Michel Anders

Blender add-on Cookbook

This is a sample of my book

Blender Add-on Cookbook

A cookbook with useful programming recipes for Blender add-ons

---

If you like to buy a full copy, please visit my Blender Market store:

https://blendermarket.com/creators/varkenvarken

In the store you will also find a more gentle introduction to add-ons called

Creating Add-ons for Blender, a practical primer.

Table of Contents

Introduction	5
Information	6
Adding an object to a scene	9
Selecting connected vertices	13
Add a modifier to an object	18
Add a vertex group to an object	21
Change vertex, edge and face properties	27
Change vertex, edge and face data layers	36
Create a Cycles material	44
Add a particle system to an object	47
Create a property selection drop down	51
Make a function available to pydrivers	55
Show a progress bar	58
Showing an info message	63
Show hints in an area header	66
Configuration parameters for add-ons	69
Defining shortcut keys	73
Use Numpy to accelerate vertex manipulation	77
Adding nodes to materials	82
Combining nodes into frames	87
Add an operator to a menu	93
Adding persistent properties to objects	97
Adding sub-menus to existing menus	103
Drawing using OpenGL	107
Using a panel to control a modal operator	114
Getting pixel values using OpenGL	118
Converting between world and object coordinates	122
Using kd-trees to speed up intersection tests	127
Converting 2d screen coordinates to 3d world coordinates	132
Converting 3d coordinates to 2d screen coordinates	136
Create a modal operator	141
Using pass through in a modal operator	144

Change the cursor in a modal operator 147
Package add-ons with more than one module 150
Bundling external assets with your add-on 153
Adding custom icons to menu entries 156
Window management in Blender 161
Creating a curve through objects 165
Parenting objects to curve vertices 171
Acknowledgments 176

Blender Add-on Cookbook

By Michel Anders

Copyright 2017 Michel Anders

Blender Market Edition

Blender Market Edition, License Notes

This ebook is licensed for your personal enjoyment only. This ebook may not be re-sold or given away to other people. If you would like to share this book with another person, please purchase an additional copy for each recipient. If you're reading this book and did not purchase it, or it was not purchased for your use only, then please return to blendermarket.com and purchase your own copy. Thank you for respecting the hard work of this author

Introduction

In the last couple of year I have created numerous add-ons and while researching and developing these add-ons I encountered all sorts of different issues that where not always very straightforward to solve.

Blender has of course extensive documentation on its Python API and a helpful community as well but sometimes it nevertheless took quite some time to figure something out.

So I decided to document these issues and bundle then into this book, where you will find quite a collection of solutions to commonly occurring issues. Some are obvious once you see and understand them, others showcase subtleties that often prove to cause trouble in unexpected places if you are not aware of them.

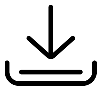
I have done my best to provide not just solutions but accompany them with detailed explanations as well. Also, where appropriate I have included references to all sorts of documentation, often the most relevant spot in the Blender Python API docs, just to save you from an endless hunt for the right information. An index of terms is also provided.

Information

Who is this book for?

This book is for add-on developers who want to go one step further and add a professional touch to their creations or want to add functionality that isn't so straightforward to implement.

You should have a good understanding of Python and have some knowledge of writing Blender add-ons already. This is a cookbook in the sense that it provides small pieces of code that are aimed at solving a particular problem but if you want a more general introduction you could consider my book Creating add-ons for Blender.



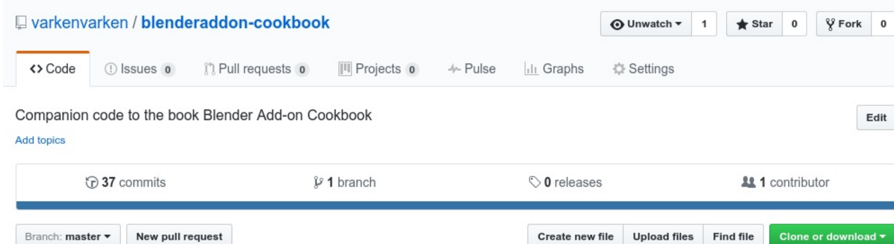
Code availability

The code in the book is presented as code snippets *that will not run on their own*. Typically just the **`execute()`** function of an operator is shown. However, to spare you the effort of typing over many lines of code, each recipe refers to a file that you can download and that implements a small but complete add-on that uses the code shown in the snippet, so you can test it out immediately.

