

## Cocos2 and pyglet Quick Reference (v1.7; © Richard Jones, richard@mechanicalcat.net 2010)

<b>from cocos.director import director</b>	
director.init( <i>arguments</i> )	by default cocos2d will auto-scale requested dimensions to the window size args include all pyglet.window.Window args and do_not_scale*
director.run( <i>scene</i> )	run the Scene
director.replace( <i>scene</i> )	replace the currently-running Scene with the supplied one
director.push( <i>scene</i> )	run the supplied Scene with the ability to return to the current one
director.pop()	return to the previous scene after a push()
director.get_window_size()	return the (width, height) of the window
director.get_virtual_coordinates( <i>x, y</i> )	map window coordinates to logical scene coordinates**
director.scene	the currently-active Scene
director.return_value	the value from the last Scene end() * turns auto-scaling off
director.window	the pyglet.window.Window ** if auto-scaling is on

**A cocosnode is a layer, sprite, text, canvas, scene, ...** (so `sprite.position`, `layer.scale = .5`, `scene.pause()`, ...)

<code>.add(<i>node</i>)</code> , <code>.remove(<i>node</i>)</code> , <code>.kill()</code>	... <code>scene.add(layer)</code> , <code>sprite.add(text)</code> , <code>layer.remove(sprite)</code> or <code>sprite.kill()</code>
<code>.schedule(<i>callback</i>)</code>	... and <code>unschedule()</code> etc. as per pyglet.clock API
<code>.pause()</code> , <code>.resume()</code>	pause and resume the execution of actions & scheduled calls
<code>.are_actions_running()</code>	determine whether any actions are running
<code>.on_enter()</code> , <code>.on_exit()</code>	called as the node enters and leaves the stage (don't forget to <code>super()</code> )
<code>.anchor</code>	pixel that position is relative to; used to transform about*
<code>.position</code>	position in (x,y) coordinates
<code>.scale</code>	scale where 1.0 the default value
<code>.rotation</code>	rotation in degrees * default is center

<b>from cocos.scene import Scene</b>	
<code>scene.end(<i>return_value</i>)</code>	end the Scene and set <code>director.return_value</code>
<b>class MyScene(cocos.layer.Scene):</b>	
def __init__(self):	initialise at creation; don't forget to <code>super(MyScene, self).__init__()</code>

<b>from cocos.layer import ...</b>	
<code>Layer()</code>	standard layer containing sprites, text, layers, ...
<code>MultiplexLayer(layer, layer, ...)</code>	composite layer that displays one layer of many at a time
<code>PythonInterpreterLayer()</code>	runs an interactive Python interpreter in a Layer
<code>ScrollableLayer(parallax=1)</code>	requires the parent to be <code>cocos.layers.ScrollingManager</code>
<code>ColorLayer(r, g, b, a)</code>	solid color layer
<b>class MyLayer(cocos.layer.Layer):</b>	
def __init__(self):	initialise at creation; don't forget to <code>super(MyLayer, self).__init__()</code>
is_event_handler = False	True to register standard pyglet event handlers on this layer
	<code>cocos2d</code> adds scaling-aware <code>on_cocos_resize(width, height)</code>

<b>from cocos.sprite import Sprite</b>	<code>sprite = Sprite('data/ship.png')</code>
<code>opacity</code>	opacity of the sprite where 0 is transparent and 255 is solid
<code>color</code>	color in R,G,B format where 0,0,0 is black and 255,255,255 is white

<b>from cocos.text import ...</b>	
<code>Label(text, position, ...)</code>	plain-text label
<code>HTMLLabel(text, position, ...)</code>	HTML 4.01 subset text label (see <code>pyglet.text.formats.html</code> for details)
<code>RichLabel(text, position, ...)</code>	rich text label with markup as per <code>pyglet.text.DocumentLabel</code>

<b>import cocos.tiles</b>	<i>scrolling...</i>
<code>level = cocos.tiles.load('my-level.xml')</code>	<code>manager = cocos.layers.ScrollingManager()</code>
<code>map = level['map-1']</code>	<code>manager.add(map)</code>
<code>scene = cocos.scene.Scene(map)</code>	<code>scene = cocos.scene.Scene(manager)</code>
<code>map.set_view(x, y, width, height)</code>	<code>manager.set_focus(x, y) or...</code>
<i>note: use do_not_scale when using tile maps</i>	<code>manager.force_focus(x, y)</code>

