 141
appendView() (*vanilla.HorizontalStackView method*), 55
appendView() (*vanilla.VerticalRadioGroup method*), 139
appendView() (*vanilla.VerticalStackView method*), 59
assignToDocument() (*vanilla.FloatingWindow method*), 15
assignToDocument() (*vanilla.Window method*), 11

B

bind() (*vanilla.Button method*), 82
bind() (*vanilla.FloatingWindow method*), 16
bind() (*vanilla.GradientButton method*), 93
bind() (*vanilla.HelpButton method*), 96
bind() (*vanilla.ImageButton method*), 89
bind() (*vanilla.Popover method*), 41
bind() (*vanilla.SquareButton method*), 85
bind() (*vanilla.Window method*), 11
Box (*class in vanilla*), 36
Button (*class in vanilla*), 80

C

center() (*vanilla.FloatingWindow method*), 15
center() (*vanilla.Sheet method*), 20
center() (*vanilla.Window method*), 10
CheckBox (*class in vanilla*), 119
CheckBoxListCell() (*in module vanilla*), 66
close() (*vanilla.Drawer method*), 23
close() (*vanilla.FloatingWindow method*), 15
close() (*vanilla.Popover method*), 41
close() (*vanilla.Sheet method*), 20
close() (*vanilla.Window method*), 10
ColorWell (*class in vanilla*), 122
columnIsVisible() (*vanilla.GridView method*), 51
ComboBox (*class in vanilla*), 125

D

DatePicker (*class in vanilla*), 113
Drawer (*class in vanilla*), 20

E

EditText (*class in vanilla*), 106
enable() (*vanilla.ActionButton method*), 133
enable() (*vanilla.Box method*), 38
enable() (*vanilla.Button method*), 82
enable() (*vanilla.CheckBox method*), 121
enable() (*vanilla.ColorWell method*), 124
enable() (*vanilla.ComboBox method*), 127
enable() (*vanilla.DatePicker method*), 116
enable() (*vanilla.Drawer method*), 23
enable() (*vanilla.EditText method*), 109
enable() (*vanilla.GradientButton method*), 93
enable() (*vanilla.GridView method*), 51
enable() (*vanilla.Group method*), 25
enable() (*vanilla.HelpButton method*), 96
enable() (*vanilla.HorizontalLine method*), 43
enable() (*vanilla.HorizontalRadioGroup method*), 141
enable() (*vanilla.HorizontalStackView method*), 55
enable() (*vanilla.ImageButton method*), 89
enable() (*vanilla.ImageView method*), 76
enable() (*vanilla.LevelIndicator method*), 79
enable() (*vanilla.List method*), 65
enable() (*vanilla.Popover method*), 41
enable() (*vanilla.PopUpButton method*), 130
enable() (*vanilla.ProgressBar method*), 148
enable() (*vanilla.ProgressSpinner method*), 151
enable() (*vanilla.RadioGroup method*), 136
enable() (*vanilla.ScrollView method*), 32
enable() (*vanilla.SearchBox method*), 118
enable() (*vanilla.SecureEditText method*), 112
enable() (*vanilla.SegmentedButton method*), 100
enable() (*vanilla.Slider method*), 145
enable() (*vanilla.SplitView method*), 35
enable() (*vanilla.SquareButton method*), 86
enable() (*vanilla.Tabs method*), 28
enable() (*vanilla.TextBox method*), 105
enable() (*vanilla.TextEditor method*), 103
enable() (*vanilla.VerticalLine method*), 46
enable() (*vanilla.VerticalRadioGroup method*), 139
enable() (*vanilla.VerticalStackView method*), 59
enableRadioButton() (*vanilla.RadioGroup method*), 136