All code examples are licensed under the GPL and available from this GitHub repository:

https://github.com/varkenvarken/blenderaddon-cookbook.

Each recipe contains individual references to the relevant code but you can also download all code a a single .zip file by clicking on the 'Clone or download' button and selecting 'Download as .zip':



All code is tested against the latest version of Blender (2.78a at time of writing)

Conventions used in this book

Code snippets are shown as follows

```
def myfunction(x):
    y = x * x
    return y
```

Function or class names used in the running text are shown in bold: **myfunction()**.

Pythonic code

The recipes in this book focus on illustrating concepts relevant to writing Blender add-ons. We do strive to keep the code readable and Pythonic but not at all costs. Sometimes the limitations inherent in formating source code in books force us to sacrifice the recommendation in pep-8. Also, although we are aware of the distinctions, we use the terms *function* and *method* interchangeably. And as for values passed to functions we follow the Klingon definition.

About the author

Although a Blender user for over ten years, I have to admit that I am an enthusiastic but (very) mediocre artist at best. I discovered however that I really enjoyed helping people out with programming related questions and a couple of years ago when Packt Publishing was looking for authors on the BlenderArtists/Python forum I stepped in.

So far this has resulted in several books in print:

Blender 2.49 Scripting, ISBN 9781849510400, Published by Packt Publishing in April 2010.

Python 3 Web Development, ISBN 9781849513746, Published by Packt Publishing in May 2011.

Recently I switched to self publishing and my third book: 'Open Shading Language for Blender', distributed by Smashwords, major retailers and Blender Market, was the first result.

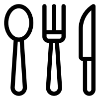
My latest book is called 'Creating add-ons for Blender' which gives a broader and more gentle introduction to Blender add-ons than this book.

I maintain a blog on Blender related things, 'Small Blender Things' (blog.michelanders.nl) and I keep an eye on the coding forums at http://www.blenderartists.org/forum/ where you can also contact me via private message if you like, my nickname there is 'varkenvarken'.

I also offer some Blender add-ons on BlenderMarket, the first one was called WeightLifter, a vertex group tool, but now accompanied by SpaceTree Pro, a environment aware tree modeler, and IDMapper, a tool to create ID-maps based on smart heuristics. If you like you can have a look at my Blender Market store to see what's on offer.

I live in a small converted farm in the southeast of the Netherlands where we raise goats for a hobby. We also keep a few chickens and the general management of the farm is left to our cats. This arrangement leaves me with with enough time to write the occasional book.

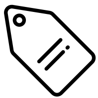
Create a property selection drop down



Intro

In many situations you might want your operator to let the user pick something from a list of choices. If these choices refer to Blender items, like objects, meshes, uv-maps, bones, modifiers, etc. we can use the **`prop_search()`** function of a layout object to tailor the **`draw()`** function of our operator in a single line of code. In doing so we no only create an easy to use list of choices but also prevent the user from the mistakes that are possible when entering a choice by hand.

In this recipe we create a simple operator that lets you select an object and that will then position the active object right next to this chosen object on the x-axis. The selection box that lets the user choose an object is presented with the help of the **`prop_search()`** function.



operator properties, prop_search, draw, bpy_props_collection, StringProperty

Explanation

When you use a **`prop_search()`** function the result of the user selection needs to be stored in a **`StringProperty`**

```
    other = StringProperty(name="Other object")
```

To control the way the operator properties are displayed we provide an implementation of the **`draw()`** function in our operator class. The string property doesn't need to be displayed but the search widget itself does

