

# CONTENTS

<b>1. Installation</b>	2
a. Requirement	
b. Install	
<b>2. Control Panel</b>	3
<b>3. Modifier Editor</b>	4
a. Link and Deep Link	
b. Reference Driver	
c. Batch Menu	
<b>4. Driver Editor</b>	16
<b>5. Mesh Editor</b>	17
<b>6. Keymap</b>	19
a. Value definition	
b. Conflict	
c. Priority	
<b>7. Basic Control</b>	23
a. Text Box	
b. Value Box	
c. Calculator	
d. Function list	
<b>8. Task Bar</b>	51
<b>9. Theme and Color</b>	52
<b>10. About</b>	53
a. Bug Report	
b. Known issues and limitations	

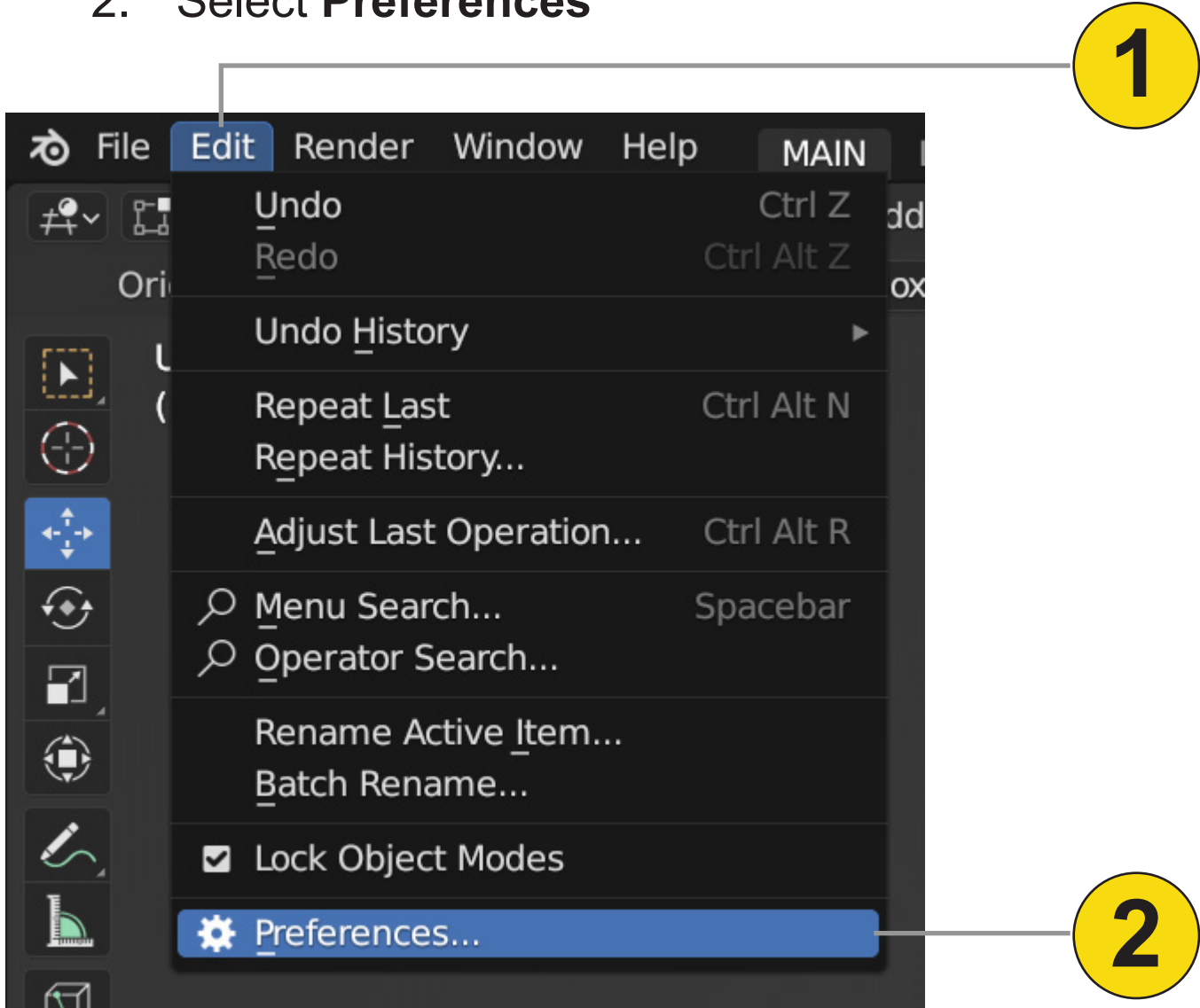
# INSTALLATION

## Requirement

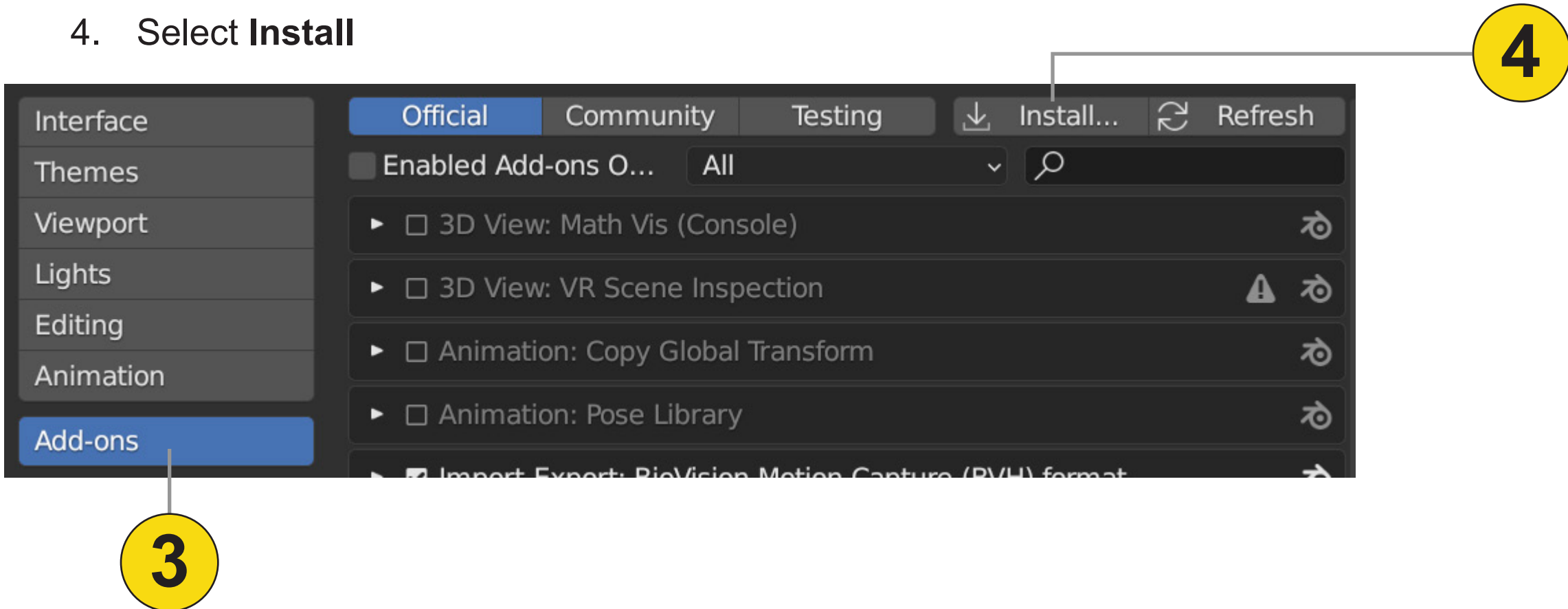
- 1. Blender 2.92 - 3.6

## Install

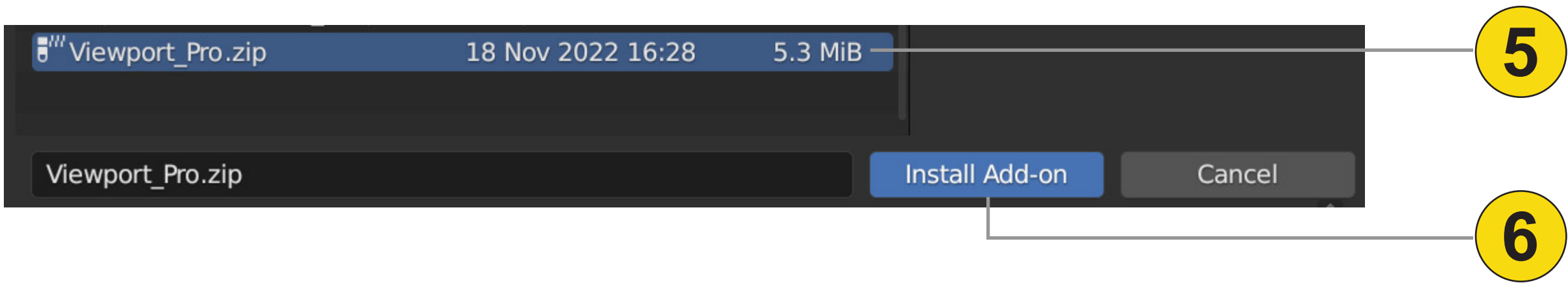
- 1. Select **Edit**
- 2. Select **Preferences**



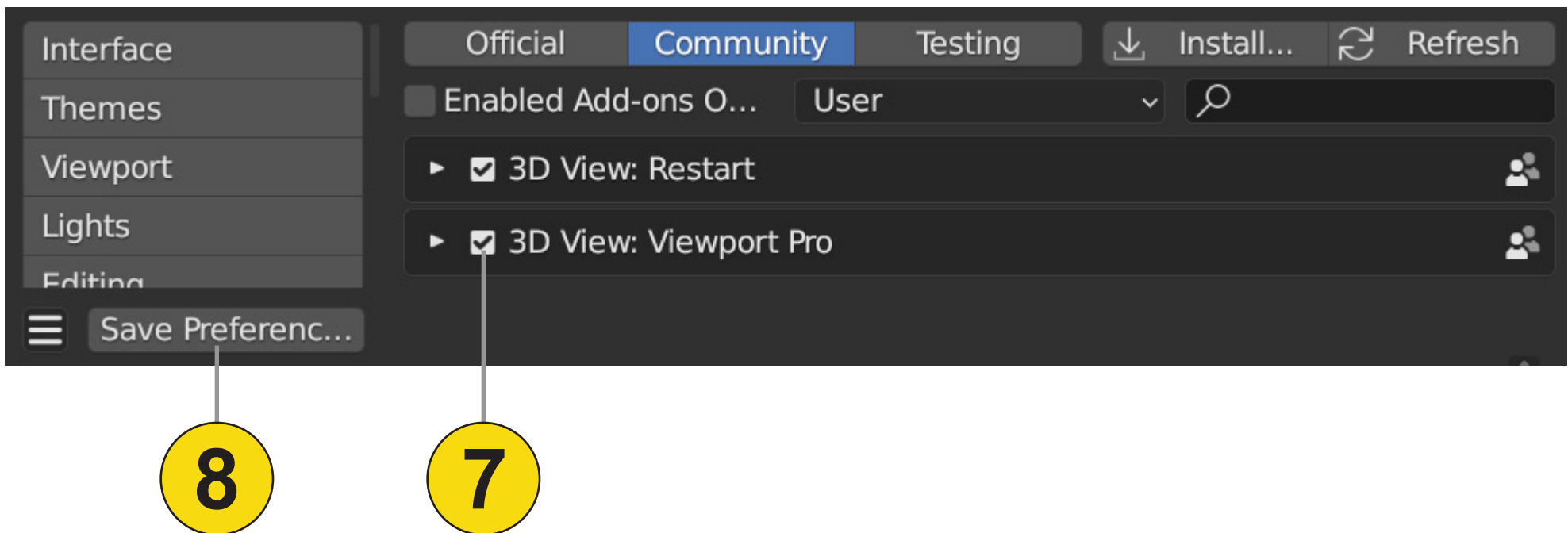
- 3. Select **Add-ons**
- 4. Select **Install**



- 5. Select the zip file named **Viewport\_Pro**
- 6. Select **Install Add-on**



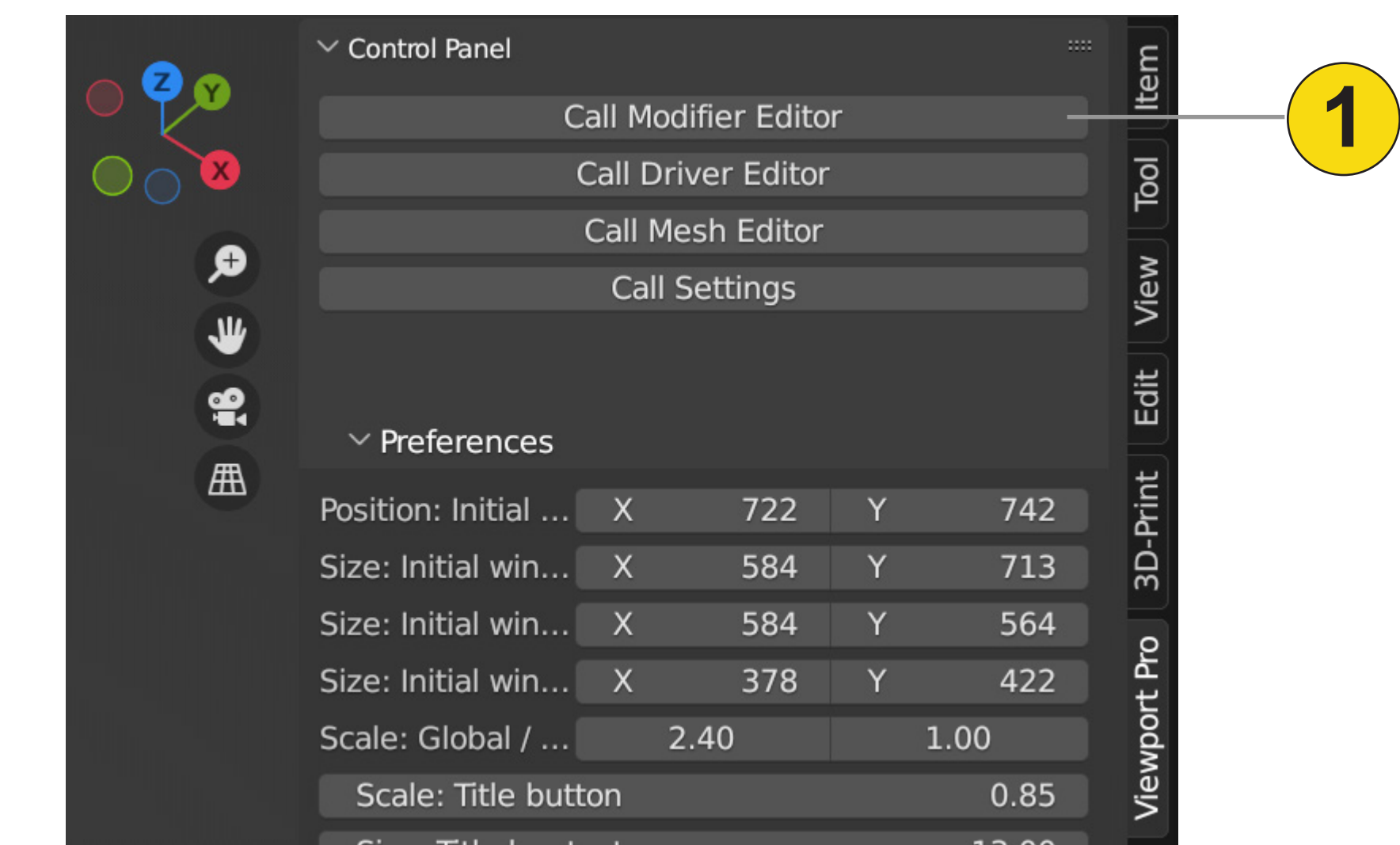
- 7. Enable the checkbox
- 8. Save Preferences