F

FloatingWindow (*class in vanilla*), 14

G

get() (*vanilla.ActionButton method*), 133
get() (*vanilla.CheckBox method*), 121
get() (*vanilla.ColorWell method*), 124
get() (*vanilla.ComboBox method*), 127
get() (*vanilla.DatePicker method*), 116
get() (*vanilla.EditText method*), 109
get() (*vanilla.HorizontalRadioGroup method*), 142
get() (*vanilla.LevelIndicator method*), 79
get() (*vanilla.List method*), 65
get() (*vanilla.PopUpButton method*), 130
get() (*vanilla.ProgressBar method*), 148
get() (*vanilla.RadioGroup method*), 136
get() (*vanilla.SearchBox method*), 118
get() (*vanilla.SecureEditText method*), 112
get() (*vanilla.SegmentedButton method*), 100
get() (*vanilla.Slider method*), 145
get() (*vanilla.Tabs method*), 28

get() (*vanilla.TextBox method*), 105
get() (*vanilla.TextEditor method*), 103
get() (*vanilla.VerticalRadioGroup method*), 139
getColumnCount() (*vanilla.GridView method*), 51
getCriticalValue() (*vanilla.LevelIndicator method*), 79
getDropItemValues() (*vanilla.Group method*), 25
getFittingHeight() (*vanilla.HorizontalRadioGroup method*), 142
getFittingHeight() (*vanilla.VerticalRadioGroup method*), 139
getItem() (*vanilla.ActionButton method*), 133
getItem() (*vanilla.PopUpButton method*), 130
getItems() (*vanilla.ActionButton method*), 133
getItems() (*vanilla.ComboBox method*), 127
getItems() (*vanilla.PopUpButton method*), 130
getMaxValue() (*vanilla.LevelIndicator method*), 79
getMinValue() (*vanilla.LevelIndicator method*), 79
getNSBox() (*vanilla.Box method*), 38
getNSBox() (*vanilla.HorizontalLine method*), 43
getNSBox() (*vanilla.VerticalLine method*), 46
getNSButton() (*vanilla.Button method*), 83
getNSButton() (*vanilla.CheckBox method*), 121
getNSButton() (*vanilla.GradientButton method*), 93
getNSButton() (*vanilla.HelpButton method*), 97
getNSButton() (*vanilla.ImageButton method*), 89
getNSButton() (*vanilla.SquareButton method*), 86
getNSColorWell() (*vanilla.ColorWell method*), 124
getNSComboBox() (*vanilla.ComboBox method*), 127
getNSDatePicker() (*vanilla.DatePicker method*), 116
getNSDrawer() (*vanilla.Drawer method*), 23
getNSImageView() (*vanilla.ImageView method*), 76
getNSLevelIndicator() (*vanilla.LevelIndicator method*), 79
getNSMatrix() (*vanilla.RadioGroup method*), 136
getNSPopUpButton() (*vanilla.ActionButton method*), 133
getNSPopUpButton() (*vanilla.PopUpButton method*), 130
getNSProgressIndicator() (*vanilla.ProgressBar method*), 148
getNSProgressIndicator() (*vanilla.ProgressSpinner method*), 151
getNSScrollView() (*vanilla.List method*), 65
getNSScrollView() (*vanilla.ScrollView method*), 32
getNSScrollView() (*vanilla.TextEditor method*), 103
getNSSearchField() (*vanilla.SearchBox method*), 119
getNSSecureTextField() (*vanilla.SecureEditText method*), 112
getNSSegmentedButton() (*vanilla.SegmentedButton method*), 100
getNSSlider() (*vanilla.Slider method*), 145
getNSSplitView() (*vanilla.SplitView method*), 35
getNSTableView() (*vanilla.List method*), 65
getNSTabView() (*vanilla.Tabs method*), 28
getNSTextField() (*vanilla.EditText method*), 109
getNSTextField() (*vanilla.SecureEditText method*), 112
getNSTextField() (*vanilla.TextBox method*), 105
getNSTextView() (*vanilla.TextEditor method*), 103
getNSView() (*vanilla.Group method*), 25
getNSVisualEffectView() (*vanilla.Group method*), 25
getNSWindow() (*vanilla.FloatingWindow method*), 15
getNSWindow() (*vanilla.Sheet method*), 20
getNSWindow() (*vanilla.Window method*), 10
getNSWindowController() (*vanilla.FloatingWindow method*), 15
getNSWindowController() (*vanilla.Sheet method*), 20
getNSWindowController() (*vanilla.Window method*), 10
getPlaceholder() (*vanilla.EditText method*), 109
getPlaceholder() (*vanilla.SecureEditText method*), 112
getPosSize() (*vanilla.ActionButton method*), 134
getPosSize() (*vanilla.Box method*), 38
getPosSize() (*vanilla.Button method*), 83
getPosSize() (*vanilla.CheckBox method*), 122
getPosSize() (*vanilla.ColorWell method*), 124
getPosSize() (*vanilla.ComboBox method*), 127
getPosSize() (*vanilla.DatePicker method*), 116
getPosSize() (*vanilla.Drawer method*), 23
getPosSize() (*vanilla.EditText method*), 109
getPosSize() (*vanilla.FloatingWindow method*), 15
getPosSize() (*vanilla.GradientButton method*), 93
getPosSize() (*vanilla.GridView method*), 51
getPosSize() (*vanilla.Group method*), 26
getPosSize() (*vanilla.HelpButton method*), 97
getPosSize() (*vanilla.HorizontalLine method*), 43
getPosSize() (*vanilla.HorizontalRadioGroup method*), 142
getPosSize() (*vanilla.HorizontalStackView method*), 55
getPosSize() (*vanilla.ImageButton method*), 89
getPosSize() (*vanilla.ImageView method*), 76
getPosSize() (*vanilla.LevelIndicator method*), 79
getPosSize() (*vanilla.List method*), 65
getPosSize() (*vanilla.Popover method*), 41
getPosSize() (*vanilla.PopUpButton method*), 130
getPosSize() (*vanilla.ProgressBar method*), 148
getPosSize() (*vanilla.ProgressSpinner method*), 151
getPosSize() (*vanilla.RadioGroup method*), 137