<b>keyboard status handler and key constants</b>		
<code>from pyglet.window import key</code>	<code>key.RIGHT</code>	<code>key.SPACE</code>
<code>keys = key.KeyStateHandler()</code>	<code>key.LEFT</code>	<code>key.A -&gt; key.Z</code>
<code>director.window.push_handlers(keys)</code>	<code>key.UP</code>	<code>key._0 -&gt; key._9</code>
	<code>key.DOWN</code>	<code>key.ENTER</code>

<b>from cocos.menu import Menu, MenuItem</b>	also <code>EntryMenuItem</code> , <code>ToggleMenuItem</code> , <code>ImageMenuItem</code> , ...	
<code>menu = cocos.menu.Menu('My Game Title')</code>		
<code>menu.create_menu([</code>		
<code>MenuItem('Play', lambda: director.push(TheGameScene()))</code> ,		
<code>MenuItem('Quit', pyglet.app.exit)]</code>		
<code>menu.on_quit = pyglet.app.exit</code>		
<code>director.run(cocos.scene.Scene(menu))</code>		

## Cocos2 and pyglet Quick Reference (v1.7; © Richard Jones, richard@mechanicalcat.net 2010)

	from cocos.actions import ...	cocosnode.do(action) (sprite.do, layer.do, ...)
Translation	MoveBy(delta, duration=5)	Moves the sprite <i>delta</i> =(x, y) pixels
	MoveTo	Moves the sprite to <i>position</i> =(x,y)
	JumpBy(delta, height=100, jumps=1, duration=5)	Jump the sprite <i>delta</i> =(x, y), <i>height</i> pixels using <i>jumps</i> hops
	JumpTo(position, height=100, jumps=1, duration=5)	Jump the sprite to <i>position</i> =(x,y), <i>height</i> pixels using <i>jumps</i> hops
	Bezier(bezier, duration=5)	Move the sprite through the <i>bezier</i> curve (cocos.path.Bezier instance)
	Place(position)	Instantly place the sprite at the <i>position</i> =(x, y)
Transform	ScaleBy(scale, duration=5)	Scale the sprite by <i>scale</i> times
	ScaleTo(scale, duration=5)	Scale the sprite to <i>scale</i>
	RotateBy(angle, duration=5)	Rotate the target by <i>angle</i> degrees
	RotateTo(angle, duration=5)	Rotate the sprite to the given <i>angle</i>
Visibility	Show()	Show the sprite
	Hide()	Hide the sprite from view
	Blink(blinks, duration)	Blink the sprite the number of <i>blinks</i> over the <i>duration</i> seconds
	ToggleVisibility()	Show if hidden and hide if shown
	FadeIn(duration)	Fade the sprite into view over <i>duration</i> seconds
	FadeOut(duration)	Fade the sprite out of view over <i>duration</i> seconds
	FadeTo(opacity, duration)	Fade the sprite to a specific <i>opacity</i> over <i>duration</i> seconds
Modifiers	Accelerate(action, rate=2)	Accelerate the <i>action</i> at its end by the given <i>rate</i> (1 is linear)
	AccelDeccel(action)	Accelerate the <i>action</i> in its middle
	Speed(action, rate)	Speed up or slow down the <i>action</i> by the given <i>rate</i> (1 is normal)
	Reverse(action)	Perform the <i>action</i> in reverse
Combine	Sequence(action, action) (+ operator)	Execute actions in sequence
	Spawn(action, action) (! operator)	Execute actions at the same time
	Repeat(action)	Repeat an action (or composite set of actions) forever
	Loop(action, times) (* operator)	Loop the action <i>n</i> times
Special	Delay(time)	Delay for <i>time</i> seconds
	RandomDelay(low, high)	Delay for some seconds below <i>low</i> and <i>high</i>
	CallFunc(callable)	Invoke the <i>callable</i> (with no arguments)
	CallFuncS(callable)	Invoke the <i>callable</i> with the sprite as the first argument
	OrbitCamera( <i>spherical coordinate arguments</i> )	Orbits the camera around the center of the screen
Move	Move()	Move the sprite based on sprite parameters (velocity, acceleration, ...)
	BoundedMove(width, height)	As above but limit movement to 0 < x < width and 0 < y < height
	WrappedMove(width, height)	As above but wrap movement outside 0 < x < width and 0 < y < height
	Driver()	Drive the sprite like a car using sprite parameters (direction, speed, ...)
<pre> class MyAction(cocos.actions.Action):     def init(self):         gets called at initialization time, before a target is defined     def step(self, dt):         called every frame with dt being the number of seconds since last call     def done(self):         return False while the step method must be called     def start(self):         start executing an action; self.target is assigned and this method is called     def stop(self):         after we finish executing an action this method is called  class MyIntervalAction(cocos.actions.IntervalAction):     def update(self, t):         called every frame with t ranging from 0..1         (also init, start and stop) </pre>		