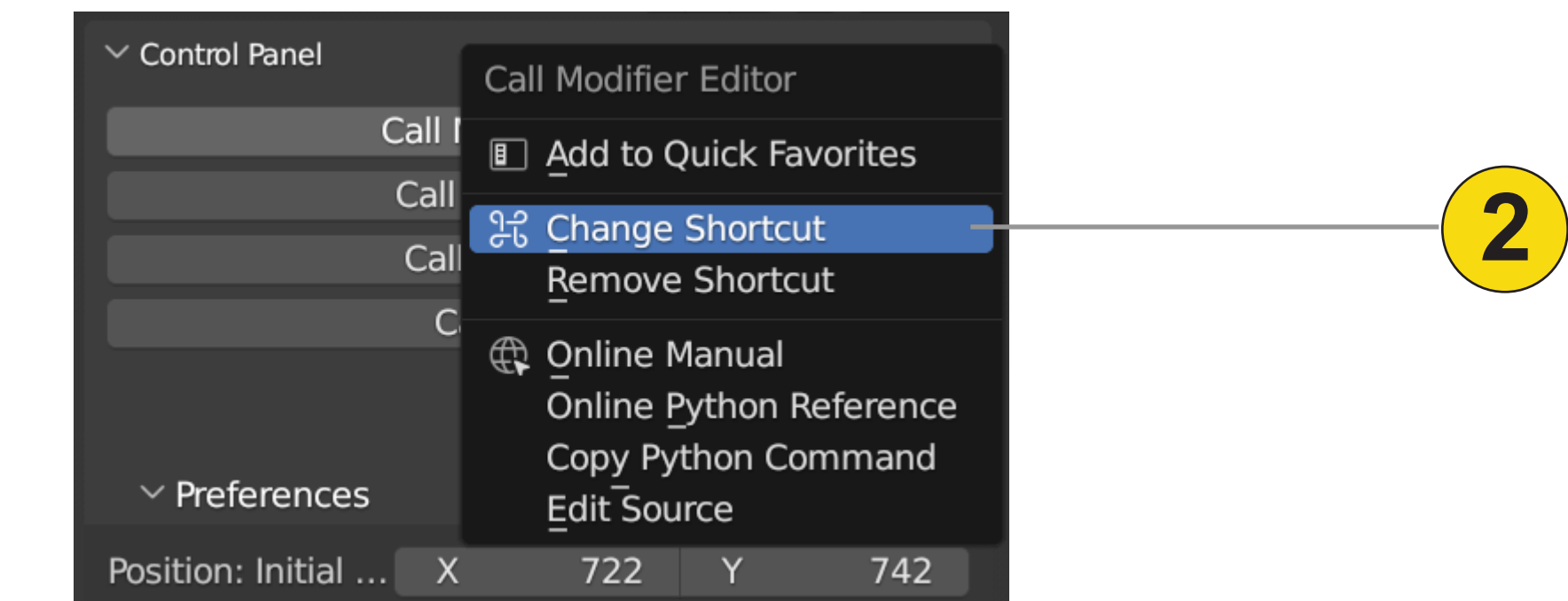
# CONTROL PANEL

You can edit most add-on settings from the Viewport N-panel.

1. Call the context menu (*Blender default shortcut: Right click*)

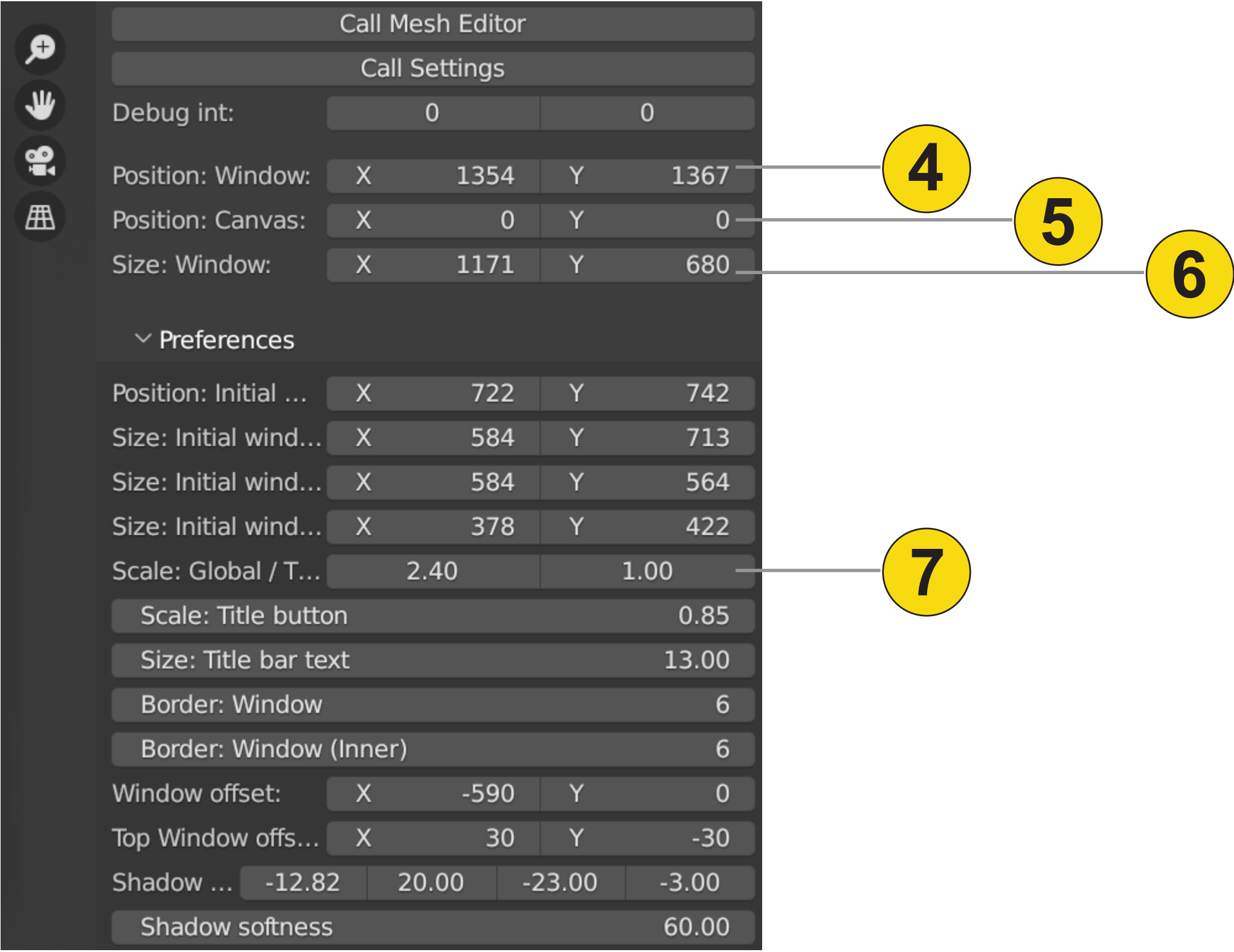
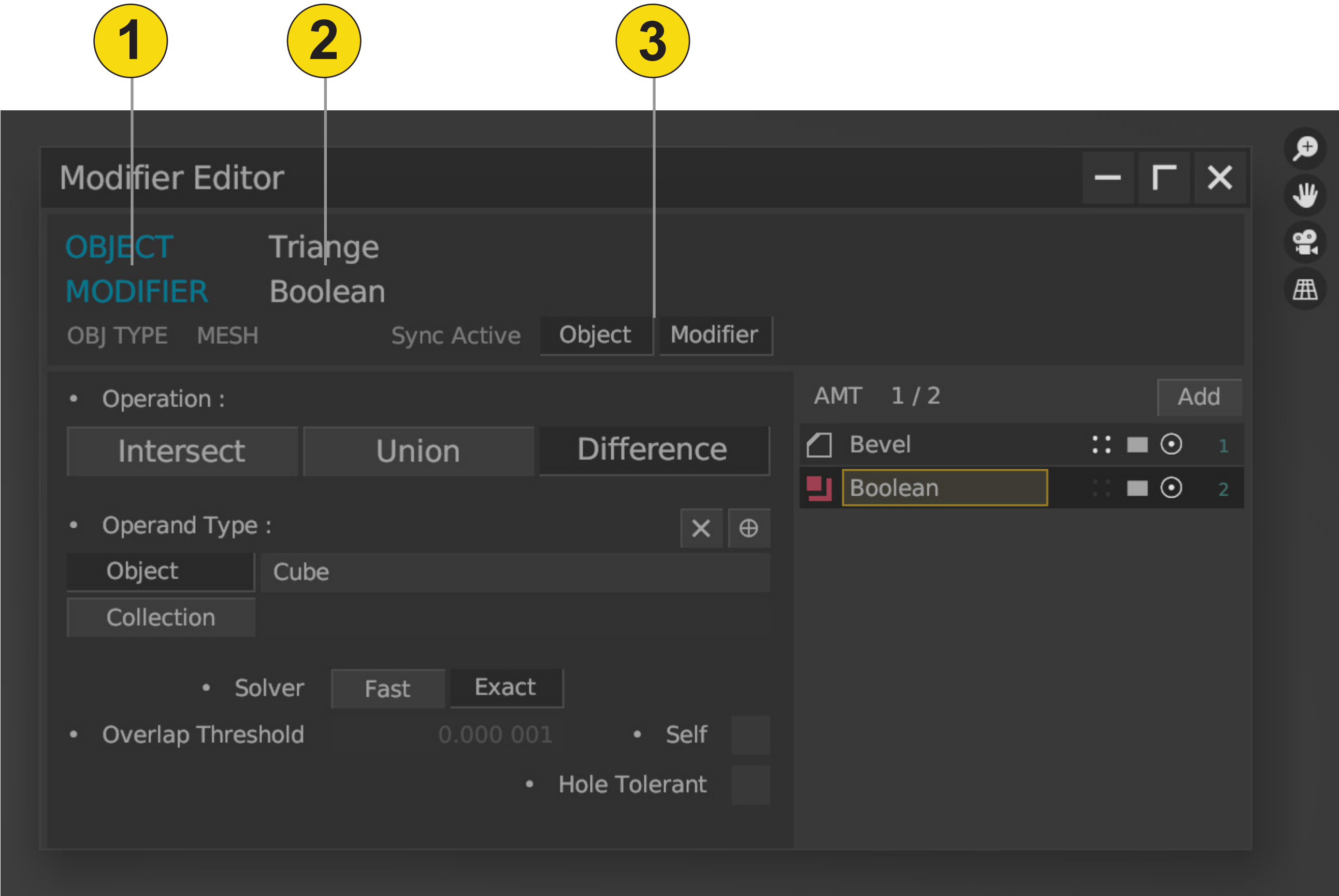


2. Assign Shortcut to operator



# MODIFIER EDITOR

The Modifier Editor displays the object's active modifier.



1. Select the *OBJECT / MODIFIER* button to change the current display object / modifier.

Default Keymap 1		Select Button (Press)
Keystroke	Left Mouse	
Value	Press	
End Value	-	
Region	Button	

2. Double-click the object / modifier name to rename the object / modifier.

Default Keymap 1

Rename

Keystroke	Left Mouse
Value	Double Press
End Value	-
Region	Button

Default Keymap 2

Rename

Keystroke	F2
Value	Press
End Value	-
Region	Button

3. When *Sync Active* is enabled, the editor’s object / modifier data will follow the active object / modifier in the 3D viewport.

Default Keymap 1

Select Button (Press)

Keystroke	Left Mouse
Value	Press
End Value	-
Region	Button

4. Control the active window position. You can also control the window position through the keymap.

Default Keymap 1

Title Bar Move

Keystroke	Left Mouse
Value	Press
End Value	Release
Region	Title Bar

5. Controls the canvas position of the active window.

Default Keymap 1

Pan Global

Keystroke	Right Mouse
Value	Drag
End Value	Release
Region	Active window

6. Control Window size.

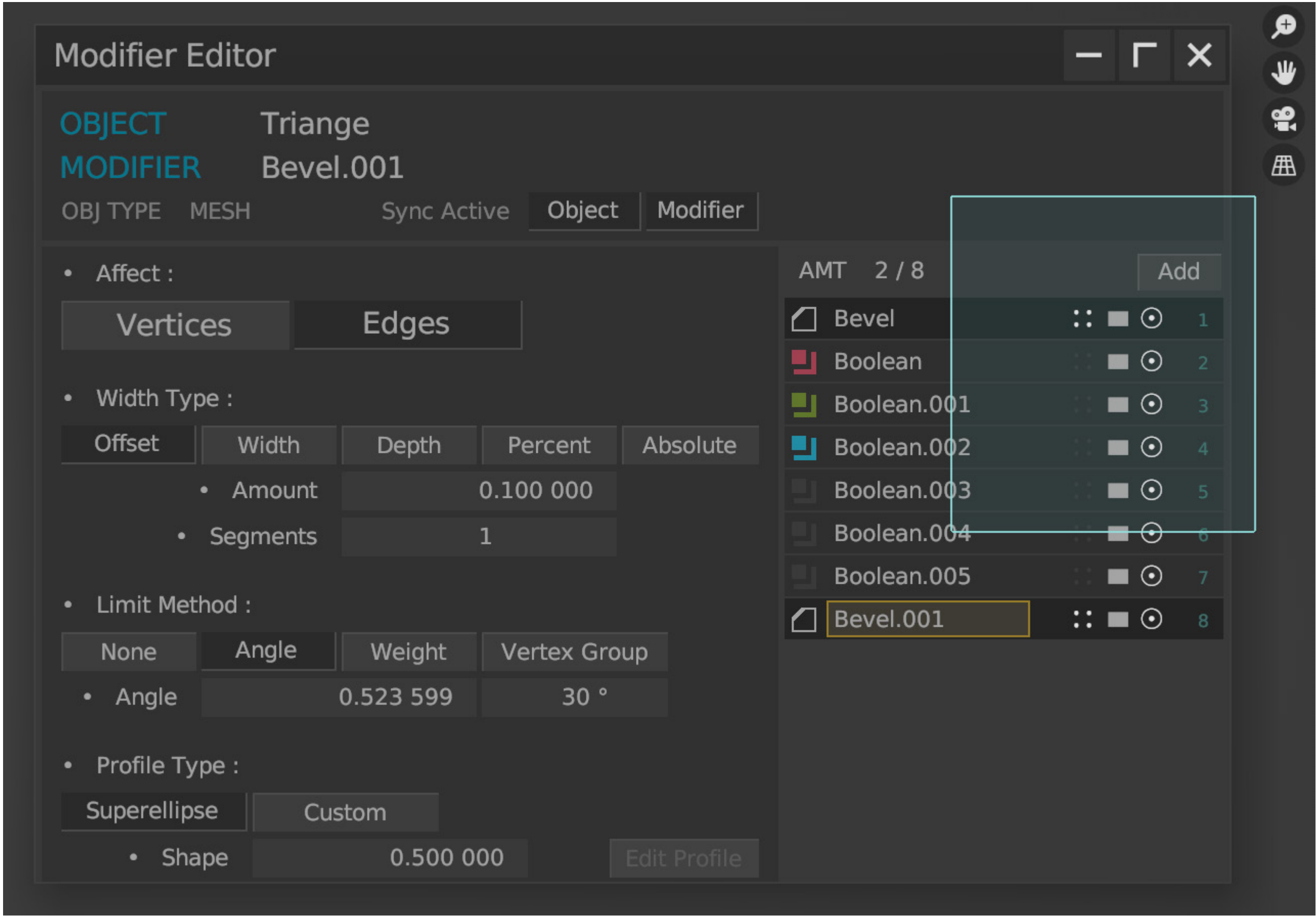
Default Keymap 1

Window Resize

Keystroke	Left Mouse
Value	Press
End Value	Release
Region	Active window border

7. Adjust window scale from N-panel (*Ignore Blender Resolution Scale and System Scale*).





1. Select the modifier to change the active modifier.

Default Keymap 1		Select (Modifier Editor)
Keystroke	Left Mouse	
Value	Release	
End Value	-	
Region	Modifier list	

2. Select the modifier to change the active modifier and keep select.

Default Keymap 1		Select Extend (Modifier Editor)
Keystroke	Left Ctrl Left Mouse	
Value	Release	
End Value	-	
Region	Modifier list	