getPosSize() (*vanilla.ScrollView method*), 32
getPosSize() (*vanilla.SearchBox method*), 119
getPosSize() (*vanilla.SecureEditText method*), 112
getPosSize() (*vanilla.SegmentedButton method*), 100
getPosSize() (*vanilla.Sheet method*), 20
getPosSize() (*vanilla.Slider method*), 145
getPosSize() (*vanilla.SplitView method*), 35
getPosSize() (*vanilla.SquareButton method*), 86
getPosSize() (*vanilla.Tabs method*), 28
getPosSize() (*vanilla.TextBox method*), 106
getPosSize() (*vanilla.TextEditor method*), 103
getPosSize() (*vanilla.VerticalLine method*), 46
getPosSize() (*vanilla.VerticalRadioGroup method*), 139
getPosSize() (*vanilla.VerticalStackView method*), 59
getPosSize() (*vanilla.Window method*), 10
getRowCount() (*vanilla.GridView method*), 51
getSelection() (*vanilla.List method*), 66
getTitle() (*vanilla.ActionButton method*), 134
getTitle() (*vanilla.Box method*), 38
getTitle() (*vanilla.Button method*), 83
getTitle() (*vanilla.CheckBox method*), 122
getTitle() (*vanilla.ComboBox method*), 127
getTitle() (*vanilla.DatePicker method*), 116
getTitle() (*vanilla.EditText method*), 109
getTitle() (*vanilla.FloatingWindow method*), 15
getTitle() (*vanilla.GradientButton method*), 93
getTitle() (*vanilla.HelpButton method*), 97
getTitle() (*vanilla.HorizontalLine method*), 43
getTitle() (*vanilla.ImageButton method*), 89
getTitle() (*vanilla.LevelIndicator method*), 79
getTitle() (*vanilla.PopUpButton method*), 130
getTitle() (*vanilla.RadioGroup method*), 137
getTitle() (*vanilla.SearchBox method*), 119
getTitle() (*vanilla.SecureEditText method*), 112
getTitle() (*vanilla.SegmentedButton method*), 100
getTitle() (*vanilla.Sheet method*), 20
getTitle() (*vanilla.Slider method*), 145
getTitle() (*vanilla.SquareButton method*), 86
getTitle() (*vanilla.TextBox method*), 106
getTitle() (*vanilla.VerticalLine method*), 46
getTitle() (*vanilla.Window method*), 11
getWarningValue() (*vanilla.LevelIndicator method*), 79
GradientButton (*class in vanilla*), 90
GridView (*class in vanilla*), 47
Group (*class in vanilla*), 24

H

HelpButton (*class in vanilla*), 94
hide() (*vanilla.FloatingWindow method*), 15
hide() (*vanilla.Sheet method*), 20
hide() (*vanilla.Window method*), 11
HorizontalLine (*class in vanilla*), 42
HorizontalRadioGroup (*class in vanilla*), 140
HorizontalStackView (*class in vanilla*), 52
HUDFloatingWindow (*class in vanilla*), 18