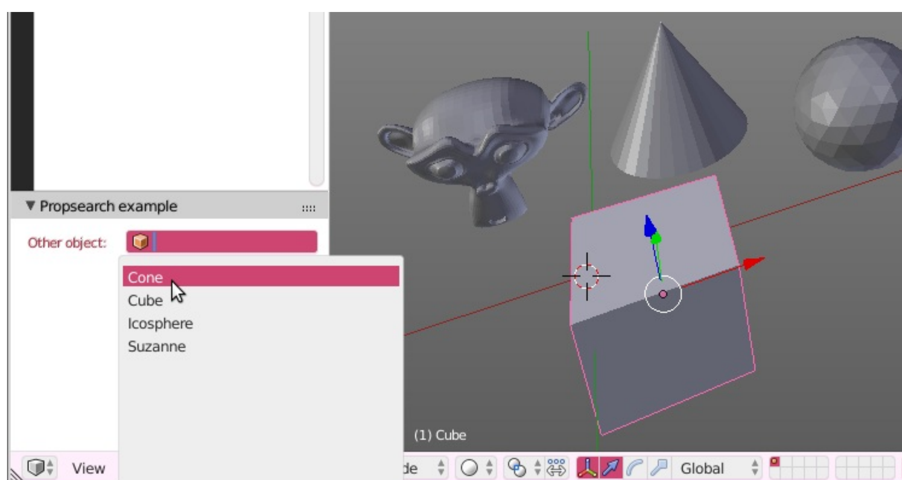
```
    def draw(self, context):
        layout = self.layout
```

Adding a widget that lets the user select an object or anything else from a

bpy_props_collection is done with the **prop_search()** function. This function takes a reference to the current operator, the name of the **StringProperty** to store the selection in (in this case '**other**'), a reference to an object and the name of the **bpy_props_collection** to list. In this case we want the user to select an object so our basis is **bpy.data** and the attribute we interested in is **objects**

```
        layout.prop_search(self,'other',bpy.data,'objects')
```

In this case this will result in a list of objects that will displayed in a drop down. The actual choice (the name of the chosen object) will be stored in the **other** property.



This is a very versatile way to present choices because many things in Blender are of the type **bpy_props_collection**. For example to present the user with a list of uv-maps for the active mesh object, you could simply write **layout.prop_search(self,'uvmap',context.object.data,'uv_layers')** assuming of course you have a **StringProperty** defined in your operator that is called '**uvmap**'.

Now with the selection of another object available, the **execute()** function can use this information to retrieve the actual chosen object

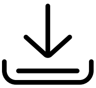
```
    def execute(self, context):
        if self.other in bpy.data.objects:
            other = bpy.data.objects[self.other]
```

The final step is to alter the **location** of the active object to put it right beside the chosen object

```
            ob = context.active_object
            ob.location = other.location
            ob.location.x += 1
        return {"FINISHED"}
```

Key points

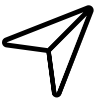
- You can provide a drop down selection of any **`bpy_props_collection`** in Blender
- Many collections of items in Blender are of the type **`bpy_props_collection`** including all the items in **`bpy.data`** and things like vertex groups or uv-maps in objects
- A **`StringProperty`** is needed to hold the result of the user selection
- The **`prop_search`** methoud of a Layout object will take care of the presentation



Code

If you install this add-on a menu item **`Object→Propsearch example`** will be added the 3d-view in object mode.

GitHub: **`propsearch.py`**



See also

Adding persistent properties to objects

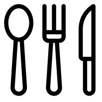
Refs

Property collections and the property selection drop-down

https://docs.blender.org/api/blender_python_api_current/bpy.types.bpy_prop_collection

https://docs.blender.org/api/blender_python_api_current/bpy.types.UILayout.html?bpy.types.UILayout.prop_search#bpy.types.UILayout.prop_search

Create a modal operator



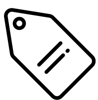
Intro

A modal operator is an operator that keeps on running after it is created. Unlike a non-modal operator that performs some specific function and then terminates, a modal operator can keep on interacting with the user. Examples of modal operators are the grab/move and scale operators.

To perform this interaction a modal operator should have a **`modal()`** method that is called every time an event occurs. Examples of events are mouse movements or key presses but timers can also create events at fixed intervals.

If a modal operator is added to a menu it starts its activity when the **`invoke()`** method is called and the operator is registered to receive events.