3. Dragging a selection box to select modifiers.

Default Keymap 1		Select Box
Keystroke	Left Mouse	
Value	Drag	
End Value	Release	
Region	Modifier list	

4. Dragging a selection box to extend select modifiers.

Default Keymap 1		Select Box Extend
Keystroke	Left Ctrl Left Mouse	
Value	Drag	
End Value	Release	
Region	Modifier list	

5. Move the canvas from the list of modifiers.

Default Keymap 1		Pan (Modifier Editor)
Keystroke	Middle Mouse	
Value	Drag	
End Value	Release	
Region	Modifier list	

Default Keymap 2		Pan (Modifier Editor)
Keystroke	Space	
Value	Press	
End Value	Press	
Region	Modifier list	

6. Change the modifier name.

Default Keymap 1Rename

Keystroke	Left Mouse
Value	Double Press
End Value	-
Region	Modifier list

Default Keymap 2Rename

Keystroke	F2
Value	Press
End Value	-
Region	Modifier list

7. Select all modifiers, deselect all inactive modifiers when all modifiers are selected.

Default Keymap 1Select All

Keystroke	A
Value	Press
End Value	-
Region	Modifier list

8. Activate top modifier.

Default Keymap 1Set Active Up

Keystroke	Mouse wheel up
Value	Press
End Value	-
Region	Active Window

9. Activate bottom modifier.

Default Keymap 1Set Active Down

Keystroke	Mouse wheel down
Value	Press
End Value	-
Region	Active Window

10. Activate top modifier (Extend).

Default Keymap 1Set Active Up Extend

Keystroke	Left Ctrl	Mouse wheel up
Value	Press	
End Value	-	
Region	Active Window	

11. Activate top modifier (Extend).

Default Keymap 1Set Active Down Extend

Keystroke	Left Ctrl	Mouse wheel down
Value	Press	
End Value	-	
Region	Active Window	

12. Move the modifier up.

Default Keymap 1Modifier Move Up

Keystroke	Left Shift	Mouse wheel up
Value	Press	
End Value	-	
Region	Active Window	

13. Move the modifier down.

Default Keymap 1Modifier Move Down

Keystroke	Left Shift	Mouse wheel down
Value	Press	
End Value	-	
Region	Active Window	

14. Delect all selected modifiers.

Default Keymap 1Modifier Delete

Keystroke	X
Value	Press
End Value	-
Region	Active Window

15. Apply all selected modifiers.

Default Keymap 1Modifier Apply

Keystroke	Left CtrlA
Value	Press
End Value	-
Region	Active Window

Modifier Editor

OBJECTPentagon

MODIFIERBoolean.009

OBJ TYPE MESH Sync Active Object Modifier

Operation :  
Intersect Union Difference

Operand Type :  
Object Plane  
Collection

Solver Fast Exact

Overlap Threshold 0.000 001 Self  
Hole Tolerant

AMT 6 / 32 Add

Boolean Boolean.002 Boolean.004 Boolean.006

Boolean.008 Boolean.010 Boolean.011 Boolean.012 Boolean.016 Boolean.017 Boolean.018 Boolean.019 Boolean.020

Multi Group 6 items

1 2 4 5 6 7 8 9 10 11 12 13 14

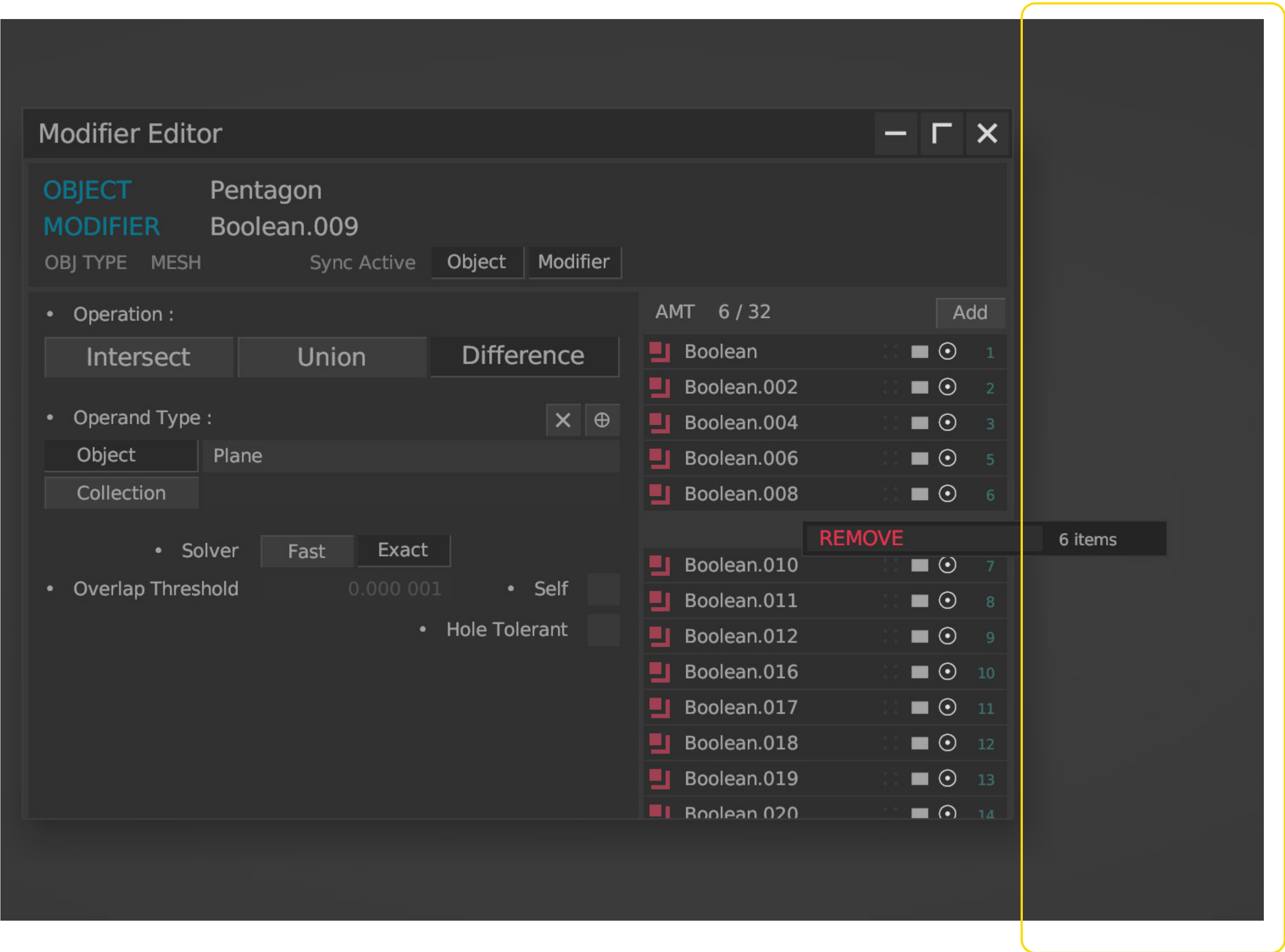
16. Change the order of modifiers. Move / Apply / Delete / Link the modifiers.

Default Keymap 1Sorting

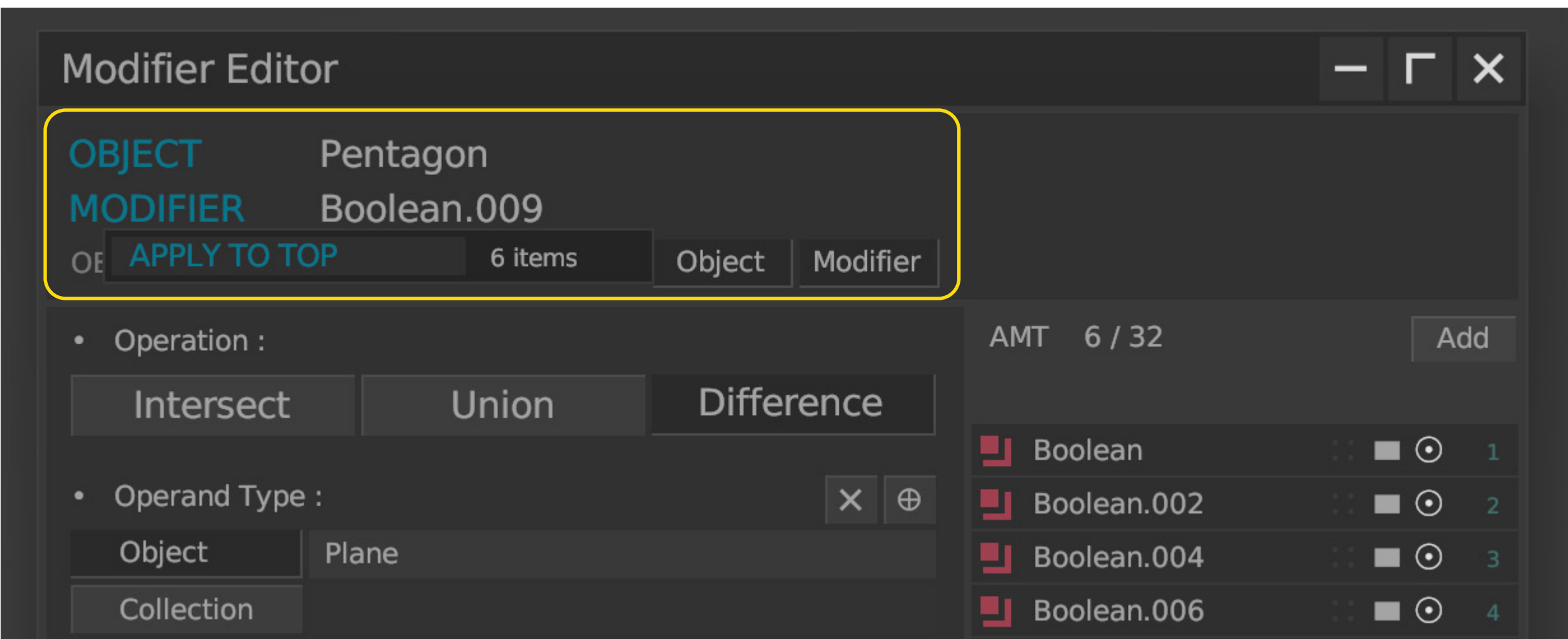
Keystroke	Left Mouse
Value	Release
End Value	Press
Region	Serial number



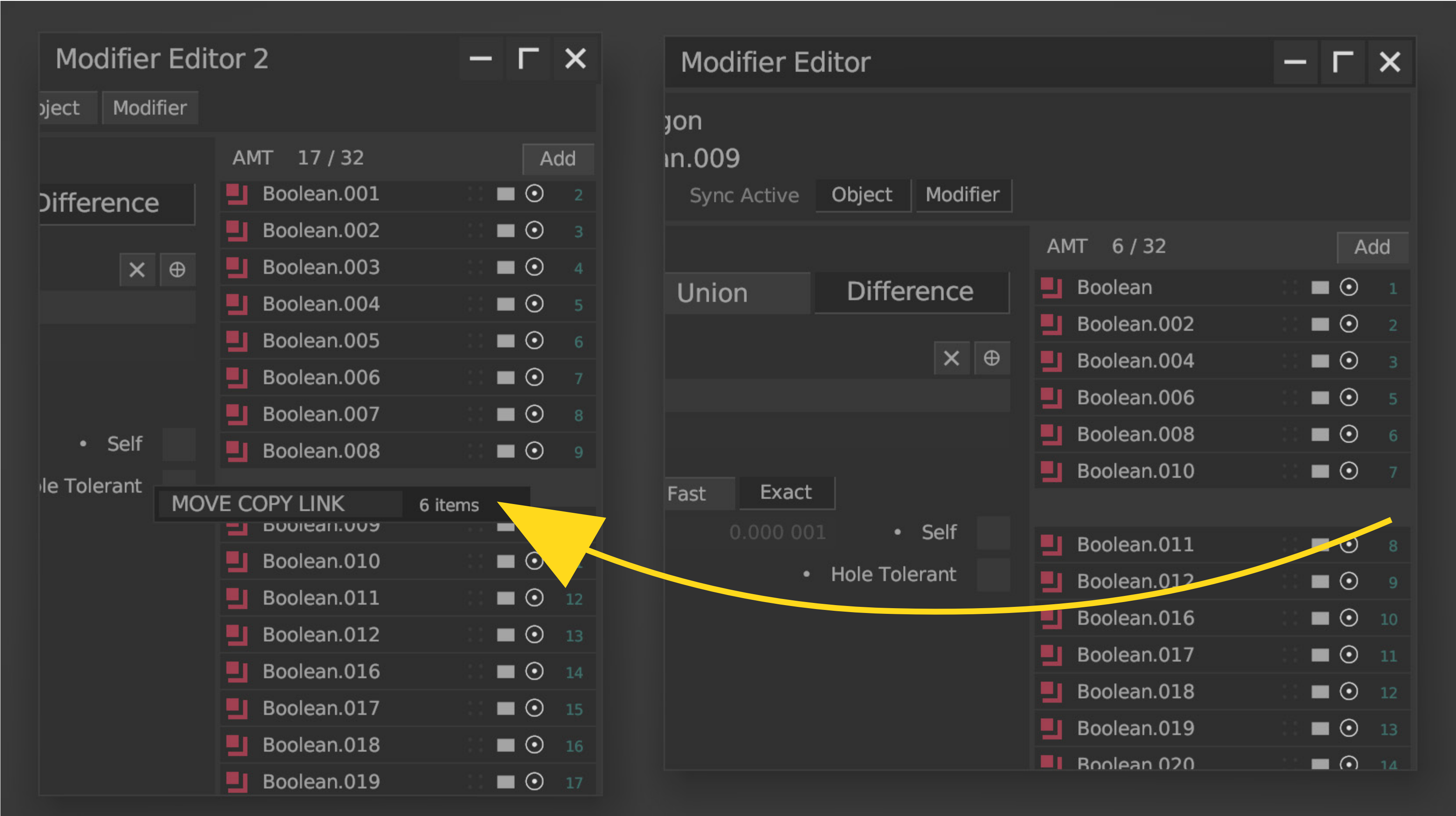
End the Operator to Delete the modifiers when the cursor is on the right side of the editor.



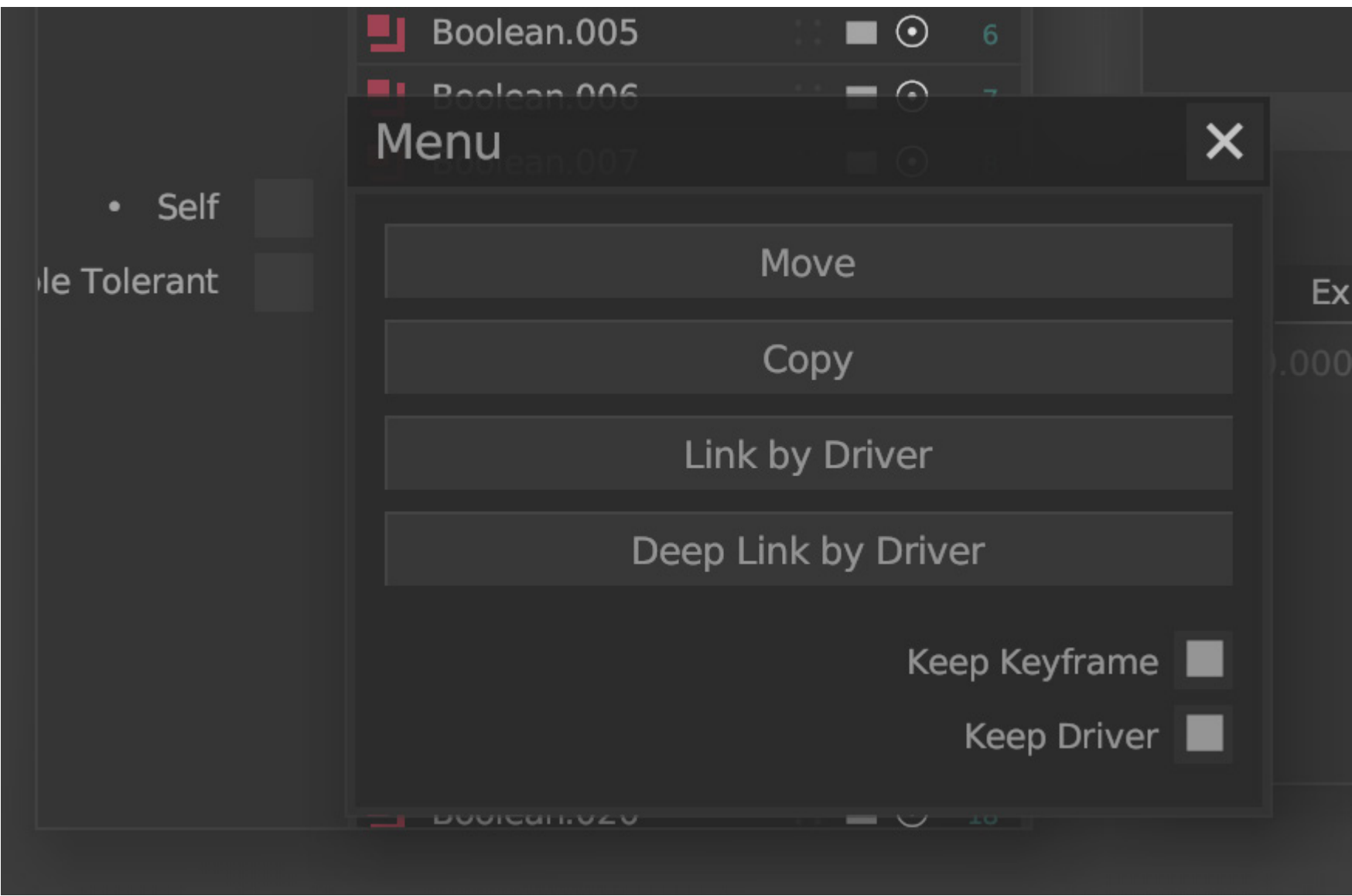
End the Operator to Apply the modifiers when the cursor is in the top area.