I

ImageButton (*class in vanilla*), 87
ImageListCell() (*in module vanilla*), 69
ImageView (*class in vanilla*), 73
increment() (*vanilla.ProgressBar method*), 148
insertColumn() (*vanilla.GridView method*), 51
insertRow() (*vanilla.GridView method*), 51
insertView() (*vanilla.HorizontalRadioGroup method*), 142
insertView() (*vanilla.HorizontalStackView method*), 55
insertView() (*vanilla.VerticalRadioGroup method*), 139
insertView() (*vanilla.VerticalStackView method*), 59
isEnabled() (*vanilla.ActionButton method*), 134
isEnabled() (*vanilla.Button method*), 83
isEnabled() (*vanilla.ComboBox method*), 127
isEnabled() (*vanilla.DatePicker method*), 116
isEnabled() (*vanilla.EditText method*), 109
isEnabled() (*vanilla.GradientButton method*), 93
isEnabled() (*vanilla.HelpButton method*), 97
isEnabled() (*vanilla.ImageButton method*), 89
isEnabled() (*vanilla.LevelIndicator method*), 79
isEnabled() (*vanilla.PopUpButton method*), 130
isEnabled() (*vanilla.RadioGroup method*), 137
isEnabled() (*vanilla.SearchBox method*), 119
isEnabled() (*vanilla.SecureEditText method*), 112
isEnabled() (*vanilla.SegmentedButton method*), 100
isEnabled() (*vanilla.Slider method*), 145
isEnabled() (*vanilla.SquareButton method*), 86
isEnabled() (*vanilla.TextBox method*), 106
isOpen() (*vanilla.Drawer method*), 23
isPaneVisible() (*vanilla.SplitView method*), 35
isVisible() (*vanilla.ActionButton method*), 134
isVisible() (*vanilla.Box method*), 38
isVisible() (*vanilla.Button method*), 83
isVisible() (*vanilla.CheckBox method*), 122
isVisible() (*vanilla.ColorWell method*), 124
isVisible() (*vanilla.ComboBox method*), 127
isVisible() (*vanilla.DatePicker method*), 116
isVisible() (*vanilla.Drawer method*), 23
isVisible() (*vanilla.EditText method*), 109
isVisible() (*vanilla.FloatingWindow method*), 15
isVisible() (*vanilla.GradientButton method*), 93
isVisible() (*vanilla.GridView method*), 51
isVisible() (*vanilla.Group method*), 26
isVisible() (*vanilla.HelpButton method*), 97
isVisible() (*vanilla.HorizontalLine method*), 43

isVisible() (*vanilla.HorizontalRadioGroup method*), 142
isVisible() (*vanilla.HorizontalStackView method*), 55
isVisible() (*vanilla.ImageButton method*), 89
isVisible() (*vanilla.ImageView method*), 76
isVisible() (*vanilla.LevelIndicator method*), 79
isVisible() (*vanilla.List method*), 66
isVisible() (*vanilla.Popover method*), 41
isVisible() (*vanilla.PopUpButton method*), 130
isVisible() (*vanilla.ProgressBar method*), 148
isVisible() (*vanilla.ProgressSpinner method*), 151
isVisible() (*vanilla.RadioGroup method*), 137
isVisible() (*vanilla.ScrollView method*), 32
isVisible() (*vanilla.SearchBox method*), 119
isVisible() (*vanilla.SecureEditText method*), 112
isVisible() (*vanilla.SegmentedButton method*), 100
isVisible() (*vanilla.Sheet method*), 20
isVisible() (*vanilla.Slider method*), 145
isVisible() (*vanilla.SplitView method*), 35
isVisible() (*vanilla.SquareButton method*), 86
isVisible() (*vanilla.Tabs method*), 28
isVisible() (*vanilla.TextBox method*), 106
isVisible() (*vanilla.TextEditor method*), 103
isVisible() (*vanilla.VerticalLine method*), 46
isVisible() (*vanilla.VerticalRadioGroup method*), 139
isVisible() (*vanilla.VerticalStackView method*), 59
isVisible() (*vanilla.Window method*), 11

L

LevelIndicator (*class in vanilla*), 76
LevelIndicatorListCell() (*in module vanilla*), 72
List (*class in vanilla*), 60