In this simple recipe we will create a modal operator that once started, keeps on showing each event in the area header until the right mouse button is clicked or the `Esc-key` pressed. This way you can get an idea of what kind of events you may act upon.



modal operator, event

Explanation

The **`invoke()`** function of a modal operator is called when someone clicks a modal operator that has been added to a menu. It can perform any initialization activity but what really makes an operator modal is when it registers itself as a modal handler with the window manager. This will cause its **`modal()`** method to be called each time some event occurs. Note that the **`invoke()`** method is also passed an event, which makes it possible to bind a modal operator to different shortcut keys for example and perform an initialization based on the actual shortcut key pressed, because the event passed to the **`invoke()`** method is actually this first key press or mouse-click

```
    def invoke(self, context, event):
        context.window_manager.modal_handler_add(self)
        return {'RUNNING_MODAL'}
```

Once registered as a modal handler the **modal()** method is called each time an event occurs. The event argument contains a **type** and a **value** and additional information based on the type. For mouse events it will also have the mouse coordinates. Here we display a few relevant attributes in the area header so you can experiment to see what kind of values are produced by different events

```
    def modal(self, context, event):
        context.area.header_text_set(
            "event: {e.type} {e.value} ({e.mouse_x},{e.mouse_y})"
                        .format(e=event))
        context.area.tag_redraw()
```

A common convention for modal operators is to cancel their activity if either the Esc-key is pressed or the right mouse button is clicked. Here we check for this scenario and reset the area header before returning **{'CANCELLED'}** to signal we are done. (Note that the return value is a **set** with a single string)

```
        if event.type in {'RIGHTMOUSE', 'ESC'}:
            context.area.header_text_set()
            context.area.tag_redraw()
            return {'CANCELLED'}
```

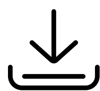
For all other events we just keep on running, something we signal by returning **{'RUNNING_MODAL'}**

```
        return {'RUNNING_MODAL'}
```

In another recipe we will see that it is also possible to pass through some events so that other operators may act on them.

Key points

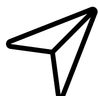
- A modal operator installs a modal handler that keeps on acting after the initial call
- This is commonly the **`modal()`** method of the operator itself
- This method is called every time an event happens
- Events can be mouse actions, key presses or timer events
- A modal handler may remove itself by returning **`{'CANCELED'}`** or keep on running by returning **`{'RUNNING_MODAL'}`**



Code

This add-on installs a modal handler when you click on **`Add→Mesh→Modal Operator`** in the 3d-view in object mode.

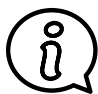
GitHub: **`modaloperator.py`**



See also

Using pass through in a modal operator

Show hints in an area header



Refs

More on the difference between the Operator methods **`poll()`**, **`invoke()`**, **`execute()`**, **`draw()`** & **`modal()`**

http://blender.stackexchange.com/questions/19416/what-do-operator-methods

Index

A

add-on distribution 1

add-on preferences 1

animation 1

area 1

area header 1

attributes 1

B

base 1

bezier splines 1

BMesh 1, 2

bpy_props_collection 1

C

cursor 1

curve objects 1

custom data layer 1

custom property 1

Cycles 1, 2

D

draw 1

draw handler 1

driver 1

E

edge 1, 2

EnumProperty 1

event 1

F

face 1, 2

foreach_get 1

foreach_set 1

frame 1

H

hotkey 1

I

icon 1

intersection 1

K

kd-tree 1

keymap 1

L

layers 1

library 1

loop 1

M

material 1

median 1

menu 1, 2

Mesh 1

modal 1

modal operator 1, 2, 3, 4, 5

modifier 1, 2, 3

module 1

N

Node Wrangler 1

nodes 1, 2, 3

numpy 1

O

object 1

object coordinates 1

object parenting 1

object_utils 1

OpenGL 1, 2, 3, 4

operator 1

operator properties 1

P

package 1

Panel 1, 2, 3

parameterized object 1

parenting objects 1
particle system 1, 2
pass through 1
pixel value 1
preview collection 1
progress bar 1
prop_search 1
property 1
PropertyGroup 1
pydriver 1

R

ray-casting 1
region 1
report() 1

S

scene 1
screen 1
screen coordinates 1, 2
separator 1
shader 1
shortcut 1
space 1
StringProperty 1, 2
submenu 1
subsurface modifier 1

T

thumbnail 1

V

Vector 1

vertex 1, 2

vertex group 1

W

window 1

window manager 1

world coordinates 1, 2, 3

world matrix 1

This is a sample of my book

Blender Add-on Cookbook

A cookbook with useful programming recipes for Blender add-ons

---

If you like to buy a full copy, please visit my Blender Market store:

https://blendermarket.com/creators/varkenvarken

In the store you will also find a more gentle introduction to add-ons called

Creating Add-ons for Blender, a practical primer.