End the Operator to Move / Copy / Link the modifiers when cursor is on another editor.



## Link and Deep Link



**Link by Driver** - Reference to the original object, all data for this modifier will be controlled by the driver and updated automatically. In the example below, change the *Amount* of *Bevel\_A* from 0.45 to 1, then the *Amount* of *Bevel\_B* will become 1.



Modifier Editor

OBJECTTriangle

MODIFIERBevel\_A

OBJ TYPE MESH Sync Active Object Modifier

Affect :

VerticesEdges

Width Type :

OffsetWidthDepthPercentAbsolute

Amount0.450 000


Segments5

AMT 1 / 1

Add

Bevel\_A

::■⦿1



Modifier Editor 2

OBJECTRectangle

MODIFIERBevel\_B

OBJ TYPE MESH Sync Active Object Modifier

Affect :

VerticesEdges

Width Type :

OffsetWidthDepthPercentAbsolute

Amount0.450 000

Segments5

AMT 1 / 1

Add

Bevel\_B

::■⦿1

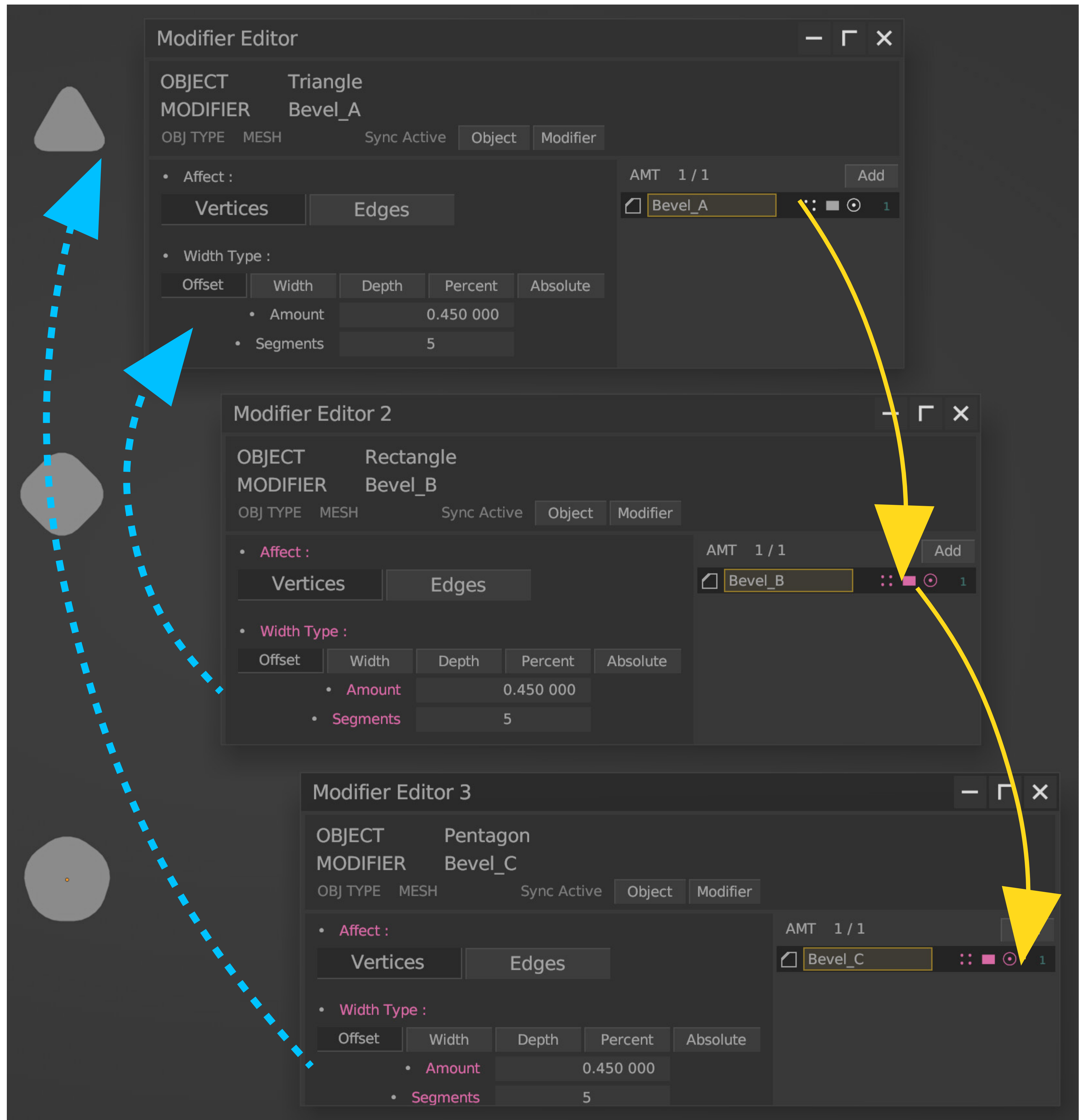
11



**Deep Link by Driver** - Reference to the deepest referenced object.

In the example below, since *Bevel\_B* is a reference to *Bevel\_A*, when *Bevel\_B* deep link to *Pentagon*, *Bevel\_C* will be a reference to *Bevel\_A*.

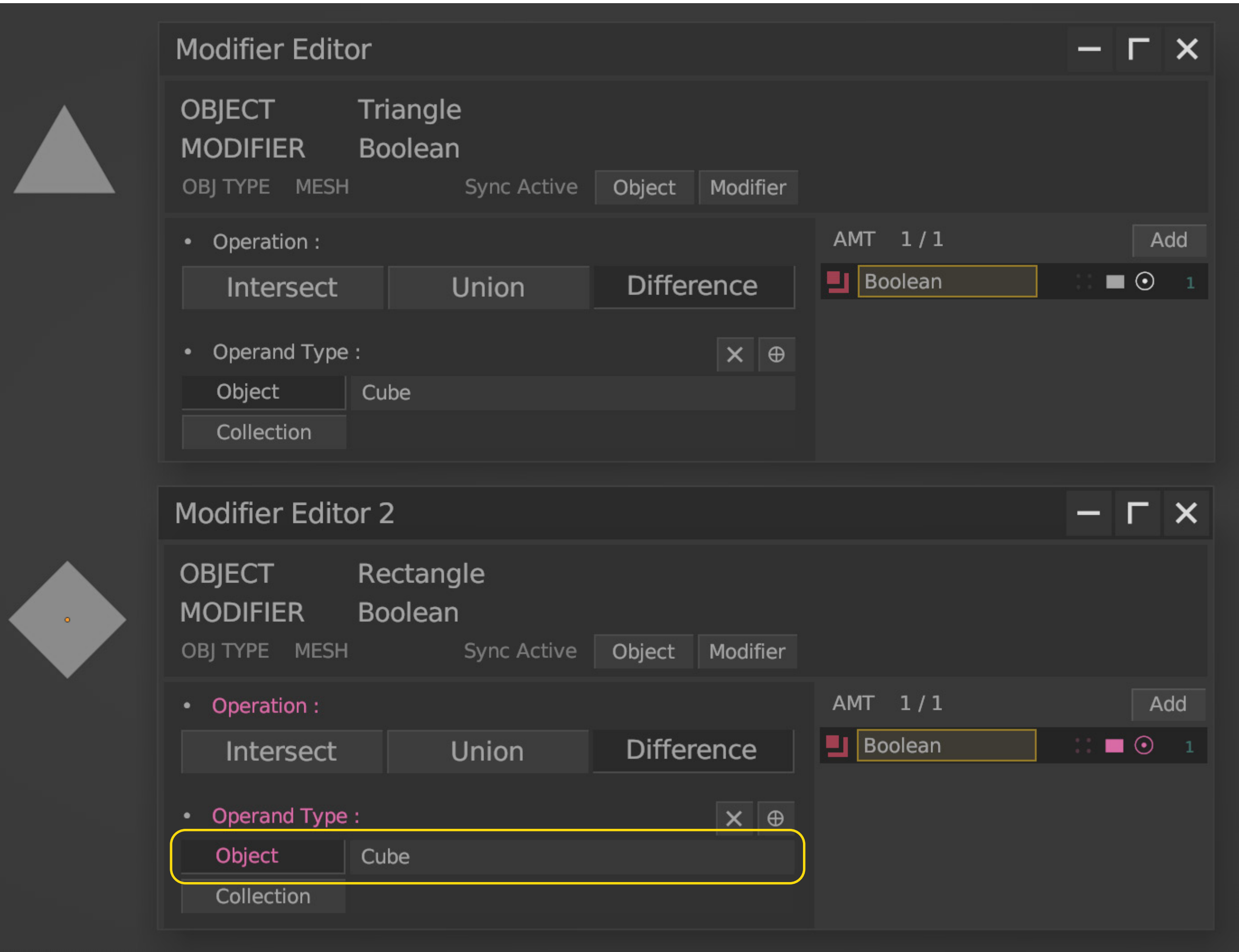
Change the *Amount* of *Bevel\_A* from 0.45 to 1, then the *Amount* of *Bevel\_B* and *Bevel\_C* will become 1, after deleting *Rectangle*, *Pentagon*'s link will continue to work.



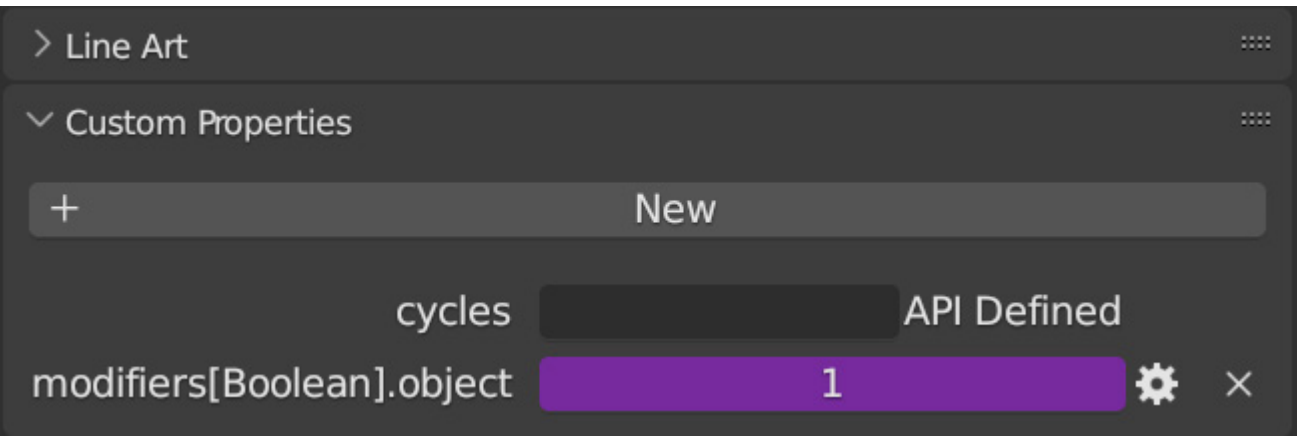


# Reference Driver

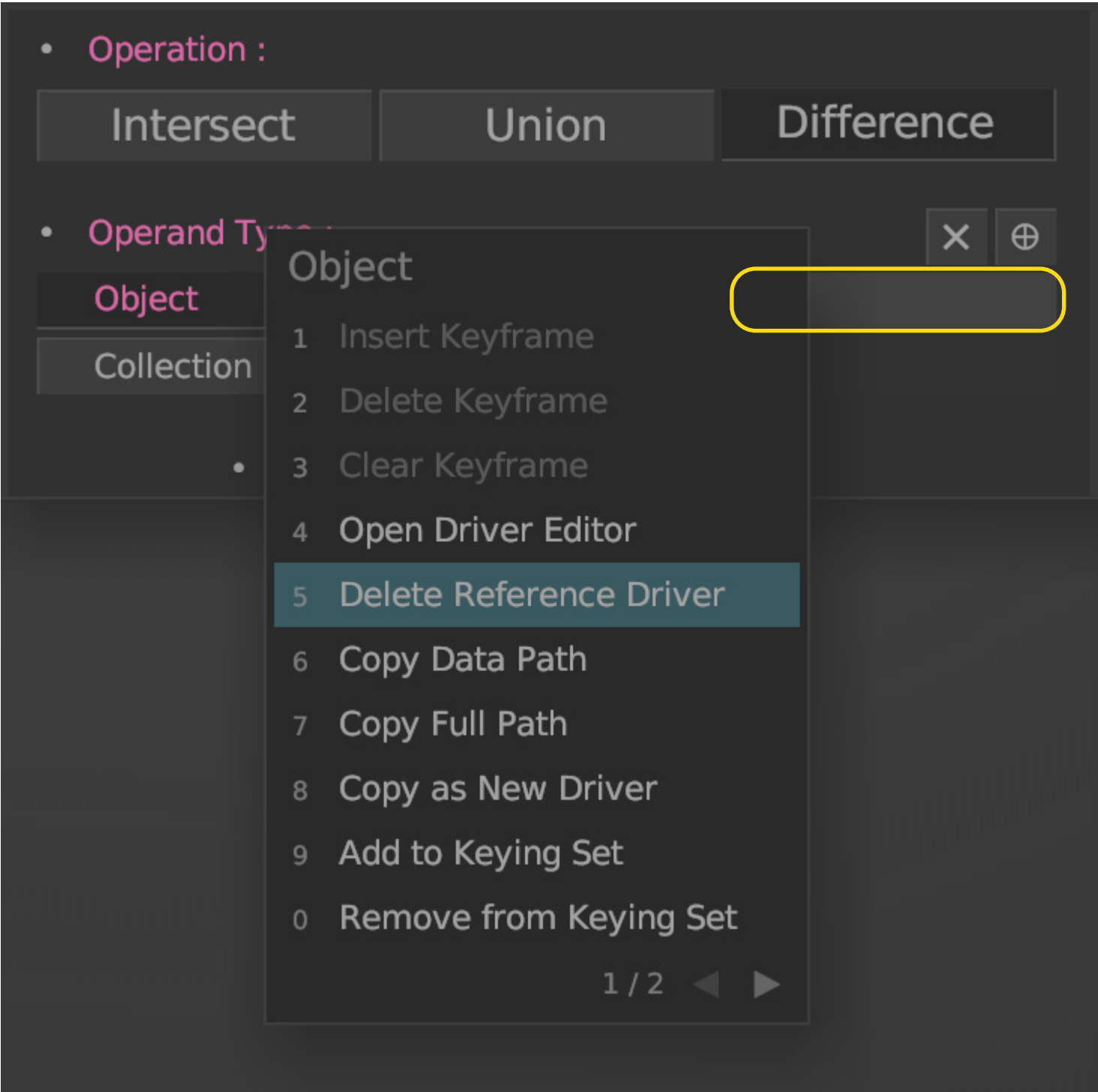
Driver / Keyframe cannot be added due to certain properties in blender (eg: Object of Boolean modifier). There is a new type of driver in this add-on.