M

makeKey() (*vanilla.FloatingWindow method*), 15
makeKey() (*vanilla.Sheet method*), 20
makeKey() (*vanilla.Window method*), 11
makeMain() (*vanilla.FloatingWindow method*), 15
makeMain() (*vanilla.Sheet method*), 20
makeMain() (*vanilla.Window method*), 11
move() (*vanilla.ActionButton method*), 134
move() (*vanilla.Box method*), 38
move() (*vanilla.Button method*), 83
move() (*vanilla.CheckBox method*), 122
move() (*vanilla.ColorWell method*), 125
move() (*vanilla.ComboBox method*), 127
move() (*vanilla.DatePicker method*), 116
move() (*vanilla.Drawer method*), 23
move() (*vanilla.EditText method*), 109
move() (*vanilla.FloatingWindow method*), 15
move() (*vanilla.GradientButton method*), 93
move() (*vanilla.GridView method*), 51
move() (*vanilla.Group method*), 26
move() (*vanilla.HelpButton method*), 97
move() (*vanilla.HorizontalLine method*), 44
move() (*vanilla.HorizontalRadioGroup method*), 142
move() (*vanilla.HorizontalStackView method*), 55
move() (*vanilla.ImageButton method*), 89
move() (*vanilla.ImageView method*), 76
move() (*vanilla.LevelIndicator method*), 79
move() (*vanilla.List method*), 66
move() (*vanilla.Popover method*), 41
move() (*vanilla.PopUpButton method*), 130
move() (*vanilla.ProgressBar method*), 148
move() (*vanilla.ProgressSpinner method*), 151
move() (*vanilla.RadioGroup method*), 137
move() (*vanilla.ScrollView method*), 32
move() (*vanilla.SearchBox method*), 119
move() (*vanilla.SecureEditText method*), 112
move() (*vanilla.SegmentedButton method*), 100
move() (*vanilla.Slider method*), 145
move() (*vanilla.SplitView method*), 35
move() (*vanilla.SquareButton method*), 86
move() (*vanilla.Tabs method*), 29
move() (*vanilla.TextBox method*), 106
move() (*vanilla.TextEditor method*), 103
move() (*vanilla.VerticalLine method*), 47
move() (*vanilla.VerticalRadioGroup method*), 139
move() (*vanilla.VerticalStackView method*), 59
move() (*vanilla.Window method*), 11
moveColumn() (*vanilla.GridView method*), 51
moveRow() (*vanilla.GridView method*), 51

O

open() (*vanilla.Drawer method*), 23
open() (*vanilla.FloatingWindow method*), 15
open() (*vanilla.Popover method*), 41
open() (*vanilla.Sheet method*), 20
open() (*vanilla.Window method*), 11

P

Popover (*class in vanilla*), 39
PopUpButton (*class in vanilla*), 128
PopUpButtonListCell() (*in module vanilla*), 68
ProgressBar (*class in vanilla*), 146
ProgressSpinner (*class in vanilla*), 149

R

RadioGroup (*class in vanilla*), 134
removeColumn() (*vanilla.GridView method*), 51
removeRow() (*vanilla.GridView method*), 51
removeToolbarItem() (*vanilla.FloatingWindow method*), 18
removeToolbarItem() (*vanilla.Window method*), 14