# Cocos2 and pyglet Quick Reference (v1.7; © Richard Jones, richard@mechanicalcat.net 2010)

## import pyglet

`window = pyglet.window.Window(...)` create a window with optional arguments  
`pyglet.app.run()` run pyglet's main loop to handle events

## registering event handlers

```
def on_draw():  
    window.clear()  
    ...  
window.push_handlers(on_draw)
```

see next page for all possible window event names  
clear the window to the `pyglet.gl.glClearColor` color  
put your other drawing code here  
push the `on_draw` handler to the window

```
class MyClass(object):  
    def on_draw(self):  
    def on_text(self, text):
```

return `pyglet.event.EVENT_HANDLED` (True) if the event has been handled

```
my_object = MyClass()  
window.push_handlers(my_object)
```

push *all* handlers defined on `my_object` onto the window's event stack

## Window arguments

`fullscreen` (there are other arguments, these are just the most common)  
make the window fullscreen  
`width=640, height=480` create the window with these dimensions (ignored if fullscreen)  
`resizable=False` allow the user to resize the window  
`vsync=True` synchronise to the monitor to avoid flicker  
`caption=sys.argv[0]` set the window title text  
`config=None` a display config as per `pyglet.gl.Config`  
`screen=None` the screen to use if fullscreen

## Image handling

```
pyglet.image.load(filename, file=None)
```

load the image from the named file or supplied file object

```
SolidColorImagePattern
```

create an image filled with a single color

```
CheckerImagePattern
```

create an image with a tileable checker image of two colors

```
image.width, image.height
```

image dimensions in pixels

```
image.anchor_x, image.anchor_y
```

coordinate of anchor, relative to bottom-left corner of image

```
image.blit(x, y, z=0)
```

render the image to the active framebuffer

```
image.save(filename, file=None)
```

save the image as a PNG file

```
image.texture
```

a `pyglet.image.Texture` view of this image

```
texture.target, texture.id
```

OpenGL texture target and id

```
texture.tex_coords
```

12-tuple of float texture coordinates (may not be simply 0 and 1)

```
get_buffer_manager().get_color_buffer()
```

get the active framebuffer as an Image

```
load_animation(filename, file=None)
```

load an animation from a file - currently only GIF is supported

```
Animation.from_image_sequence(sequence, period, loop=False)
```

create an animation from a sequence of images

## Sprites

(all attributes are re-assignable)

```
pyglet.sprite.Sprite(image, ...)
```

create a sprite from the image *or* Animation instance

```
sprite.position
```

position of the sprite in (x, y) (also as `sprite.x`, `sprite.y`)

```
sprite.image
```

image rendered for the sprite (image anchor is honored)

```
sprite.rotation
```

sprite rotation in degrees

```
sprite.scale
```

amount to scale the sprite image by - 1.0 is unscaled

```
sprite.opacity
```

control transparency - 0 is fully transparent, 255 is fully opaque

```
sprite.color
```

coloring of sprite image, normal (white) is R, G, B (255, 255, 255)

```
sprite.visible
```

boolean controlling sprite visibility

```
sprite.draw()
```

render the sprite to the active framebuffer

## Text rendering

(see the docs for the complete, extensive set of options you may pass)

```
pyglet.text.Label(text, ...)
```

lay out some plain text

```
pyglet.text.HTMLLabel(text, ...)
```

lay out some HTML (4.01, limited) text

```
text.draw()
```

render the laid-out text

## Media playback

```
pyglet.media.load(filename, file=None)
```

load the media file (audio, video or both) as a "source"

```
source.audio_format
```

an instance of `pyglet.media.AudioFormat` or None if no audio

```
source.video_format
```

an instance of `pyglet.media.VideoFormat` or None if no video

```
source.info
```

a `pyglet.media.SourceInfo` giving title, author, etc. if known

```
source.play()
```

convenience method to immediately play the source

```
player = Player()
```

create a player to manage playback; see possible events below

```
player.queue(source)
```

queue the source to be played

```
player.play(), .pause(), .stop()
```

control playback

```
player.time, player.seek(time)
```

report current position and seek to a different *time*

```
player.get_texture()
```

get the current video frame as a `pyglet.image.Texture`

```
player.eos_action
```