When you Link the Boolean modifier, it will add a custom property for updating referenced properties.  
From the above example, when the Boolean property (Object) of Triangle changes, the Boolean property (Object) of Pentagon will change accordingly.



Open the Driver Editor or Invoke the context menu from the object name to Remove the Reference Driver and it will automatically remove the custom Property.



1. Call Context Menu.

Default Keymap 1		Context Menu
Keystroke	Right Mouse	
Value	Release	
End Value	-	
Region	Global	

Default Keymap 2		Context Menu
Keystroke	Application	
Value	Press	
End Value	-	
Region	Global	

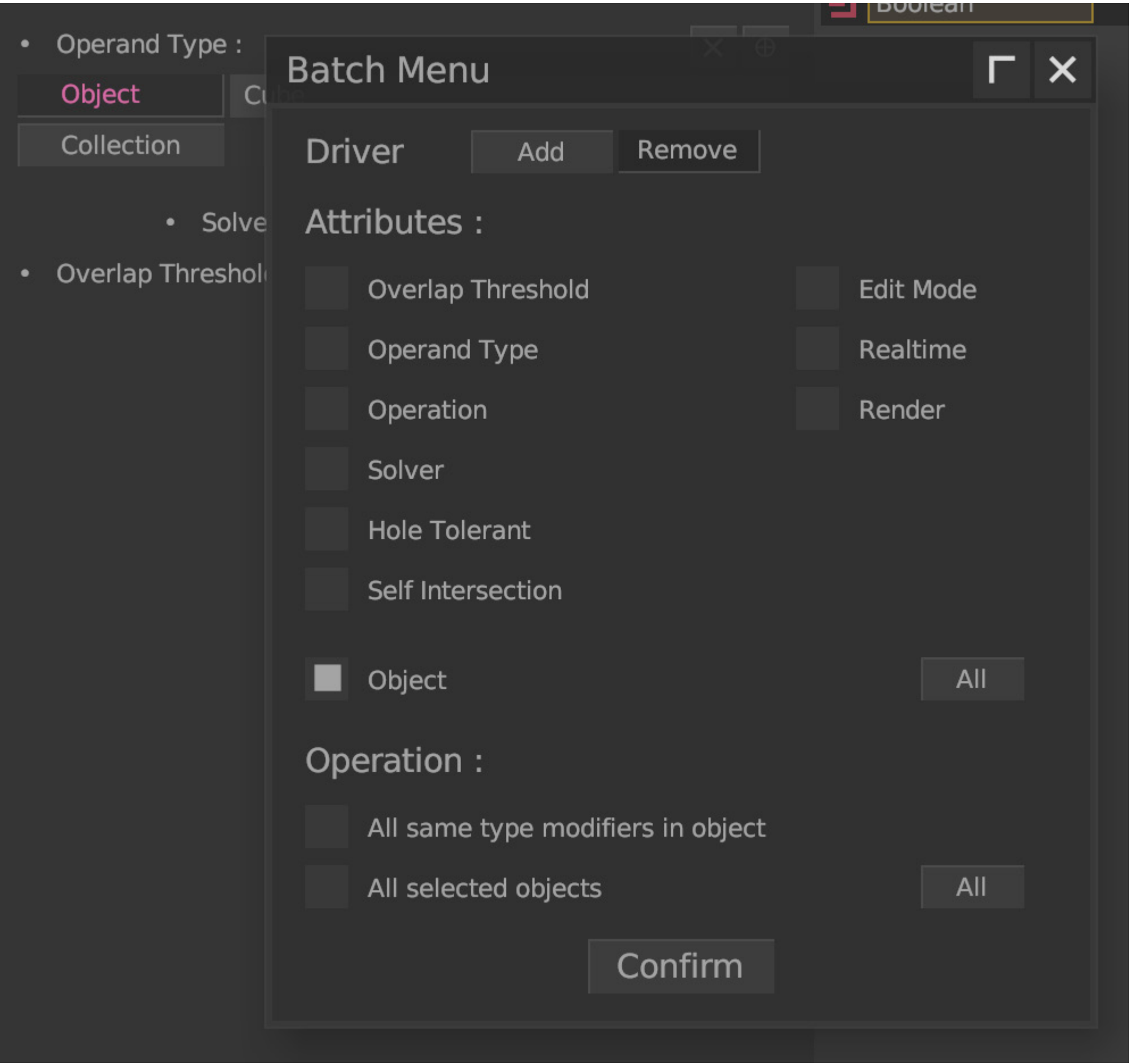
2. Context Menu Button.

Default Keymap 1		Button 5
Keystroke	5	
Value	Press	
End Value	-	
Region	Context Menu	

Default Keymap 2		Button 5
Keystroke	Numpad 5	
Value	Press	
End Value	-	
Region	Context Menu	

# Batch Menu

Invoke the Batch Menu from context menu or via keymap.

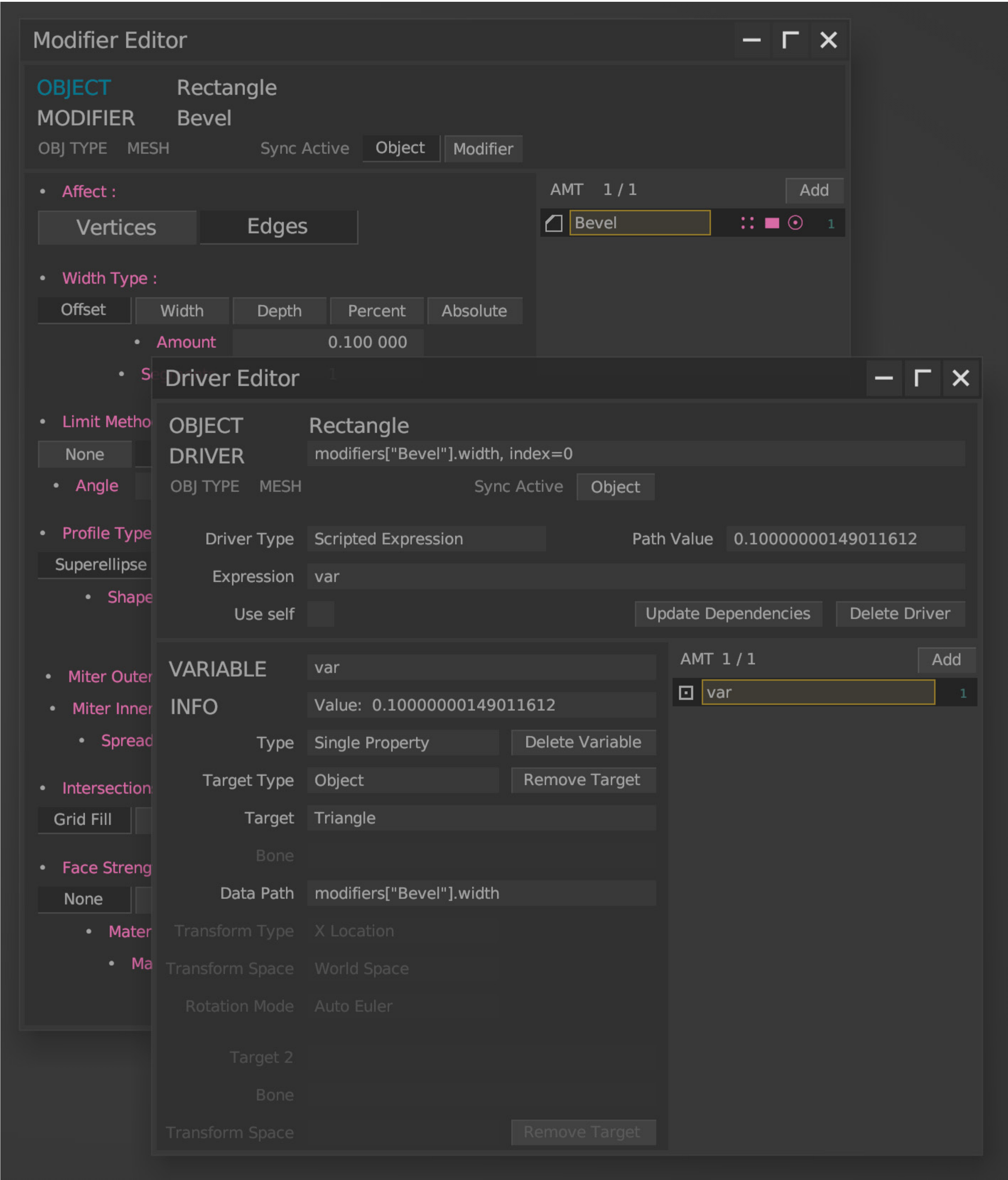


- 1. Invoke Batch Menu.

Default Keymap 1		Batch
Keystroke	Left Alt   A	
Value	Press	
End Value	-	
Region	Button	

# DRIVER EDITOR

Invoke Driver Editor from Properties of Modifier Editor or via N-Panel / Shortcut.  
If the Data Path of the Property does not have a driver, it will add the driver to the object.

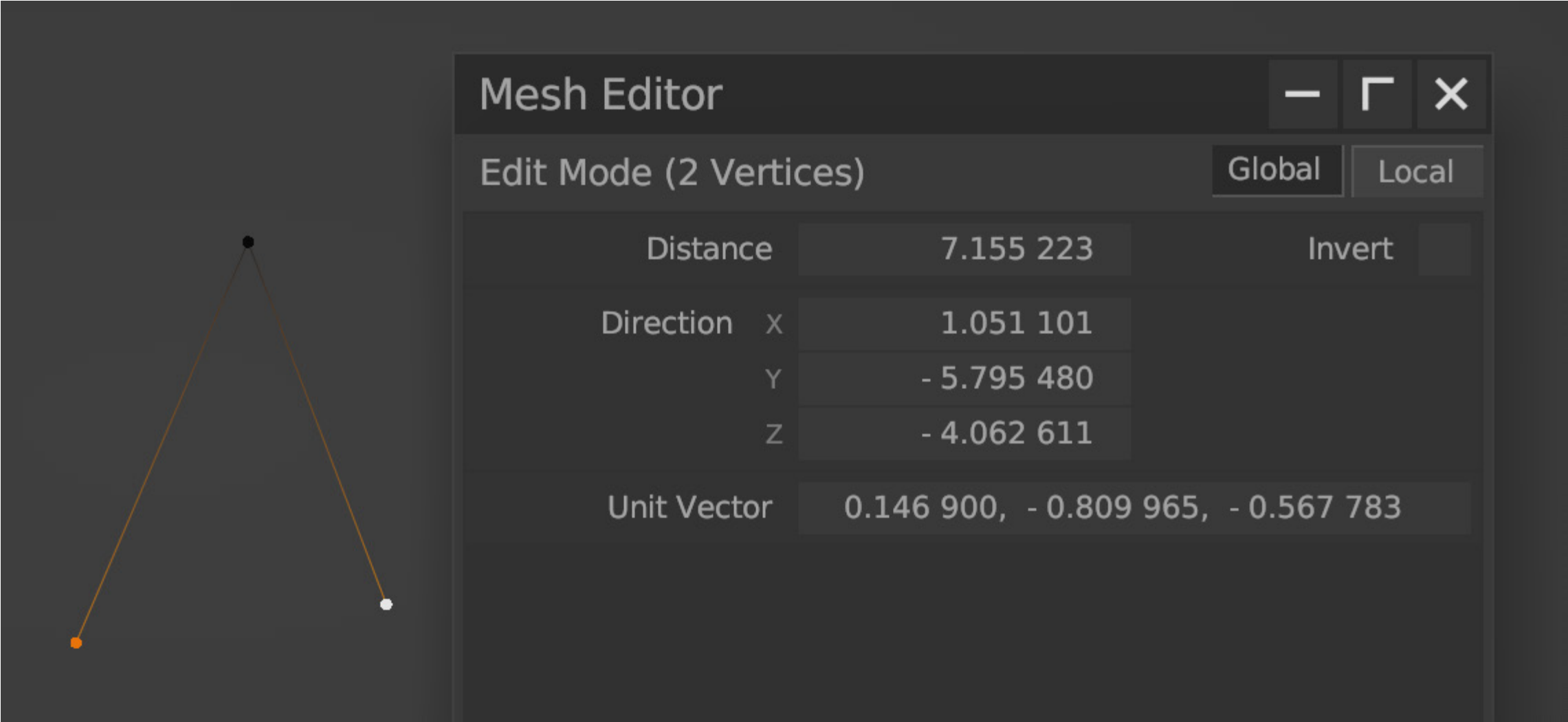


Driver variable can Sorting / Move / Copy like the Modifier Editor.



# MESH EDITOR

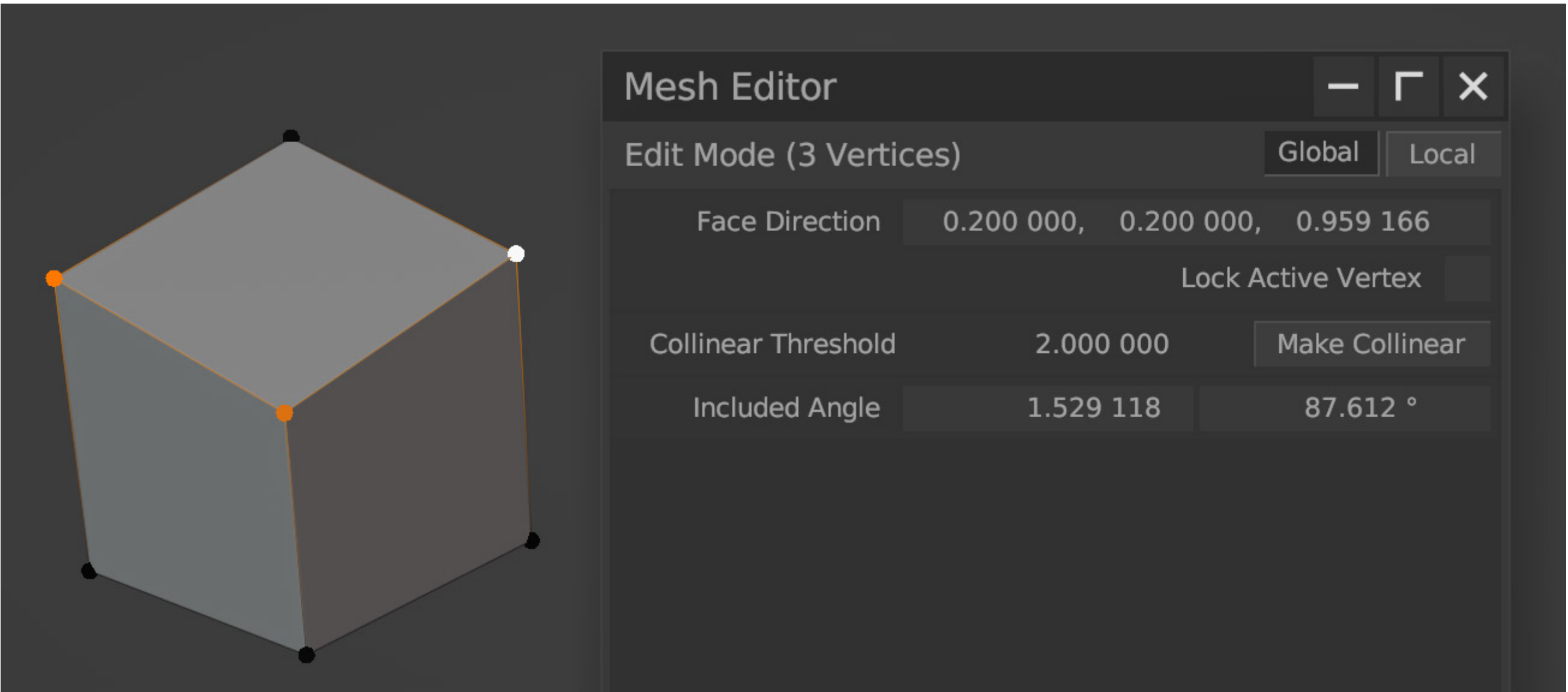
Invoke Mesh Editor via N-Panel / Shortcut.  
Mesh Editor allows users to edit distance / angle / normal etc. mesh data in Edit Mode.