removeView() (*vanilla.HorizontalRadioGroup method*), 142
removeView() (*vanilla.HorizontalStackView method*), 55
removeView() (*vanilla.VerticalRadioGroup method*), 139
removeView() (*vanilla.VerticalStackView method*), 59
resize() (*vanilla.ActionButton method*), 134
resize() (*vanilla.Box method*), 38
resize() (*vanilla.Button method*), 83
resize() (*vanilla.CheckBox method*), 122
resize() (*vanilla.ColorWell method*), 125
resize() (*vanilla.ComboBox method*), 127
resize() (*vanilla.DatePicker method*), 116
resize() (*vanilla.Drawer method*), 23
resize() (*vanilla.EditText method*), 109
resize() (*vanilla.FloatingWindow method*), 15
resize() (*vanilla.GradientButton method*), 93
resize() (*vanilla.GridView method*), 52
resize() (*vanilla.Group method*), 26
resize() (*vanilla.HelpButton method*), 97
resize() (*vanilla.HorizontalLine method*), 44
resize() (*vanilla.HorizontalRadioGroup method*), 142
resize() (*vanilla.HorizontalStackView method*), 55
resize() (*vanilla.ImageButton method*), 90
resize() (*vanilla.ImageView method*), 76
resize() (*vanilla.LevelIndicator method*), 79
resize() (*vanilla.List method*), 66
resize() (*vanilla.Popover method*), 41
resize() (*vanilla.PopUpButton method*), 130
resize() (*vanilla.ProgressBar method*), 148
resize() (*vanilla.ProgressSpinner method*), 151
resize() (*vanilla.RadioGroup method*), 137
resize() (*vanilla.ScrollView method*), 32
resize() (*vanilla.SearchBox method*), 119
resize() (*vanilla.SecureEditText method*), 112
resize() (*vanilla.SegmentedButton method*), 100
resize() (*vanilla.Slider method*), 145
resize() (*vanilla.SplitView method*), 35
resize() (*vanilla.SquareButton method*), 86
resize() (*vanilla.Tabs method*), 29
resize() (*vanilla.TextBox method*), 106
resize() (*vanilla.TextEditor method*), 103
resize() (*vanilla.VerticalLine method*), 47
resize() (*vanilla.VerticalRadioGroup method*), 139
resize() (*vanilla.VerticalStackView method*), 59
resize() (*vanilla.Window method*), 11
rowIsVisible() (*vanilla.GridView method*), 52

S

scrollToSelection() (*vanilla.List method*), 66
ScrollView (*class in vanilla*), 29
SearchBox (*class in vanilla*), 117
SecureEditText (*class in vanilla*), 110
SegmentedButton (*class in vanilla*), 97
SegmentedButtonListCell() (*in module vanilla*), 71
select() (*vanilla.FloatingWindow method*), 15
select() (*vanilla.Sheet method*), 20
select() (*vanilla.Window method*), 11
selectAll() (*vanilla.EditText method*), 109
selectAll() (*vanilla.SecureEditText method*), 112
selectAll() (*vanilla.TextEditor method*), 103
set() (*vanilla.ActionButton method*), 134
set() (*vanilla.CheckBox method*), 122
set() (*vanilla.ColorWell method*), 125
set() (*vanilla.ComboBox method*), 127
set() (*vanilla.DatePicker method*), 116
set() (*vanilla.EditText method*), 109
set() (*vanilla.HorizontalRadioGroup method*), 142
set() (*vanilla.LevelIndicator method*), 79
set() (*vanilla.List method*), 66
set() (*vanilla.PopUpButton method*), 131
set() (*vanilla.ProgressBar method*), 148
set() (*vanilla.RadioGroup method*), 137
set() (*vanilla.SearchBox method*), 119
set() (*vanilla.SecureEditText method*), 112
set() (*vanilla.SegmentedButton method*), 100
set() (*vanilla.Slider method*), 145
set() (*vanilla.Tabs method*), 29
set() (*vanilla.TextBox method*), 106
set() (*vanilla.TextEditor method*), 103
set() (*vanilla.VerticalRadioGroup method*), 140
setBackgroundColor() (*vanilla.ScrollView method*), 32
setBorderColor() (*vanilla.Box method*), 38
setBorderColor() (*vanilla.HorizontalLine method*), 44
setBorderColor() (*vanilla.VerticalLine method*), 47
setBorderWidth() (*vanilla.Box method*), 38
setBorderWidth() (*vanilla.HorizontalLine method*), 44
setBorderWidth() (*vanilla.VerticalLine method*), 47
setCornerRadius() (*vanilla.Box method*), 38
setCornerRadius() (*vanilla.HorizontalLine method*), 44
setCornerRadius() (*vanilla.VerticalLine method*), 47
setCriticalValue() (*vanilla.LevelIndicator method*), 79
setDefaultButton() (*vanilla.FloatingWindow method*), 16
setDefaultButton() (*vanilla.Window method*), 11
setDividerDrawingFunction() (*vanilla.SplitView method*), 35
setEdgeInsets() (*vanilla.HorizontalRadioGroup method*), 142