the action the player will take when it reaches the end of the current source

# Cocos2 and pyglet Quick Reference (v1.7; © Richard Jones, richard@mechanicalcat.net 2010)

## Graphics abstraction

<code>pyglet.graphics.Batch()</code>	batch up graphics drawing operations
<code>pyglet.sprite.Sprite(image, batch=batch)</code>	create a sprite belonging to the batch
<code>batch.draw()</code>	<b>much</b> faster than individual <code>sprite.draw()</code> calls
<code>pyglet.graphics.Group</code>	group common OpenGL state objects in a batch
<code>pyglet.graphics.OrderedGroup</code>	arbitrarily order objects in a batch (typically for display sorting)
<code>pyglet.graphics.TextureGroup</code>	enable and bind a texture for a group of objects in a batch
<code>batch.add(count, mode, group, *data)</code>	create an OpenGL vertex list in the batch using <i>data</i> 's items
<code>batch.add(count, mode, group, indices, *data)</code>	create an OpenGL indexed vertex list

```
# draw a white line from (0, 1) to (1, 0)
vertex_list = batch.add(2, GL_LINES, None, ('v2f', (0.0, 1.0, 1.0, 0.0)),
                        ('c4B', (255, 255, 255, 255) * 2))
```

## Resources

<code>pyglet.resource.image(filename)</code>	load the named image file
<code>pyglet.resource.media(filename)</code>	load the named media file
<code>pyglet.resource.add_font(filename)</code>	make a font available to pyglet's text rendering
<code>pyglet.resource.file(filename)</code>	open the named resource file, returning a file object
<code>pyglet.resource.location(filename)</code>	return the location of the resource file (only useful for on-disk files)
<code>pyglet.resource.path</code>	list containing the places to look for resources
<code>pyglet.resource.reindex()</code>	should be called if the path is modified

## Clock handling

<code>pyglet.clock.schedule(callback)</code>	note: <code>pyglet.app.run()</code> automatically calls <code>pyglet.clock.tick()</code>
<code>...unschedule(callback)</code>	<i>callback</i> when the clock is ticked, passing the seconds since last call
<code>...schedule_interval(callback, n)</code>	remove <i>callback</i> from the schedule
<code>...schedule_once(callback, n)</code>	<i>callback</i> every <i>n</i> seconds
<code>fps = pyglet.clock.ClockDisplay()</code>	<i>callback</i> once in <i>n</i> seconds
	a simple FPS counter .. use <code>fps.draw()</code> to display

## pyglet media player events

<code>on_player_eos()</code>	the player has run out of sources
<code>on_source_group_eos()</code>	the current source group has run out of data
<code>on_eos()</code>	the current source has run out of data

## pyglet window events

<code>on_key_press(symbol, modifiers)</code>	<i>symbol</i> and <i>modifiers</i> as in <code>pyglet.window.key</code> keys and <code>MOD_*</code>
<code>on_key_release(symbol, modifiers)</code>	as above
<code>on_text(text)</code>	<i>text</i> is a unicode string of the text input
<code>on_text_motion(motion)</code>	<i>motion</i> as in <code>pyglet.window.key.MOTION_*</code>
<code>on_text_motion_select(motion)</code>	as above but during a text selection event ( <code>MOD_SHIFT</code> held)
<code>on_mouse_press(x, y, button, modifiers)</code>	<i>buttons</i> and <i>modifiers</i> pressed at position ( <i>x, y</i> )
<code>on_mouse_release(x, y, button, modifiers)</code>	as above but <i>buttons</i> released
<code>on_mouse_motion(x, y, dx, dy)</code>	mouse absolute ( <i>x, y</i> ) and movement ( <i>dx, dy</i> ) since last event
<code>on_mouse_drag(x, y, dx, dy, buttons, modifiers)</code>	as above but with <i>buttons</i> and <i>modifiers</i> held
<code>on_mouse_scroll(x, y, dx, dy)</code>	mouse scroll wheel scrolled by ( <i>dx, dy</i> ) at position ( <i>x, y</i> )
<code>on_mouse_enter(x, y)</code>	mouse entered window
<code>on_mouse_leave(x, y)</code>	mouse exited window
<code>on_resize(width, height)</code>	the window has been created or resized
<code>on_draw()</code>	application should draw (not relevant to cocos2d)
<code>on_show()</code>	window has been made visible (or created)
<code>on_hide()</code>	window has been hidden
<code>on_close()</code>	window close button pressed
<code>on_expose()</code>	redraw is needed
<code>on_move(x, y)</code>	window has been moved to position ( <i>x, y</i> )
<code>on_activate()</code>	window has been activated (focused)
<code>on_deactivate()</code>	window has been deactivated (lost focus)
<code>on_context_lost()</code>	window's OpenGL context was lost (no drawing possible)
<code>on_context_state_lost()</code>	window's OpenGL context state was destroyed by pyglet

## pyglet sprite event

<code>on_animation_end</code>	the sprite's image animation has ended
-------------------------------	--