**Distance**  
When edit the Distance between 2 points, it will keep the Direction and moving from the active point.

**Direction**  
The Direction of 2 vertices points to the active vertex by default.

**Unit Vector**  
When editing the directional unit vector, the distance between 2 points is maintained.

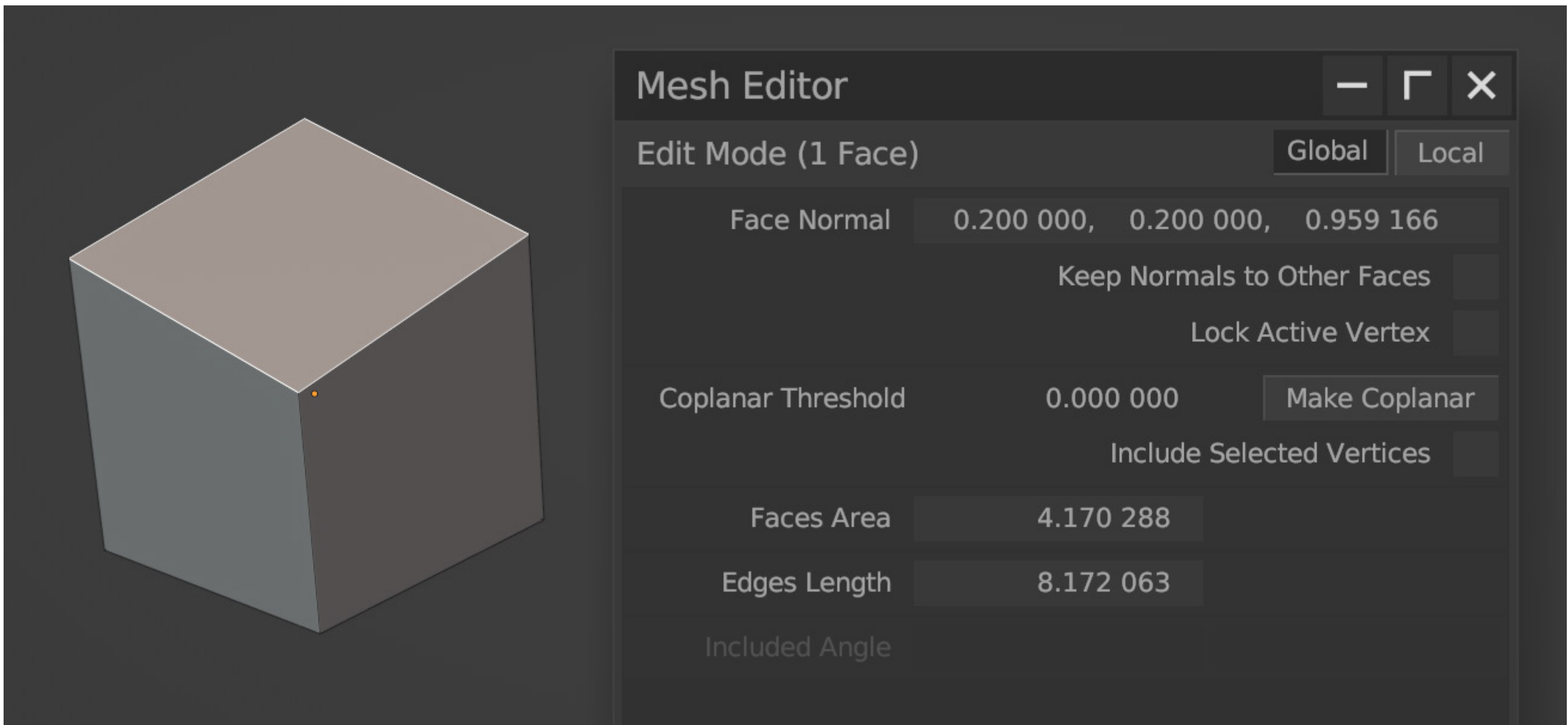


**Face Direction**  
Unit vector direction of Face Normal, based on vertex selection order.  
Properties: Lock Active Vertex.

**Collinear Threshold**  
In the collinear judgment under a specific accuracy, 0 is the most accurate.

**Make Collinear**  
Make vertices collinear.  
Properties: Lock Active Vertex.

**Included Angle**  
If 3 vertices selected, the angle will be based on the second selected vertex. When modifying the angle, the last selected vertex will move and maintain the length from the second vertex.



**Face Normal**

Unit vector direction of Face Normal.  
Properties: Keep Normals to Other Faces, Lock Active Vertex.

**Coplanar Threshold**

In the coplanar judgment under a specific accuracy, 0 is the most accurate.

**Make Coplanar**

Make vertices coplanar.  
Properties: Include Selected Vertices, Keep Normals to Other Faces, Lock Active Vertex

**Faces Area**

Area of selected faces.

**Edges Length**

The total Length of selected edges.

**Keep Normals to Other Faces**

Keep normals of unselected faces if possible.

**Lock Active Vertex**

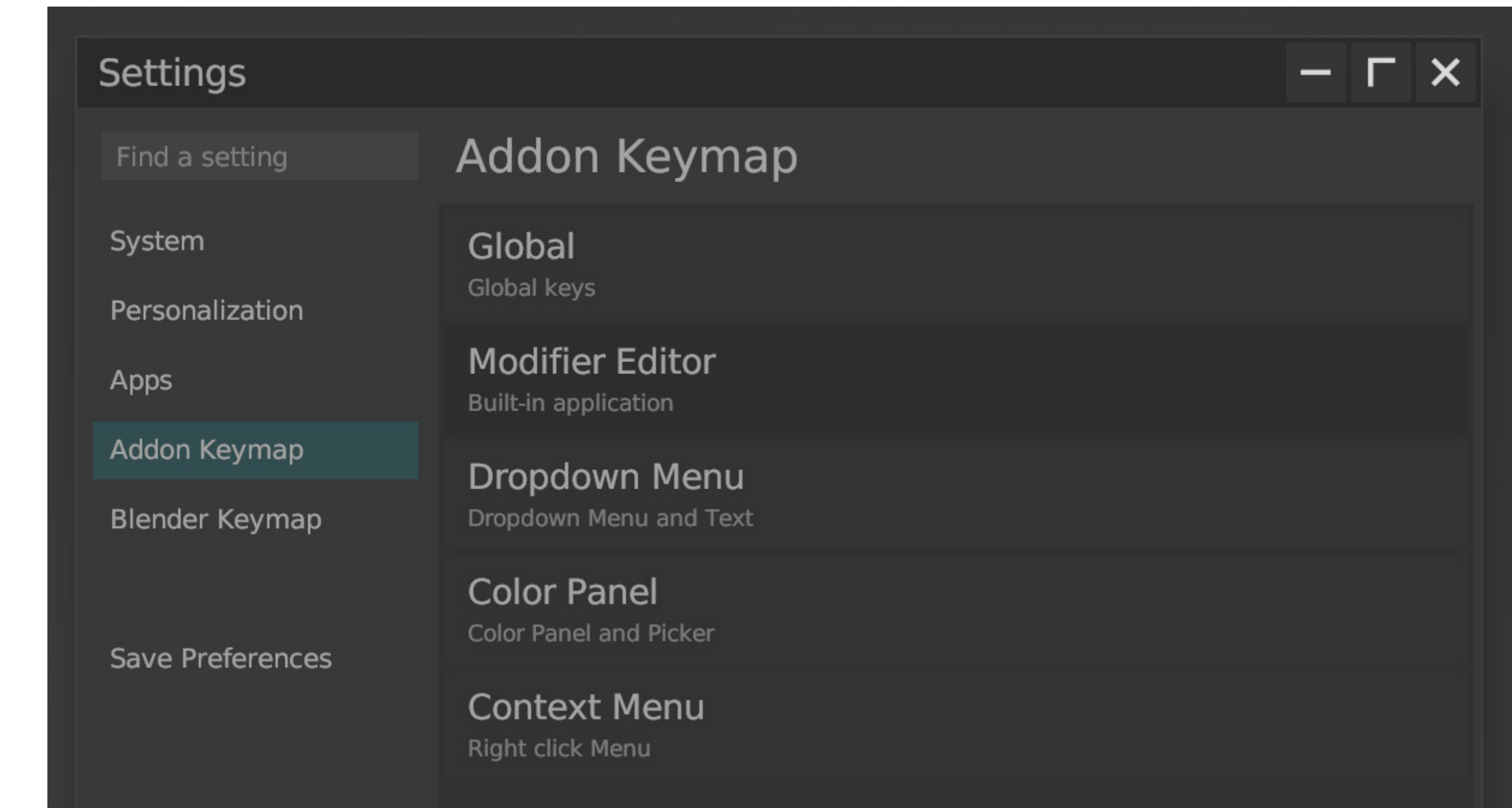
Keep active vertex location.

**Include Selected Vertices**

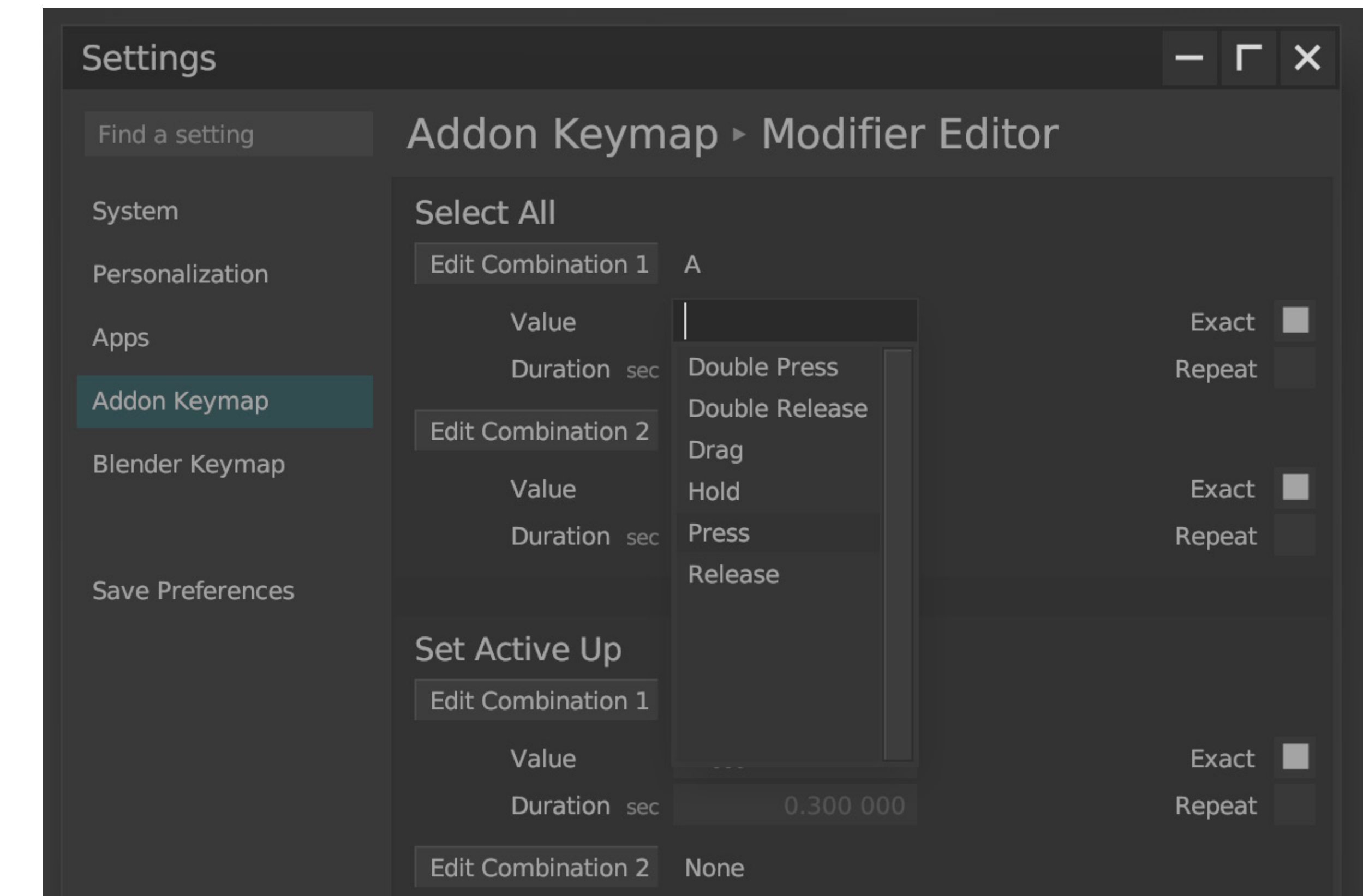
If false, only count selected faces.

# KEYMAP

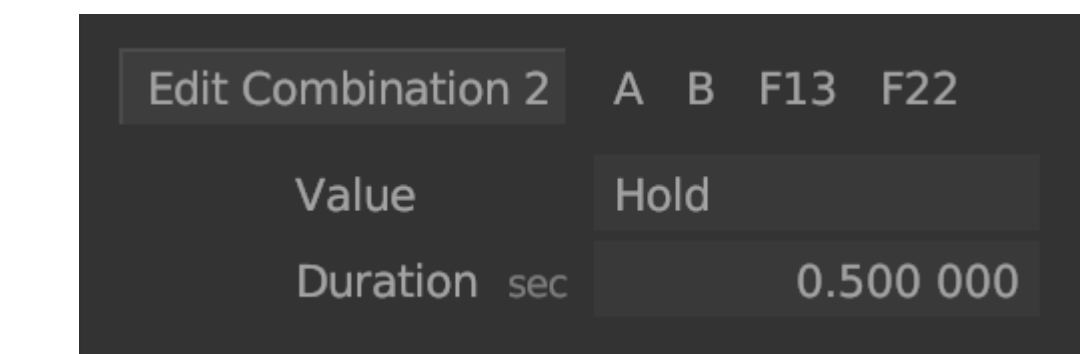
Invoke Settings via N-Panel / Shortcut.



Each Keymap has a Value property to define how to trigger the Operator  
Some Keymaps have an End Value property to define how to end the Operator (eg: Pan).



Addon Keymaps allows assigning up to 4 single keys.