setEdgeInsets() (*vanilla.HorizontalStackView method*), 55
setEdgeInsets() (*vanilla.VerticalRadioGroup method*), 140
setEdgeInsets() (*vanilla.VerticalStackView method*), 59
setFillColor() (*vanilla.Box method*), 38
setFillColor() (*vanilla.HorizontalLine method*), 44
setFillColor() (*vanilla.VerticalLine method*), 47
setImage() (*vanilla.GradientButton method*), 94
setImage() (*vanilla.ImageButton method*), 90
setImage() (*vanilla.ImageView method*), 76
setItem() (*vanilla.ActionButton method*), 134
setItem() (*vanilla.PopUpButton method*), 131
setItems() (*vanilla.ActionButton method*), 134
setItems() (*vanilla.ComboBox method*), 128
setItems() (*vanilla.PopUpButton method*), 131
setMargins() (*vanilla.Box method*), 38
setMargins() (*vanilla.HorizontalLine method*), 44
setMargins() (*vanilla.VerticalLine method*), 47
setMaxValue() (*vanilla.LevelIndicator method*), 79
setMaxValue() (*vanilla.Slider method*), 145
setMinValue() (*vanilla.LevelIndicator method*), 79
setMinValue() (*vanilla.Slider method*), 145
setPlaceholder() (*vanilla.EditText method*), 109
setPlaceholder() (*vanilla.SecureEditText method*), 112
setPosSize() (*vanilla.ActionButton method*), 134
setPosSize() (*vanilla.Box method*), 38
setPosSize() (*vanilla.Button method*), 83
setPosSize() (*vanilla.CheckBox method*), 122
setPosSize() (*vanilla.ColorWell method*), 125
setPosSize() (*vanilla.ComboBox method*), 128
setPosSize() (*vanilla.DatePicker method*), 116
setPosSize() (*vanilla.Drawer method*), 23
setPosSize() (*vanilla.EditText method*), 109
setPosSize() (*vanilla.FloatingWindow method*), 15
setPosSize() (*vanilla.GradientButton method*), 94
setPosSize() (*vanilla.GridView method*), 52
setPosSize() (*vanilla.Group method*), 26
setPosSize() (*vanilla.HelpButton method*), 97
setPosSize() (*vanilla.HorizontalLine method*), 44
setPosSize() (*vanilla.HorizontalRadioGroup method*), 142
setPosSize() (*vanilla.HorizontalStackView method*), 55
setPosSize() (*vanilla.ImageButton method*), 90
setPosSize() (*vanilla.ImageView method*), 76
setPosSize() (*vanilla.LevelIndicator method*), 80
setPosSize() (*vanilla.List method*), 66
setPosSize() (*vanilla.Popover method*), 41
setPosSize() (*vanilla.PopUpButton method*), 131
setPosSize() (*vanilla.ProgressBar method*), 149
setPosSize() (*vanilla.ProgressSpinner method*), 151
setPosSize() (*vanilla.RadioGroup method*), 137
setPosSize() (*vanilla.ScrollView method*), 32
setPosSize() (*vanilla.SearchBox method*), 119
setPosSize() (*vanilla.SecureEditText method*), 112
setPosSize() (*vanilla.SegmentedButton method*), 100
setPosSize() (*vanilla.Slider method*), 145
setPosSize() (*vanilla.SplitView method*), 36
setPosSize() (*vanilla.SquareButton method*), 86
setPosSize() (*vanilla.Tabs method*), 29
setPosSize() (*vanilla.TextBox method*), 106
setPosSize() (*vanilla.TextEditor method*), 103
setPosSize() (*vanilla.VerticalLine method*), 47
setPosSize() (*vanilla.VerticalRadioGroup method*), 140
setPosSize() (*vanilla.VerticalStackView method*), 59
setPosSize() (*vanilla.Window method*), 11
setSelection() (*vanilla.List method*), 66
setTickMarkCount() (*vanilla.Slider method*), 145
setTickMarkPosition() (*vanilla.Slider method*), 145
setTitle() (*vanilla.ActionButton method*), 134
setTitle() (*vanilla.Box method*), 38
setTitle() (*vanilla.Button method*), 83
setTitle() (*vanilla.CheckBox method*), 122
setTitle() (*vanilla.ComboBox method*), 128
setTitle() (*vanilla.DatePicker method*), 116
setTitle() (*vanilla.EditText method*), 109
setTitle() (*vanilla.FloatingWindow method*), 15
setTitle() (*vanilla.GradientButton method*), 94
setTitle() (*vanilla.HelpButton method*), 97
setTitle() (*vanilla.HorizontalLine method*), 44
setTitle() (*vanilla.ImageButton method*), 90
setTitle() (*vanilla.LevelIndicator method*), 80
setTitle() (*vanilla.PopUpButton method*), 131
setTitle() (*vanilla.RadioGroup method*), 137
setTitle() (*vanilla.SearchBox method*), 119
setTitle() (*vanilla.SecureEditText method*), 113
setTitle() (*vanilla.SegmentedButton method*), 100
setTitle() (*vanilla.Slider method*), 146
setTitle() (*vanilla.SquareButton method*), 86
setTitle() (*vanilla.TextBox method*), 106
setTitle() (*vanilla.VerticalLine method*), 47
setTitle() (*vanilla.Window method*), 11
setWarningValue() (*vanilla.LevelIndicator method*), 80
Sheet (*class in vanilla*), 19
show() (*vanilla.ActionButton method*), 134
show() (*vanilla.Box method*), 38
show() (*vanilla.Button method*), 83
show() (*vanilla.CheckBox method*), 122
show() (*vanilla.ColorWell method*), 125
show() (*vanilla.ComboBox method*), 128

show() (*vanilla.DatePicker method*), 116
show() (*vanilla.Drawer method*), 23
show() (*vanilla.EditText method*), 109
show() (*vanilla.FloatingWindow method*), 15
show() (*vanilla.GradientButton method*), 94
show() (*vanilla.GridView method*), 52
show() (*vanilla.Group method*), 26
show() (*vanilla.HelpButton method*), 97
show() (*vanilla.HorizontalLine method*), 44
show() (*vanilla.HorizontalRadioGroup method*), 142
show() (*vanilla.HorizontalStackView method*), 56
show() (*vanilla.ImageButton method*), 90
show() (*vanilla.ImageView method*), 76
show() (*vanilla.LevelIndicator method*), 80
show() (*vanilla.List method*), 66
show() (*vanilla.Popover method*), 41
show() (*vanilla.PopUpButton method*), 131
show() (*vanilla.ProgressBar method*), 149
show() (*vanilla.ProgressSpinner method*), 151
show() (*vanilla.RadioGroup method*), 137
show() (*vanilla.ScrollView method*), 32
show() (*vanilla.SearchBox method*), 119
show() (*vanilla.SecureEditText method*), 113
show() (*vanilla.SegmentedButton method*), 100
show() (*vanilla.Sheet method*), 20
show() (*vanilla.Slider method*), 146
show() (*vanilla.SplitView method*), 36
show() (*vanilla.SquareButton method*), 86
show() (*vanilla.Tabs method*), 29
show() (*vanilla.TextBox method*), 106
show() (*vanilla.TextEditor method*), 103
show() (*vanilla.VerticalLine method*), 47
show() (*vanilla.VerticalRadioGroup method*), 140
show() (*vanilla.VerticalStackView method*), 60
show() (*vanilla.Window method*), 11
showColumn() (*vanilla.GridView method*), 52
showPane() (*vanilla.SplitView method*), 36
showRow() (*vanilla.GridView method*), 52
Slider (*class in vanilla*), 142
SliderListCell() (*in module vanilla*), 67
SplitView (*class in vanilla*), 32
SquareButton (*class in vanilla*), 83
start() (*vanilla.ProgressBar method*), 149
start() (*vanilla.ProgressSpinner method*), 151
stop() (*vanilla.ProgressBar method*), 149
stop() (*vanilla.ProgressSpinner method*), 152

T

Tabs (*class in vanilla*), 26
TextBox (*class in vanilla*), 103
TextEditor (*class in vanilla*), 101
toggle() (*vanilla.CheckBox method*), 122
toggle() (*vanilla.Drawer method*), 23
togglePane() (*vanilla.SplitView method*), 36

U

unbind() (*vanilla.FloatingWindow method*), 16
unbind() (*vanilla.Popover method*), 41
unbind() (*vanilla.Window method*), 12

V

vanilla (*module*), 9, 14, 19, 20, 24, 26, 29, 32, 36, 39, 42, 44, 47, 52, 56, 60, 73, 76, 80, 83, 87, 90, 94, 97, 101, 103, 106, 110, 113, 117, 119, 122, 125, 128, 131, 134, 142, 146, 149
vanilla2 (*module*), 60
VerticalLine (*class in vanilla*), 44
VerticalRadioGroup (*class in vanilla*), 137
VerticalStackView (*class in vanilla*), 56

W

Window (*class in vanilla*), 9