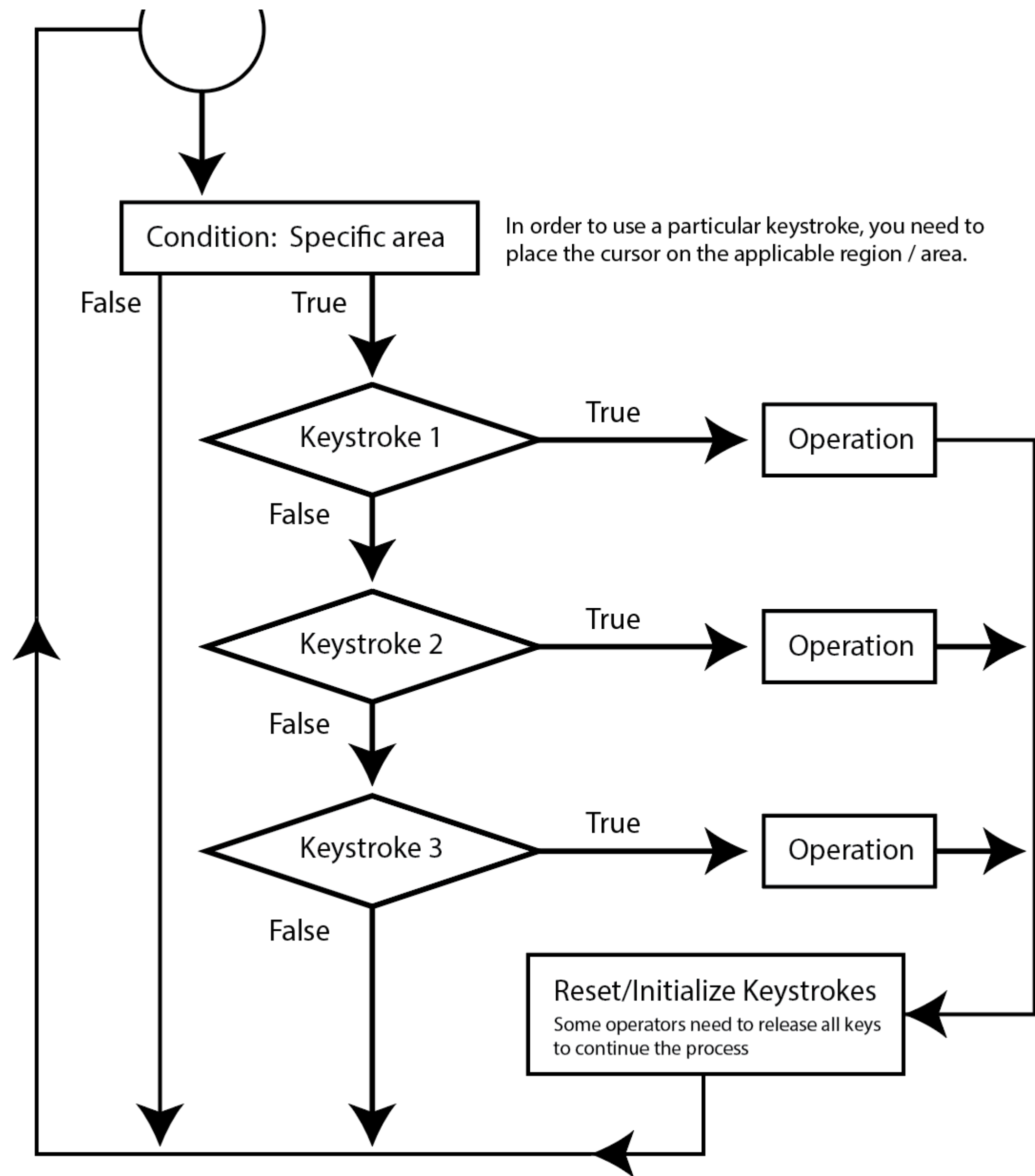
Most membrane keyboards do not respond when pressing multiple keys at the same time (Keyboard Ghosting). For complex keystrokes, we recommend using a mechanical keyboard instead.

# Value definition

- Press** – Triggered when a keystroke is pressed
- Release** – Triggered when a keystroke is released
- Double Press** – Press the keystroke twice within the timer (Duration), each keymap has a separate duration property
- Double Release** – Release the keystroke twice within the timer (Duration)
- Hold** – Hold down the keystroke for a specified time (Duration),
- Drag** – Threshold exceeded while holding keystroke (Default 3 pixels)

# Conflict

In some cases, you need to set some parameters for the key to avoid conflicts (That is, some Operator cannot be executed).



## Example A:

Operator / Keystroke 1

Edit Combination 1	A
Value	Press
	Exact <input type="checkbox"/>

Operator / Keystroke 2

Edit Combination 1	A SPACE
Value	Press
	Exact <input type="checkbox"/>

Operator / Keystroke 3

Edit Combination 1	SPACE A
Value	Press
	Exact <input type="checkbox"/>



To trigger Operator 2, you need to press “Space” before pressing “A”. If “A” is pressed first, Operator 1 will be triggered instead.

If you set the property (Exact) of Operator 1 to True, it means that Operator 1 will only be triggered when a specific button (In this case “A”) is pressed. That means Operator 1 will not be triggered when more than one button is pressed.

Operator 2 and Operator 3 are regarded as the same key combination, since Operator 2 has a higher priority, Operator 3 will never be executed.

**Example B:**

Operator / Keystroke 1

Edit Combination 1

SPACE A

Value

Press

Exact ☒

Operator / Keystroke 2

Edit Combination 1

SPACE A

Value

Press

Exact ☐

Although they have the same keystrokes, Operator 1 Exact is enabled, so you can execute Operator 2 by pressing any key along with “Space” and “A”.

**Priority**

When there is a key conflict, the Operator with higher priority will be executed.

Modifier Editor

Priority	Operator name
(Smaller values have a higher priority)	
1	Select All
2	Set Active Up
3	Set Active Down
4	Set Active Up Extend
5	Set Active Down Extend
6	Modifier Move Up
7	Modifier Move Down
8	Undo
9	Redo
10	Modifier Delete
11	Modifier Apply
12	Pan
13	Sorting
14	Rename
15	Select Extend
16	Select
17	Select Box Extend
18	Select Box

Value Box

1	Batch
2	Quick Edit
3	Button Execute
4	Context Menu
5	Copy
6	Paste
7	Cut (Copy Array)
8	Reset

Modal Quick Edit

Priority	Operator name
(Smaller values have a higher priority)	

- |   |                   |
|---|-------------------|
| 1 | Quick Edit Cancel |
| 2 | Quick Edit End    |
| 3 | Quick Edit Slow   |
| 4 | Quick Edit Fast   |

Color Panel

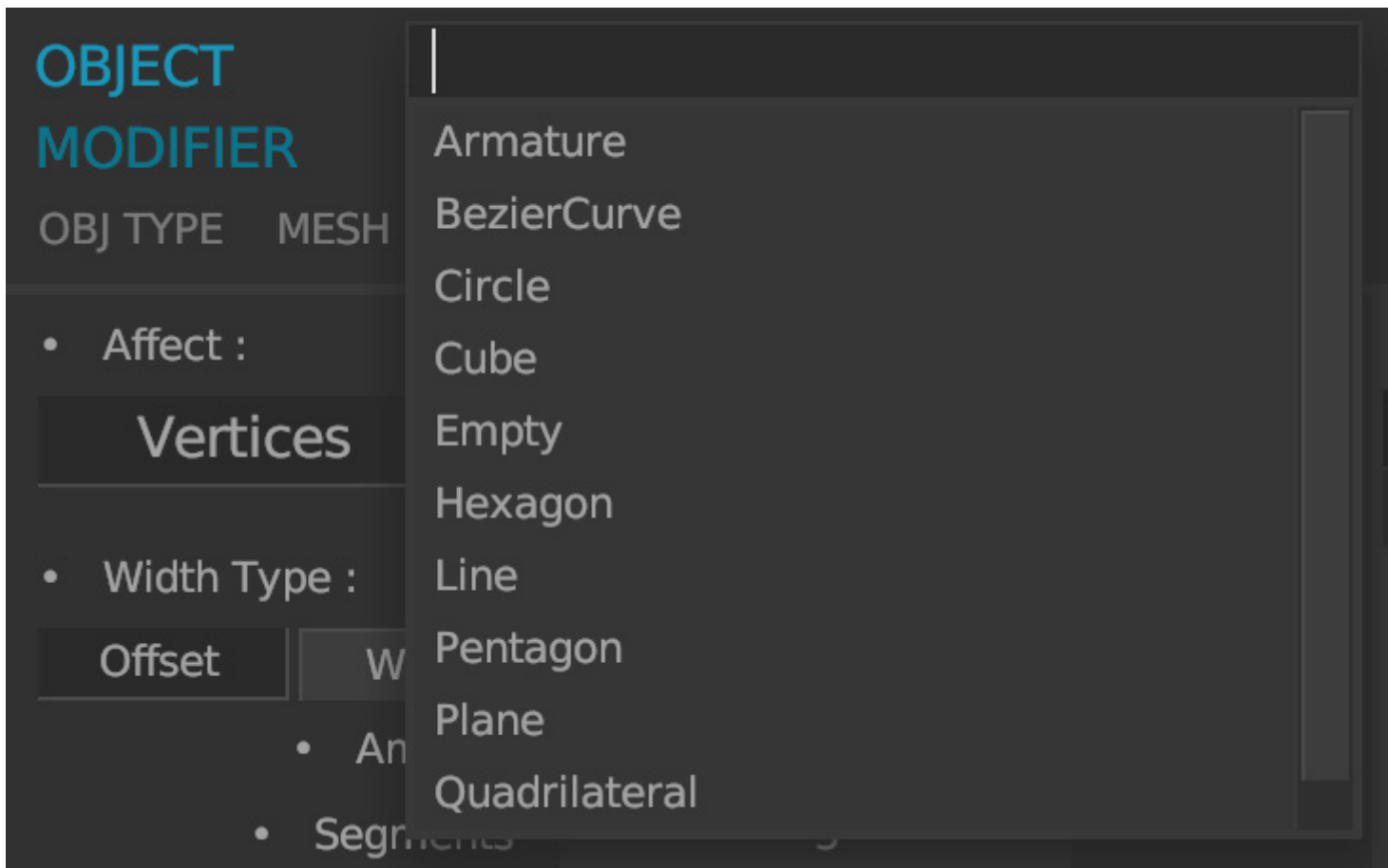
- |   |                          |
|---|--------------------------|
| 1 | Cancel (Drop Down Menu)  |
| 2 | Confirm (Drop Down Menu) |
| 3 | Select (Drop Down Menu)  |
| 4 | Title Bar Move           |
| 5 | Hue Select Continue      |
| 6 | Hue Select               |

Text Box

- |    |                             |
|----|-----------------------------|
| 1  | Cancel (Drop Down Menu)     |
| 2  | Confirm (Drop Down Menu)    |
| 3  | Delete All                  |
| 4  | Delete Word                 |
| 5  | Backspace                   |
| 6  | Cursor Select Left          |
| 7  | Cursor Select Right         |
| 8  | Cursor Select Up            |
| 9  | Cursor Select Down          |
| 10 | Cursor Left                 |
| 11 | Cursor Right                |
| 12 | Cursor Up                   |
| 13 | Cursor Down                 |
| 14 | Select Box (Drop Down Menu) |
| 15 | Pan (Drop Down Menu)        |
| 16 | Copy                        |
| 17 | Paste                       |
| 18 | Cut                         |
| 19 | Select All (Drop Down Menu) |
| 20 | Select (Drop Down Menu)     |
| 21 | Tab                         |
| 22 | Scroll Up                   |
| 23 | Scroll Down                 |
| 24 | Text Input                  |

# BASIC CONTROL

## Text Box



1. Cancel.

Default Keymap 1		Cancel (Drop Down Menu)
Keystroke	ESC	
Value	Press	
End Value	-	
Region	-	

Default Keymap 2		Cancel (Drop Down Menu)
Keystroke	Right Mouse	
Value	Release	
End Value	-	
Region	-	

2. Confirm.

Default Keymap 1		Confirm (Drop Down Menu)
Keystroke	Return	
Value	Press	
End Value	-	
Region	-	

Default Keymap 2		Confirm (Drop Down Menu)
Keystroke	Numpad Enter	
Value	Press	
End Value	-	
Region	-	

3. Delete all character.

Default Keymap 1		Delete All
Keystroke	Delete	
Value	Press	
End Value	-	
Region	-	

4. Delete word from the text cursor.

Default Keymap 1		Delete Word
Keystroke	Left Ctrl Backspace	
Value	Press	
End Value	-	
Region	-	

5. Delete character from the text cursor.

Default Keymap 1		Backspace
Keystroke	Backspace	
Value	Press	
End Value	-	
Region	-	

6. Move the text cursor left and select.

Default Keymap 1		Cursor Select Left
Keystroke	Left Shift	Left Arrow
Value	Press	
End Value	-	
Region	-	

7. Move the text cursor right and select.

Default Keymap 1		Cursor Select Right
Keystroke	Left Shift	Right Arrow
Value	Press	
End Value	-	
Region	-	

8. Move the text cursor up and select.

Default Keymap 1		Cursor Select Up
Keystroke	Left Shift	Up Arrow
Value	Press	
End Value	-	
Region	-	

9. Move the text cursor down and select.

Default Keymap 1		Cursor Select Down
Keystroke	Left Shift	Down Arrow
Value	Press	
End Value	-	
Region	-	

10. Move the text cursor left.

Default Keymap 1		Cursor Left
Keystroke	Left Arrow	
Value	Press	
End Value	-	
Region	-	

11. Move the text cursor right and select.

Default Keymap 1		Cursor Right
Keystroke	Right Arrow	
Value	Press	
End Value	-	
Region	-	

12. Move the text cursor up and select.

Default Keymap 1		Cursor Up
Keystroke	Up Arrow	
Value	Press	
End Value	-	
Region	-	

13. Move the text cursor down and select.

Default Keymap 1		Cursor Down
Keystroke	Down Arrow	
Value	Press	
End Value	-	
Region	-	



14. Use the selection box to select text.

Default Keymap 1

Select Box (Drop Down Menu)

Keystroke	Left Mouse
Value	Drag
End Value	Release
Region	Text Box

15. Move the Text Box canvas.

Default Keymap 1

Pan (Drop Down Menu)

Keystroke	Middle Mouse
Value	Drag
End Value	Release
Region	Text Box

16. Copy text.

Default Keymap 1

Copy

Keystroke	Left Ctrl	C
Value	Press	
End Value	-	
Region	-	

17. Paste text.

Default Keymap 1

Paste

Keystroke	Left Ctrl	V
Value	Press	
End Value	-	
Region	-	

18. Cut text.

Default Keymap 1

Cut

Keystroke	Left Ctrl	X
Value	Press	
End Value	-	
Region	-	

19. Select all characters.

Default Keymap 1

Select All (Drop Down Menu)

Keystroke	Left Mouse
Value	Double Press
End Value	-
Region	Text Box

Default Keymap 1

Select All (Drop Down Menu)

Keystroke	Left Ctrl	A
Value	Press	
End Value	-	
Region	Text Box	

20. Move the text cursor.

Default Keymap 1

Select (Drop Down Menu)

Keystroke	Left Mouse
Value	Press
End Value	-
Region	Text Box

21. Paste text from the filter.

Default Keymap 1

Tab

Keystroke	Tab
Value	Press
End Value	-
Region	-

22. Move the filter canvas down.

Default Keymap 1		Scroll Up
Keystroke	Wheel Up	
Value	Press	
End Value	-	
Region	-	

23. Move the filter canvas up.

Default Keymap 1		Scroll Down
Keystroke	Wheel Down	
Value	Press	
End Value	-	
Region	-	

24. Select results in the filter.

Default Keymap 1		Select (Drop Down Menu)
Keystroke	Left Mouse	
Value	Press	
End Value	-	
Region	Filter	

Value Box

• Amount	0.450 000
• Segments	5

1. Invoke Batch Menu.

Default Keymap 1		Batch
Keystroke	Left Alt Left Mouse	
Value	Press	
End Value	-	
Region	Value Box	

2. Execute Quick Edit modal.

Default Keymap 1		Quick Edit
Keystroke	Left Mouse	
Value	Drag	
End Value	Release	
Region	Value Box	

3. Execute the button function.

Default Keymap 1		Button Execute
Keystroke	Left Mouse	
Value	Release	
End Value	-	
Region	Value Box	

4. Open Context Menu.

Default Keymap 1		Context Menu
Keystroke	Right Mouse	
Value	Release	
End Value	-	
Region	Value Box	

Default Keymap 2		Context Menu
Keystroke	Application	
Value	Press	
End Value	-	
Region	Value Box	

5. Copy text.

Default Keymap 1		Copy
Keystroke	Left Ctrl C	
Value	Press	
End Value	-	
Region	Value Box	

6. Paste text.

Default Keymap 1		Paste
Keystroke	Left Ctrl V	
Value	Press	
End Value	-	
Region	Value Box	

7. Copy Array.

Default Keymap 1		Cut
Keystroke	Left Ctrl X	
Value	Press	
End Value	-	
Region	Value Box	

8. Reset Value.

Default Keymap 1		Reset
Keystroke	Backspace	
Value	Press	
End Value	-	
Region	Value Box	

Modal Quick Edit

1. Cancel Quick Edit modal and restore the value.

Default Keymap 1		Quick Edit Cancel
Keystroke	ESC	
Value	Press	
End Value	-	
Region	-	

Default Keymap 2		Quick Edit Cancel
Keystroke	Right Mouse	
Value	Release	
End Value	-	
Region	-	

2. Decrease the unit in Quick Edit modal while holding the key.

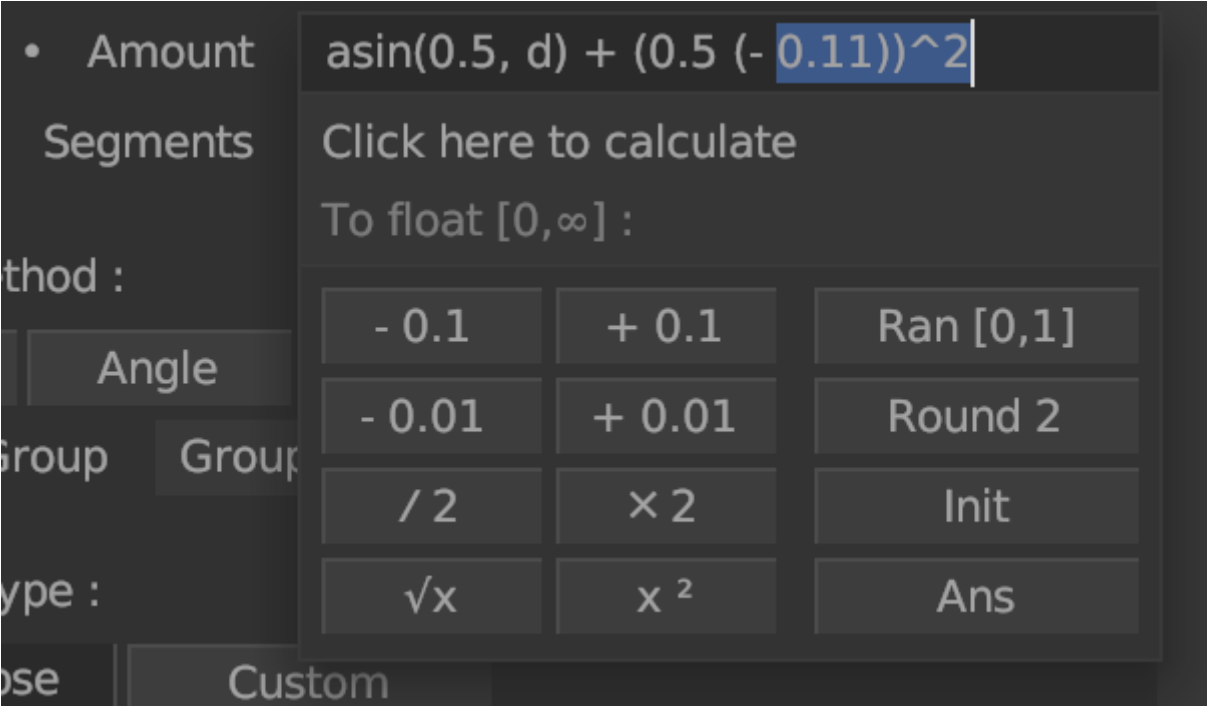
Default Keymap 1		Quick Edit Slow
Keystroke	Left Ctrl	
Value	-	
End Value	-	
Region	-	

3. Increase the unit in Quick Edit modal while holding the key.

Default Keymap 1		Quick Edit Fast
Keystroke	Left Shift	
Value	-	
End Value	-	
Region	-	

# Calculator

When entering value in the Value Box, general expressions and python expressions are supported for calculation. When calculator enable, it allows the user to customize button functionality.



When Output is a complex number, it will be converted to an absolute value.

### Example 1:

Input	1+i
Output	1+i
Value	1.414213562

### Example 2:

Input	floor(pi^2-e)!
Output	5040
Value	5040

To enter a Python expression, you need to insert a semicolon before the expression.

### Example 3:

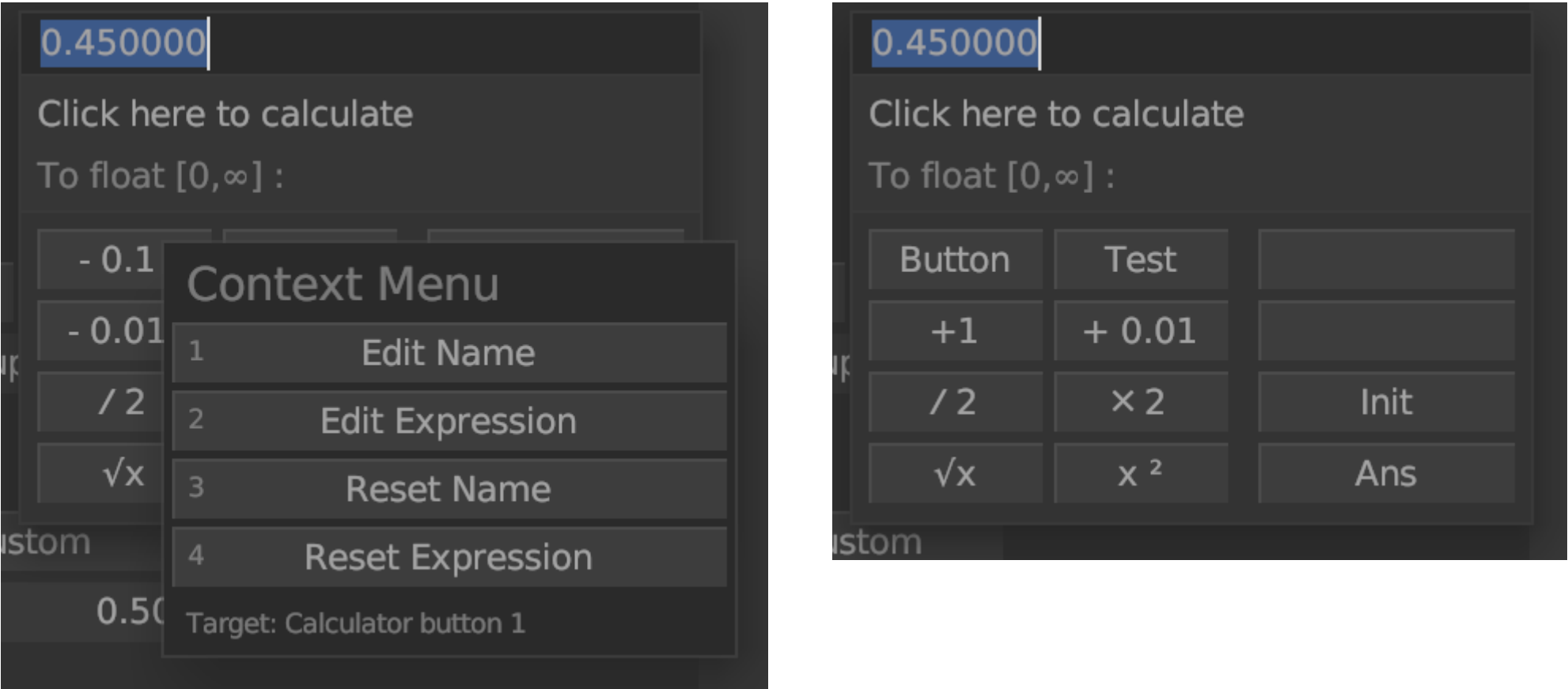
Input	;abs(1j**2)
Output	1
Value	1

To add a driver, you need to insert a hashtag before the expression.

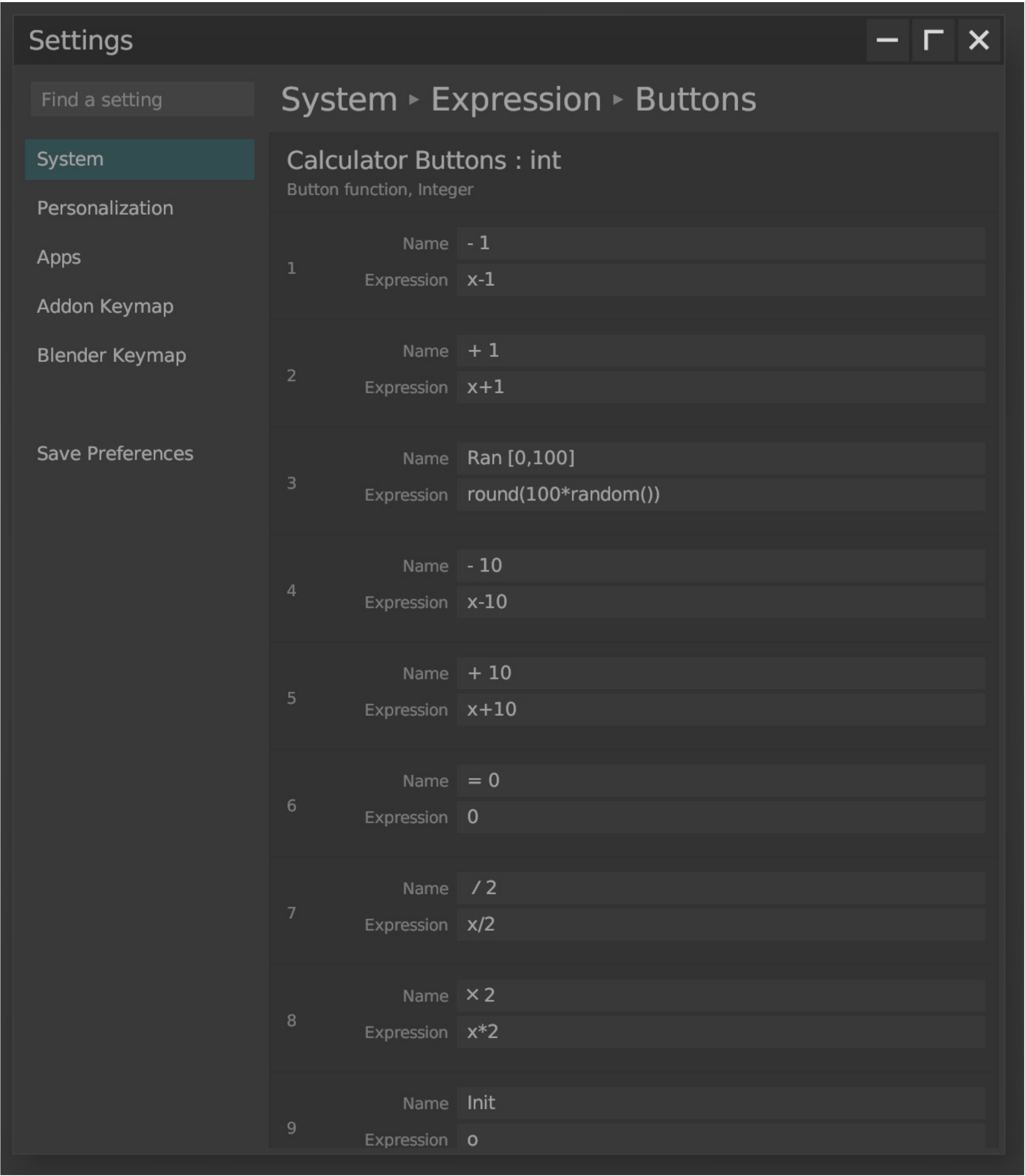
### Example 4:

Input	#1 if frame == 1 else 2
-------	-------------------------

Invoke context menu to edit calculator button names and expressions.



It can also be edited from Settings.





Function List (General Expression)

Constants

- i

Imaginary unit
- j

Same as **i**
- e

Euler’s number, the base of natural logarithms
- $\pi$

The ratio of the circumference of a circle to its diameter
- pi

Same as  $\pi$
- tau

$2\pi$
- ^

Power
- !

Factorial
- %

÷100
- X

Current value
- O

Original value

Constants (Length Unit, Ignore lowercase/uppercase)

- km

kilometer

kilometers

Kilometers to current unit
- m

meter

meters

Meters to current unit
- cm

centimeter

centimeters

Centimeters to current unit

mm  
millimeter  
millimeters

Millimeters to current unit

um  
μ  
mμ  
micrometer  
micrometers

Micrometers to current unit

mi  
mile  
miles

Miles to current unit

ft  
feet  
foot

Feet to current unit

in  
inche  
inches

Inches to current unit

mil  
thou

Thou to current unit

Example

Scene Length unit: Inches  
Scene Unit Scale: 1.5

Input:        2 + 4 mm + 6 ft  
Output:       74.15748  
Attribute value stored in Python:  $74.15748 * 0.0254 / 1.5 = 1.2557333$

## Functions

seed	34	acosh	41
getstate	34	asinh	41
setstate	34	atanh	41
randbytes	34	asech	41
randrange	34	acsch	41
randint	34	acoth	42
getrandbits	34	cosh	42
choice	35	sinh	42
choices	35	tanh	42
shuffle	35	sech	42
sample	35	csch	42
random	35	coth	42
uniform	36	isfinite	42
triangular	36	isinf	43
betavariate	36	isnan	43
expovariate	36	isclose	43
gammavariate	36	Re	43
gauss	36	Im	43
lognormvariate	37	conj	43
normalvariate	37	ceil	44
vonmisesvariate	37	comb	44
paretovariate	37	C	44
weibullvariate	37	copysign	44
phase	38	fabs	44
polar	38	factorial	44
rect	38	floor	44
exp	38	fmod	45
log	38	frexp	45
log10	38	fsum	45
log2	38	gcd	45
ln	38	isqrt	45
sqrt	39	lcm	46
rt	39	ldexp	46
acos	39	modf	46
asin	39	nextafter	46
atan	39	perm	46
asec	39	P	46
acsc	40	prod	47
acot	40	remainder	47
cos	40	trunc	47
sin	40	ulp	47
tan	40	cbirt	48
sec	40	exp2	48
csc	41	expm1	48
cot	41	log1p	48

atan2	48
dist	48
hypot	48
degrees	49
radians	49
erf	49
erfc	49
gamma	49
lgamma	49
round	49
pow	50
min	50
max	50



## seed()

Initialize the random number generator.

## getstate()

Return an object capturing the current internal state of the generator. This object can be passed to setstate() to restore the state.

## setstate()

var 1: state

state should have been obtained from a previous call to getstate(), and setstate() restores the internal state of the generator to what it was at the time getstate() was called.

## randbytes()

var 1 (n): positive integer

Generate n random bytes.

Input:       int(randbytes(5).hex(), 16)  
Output:       621644248778

## randrange()

var 1 (stop): integer

Return a randomly selected element from range(0, stop)

Input:       randrange(6)  
Output:       5

## randrange()

var 1 (start): integer

var 2 (stop): integer

var 3 (step = 1, Optional): integer

Return a randomly selected element from range(start, stop, step)

Input:       randrange(3, 6, 2)  
Output:       3

## randint()

var 1 (a): integer

var 2 (b): integer

Return a random integer N such that  $a \leq N \leq b$ . Alias for randrange(a, b+1)

Input:       randint(3, 5)  
Output:       5

## getrandbits()

var 1 (k): nonnegative integer

Returns a non-negative Python integer with k random bits. This method is supplied with the MersenneTwister generator and some other generators may also provide it as an optional part of the API. When available, getrandbits() enables randrange() to handle arbitrarily large ranges.

Input:       getrandbits(5)  
Output:       30

**choice()**                      var 1 (seq): real number

Return a random element from the non-empty sequence seq.

```
Input:      choice((2, 3, 4))
Output:     2
```

**choices()**

var 1 (population): seq                      var 2 (k = 1, Optional): positive integer  
var 3 (weights, Optional): seq              var 4 (cum\_weights, Optional): seq

Return a k sized list of elements chosen from the population with replacement. If a weights sequence is specified, selections are made according to the relative weights. Alternatively, if a cum\_weights sequence is given, the selections are made according to the cumulative weights. For example, the relative weights [10, 5, 30, 5] are equivalent to the cumulative weights [10, 15, 45, 50]. Internally, the relative weights are converted to cumulative weights before making selections.

```
Input:      sum(choices((1, 2, 3, 4), 1000)) / 1000
Output:     2.506

Input:      sum(choices((1, 2, 3, 4), 1000, None, (1, 1, 1, 1))) / 1000
Output:     1
```

**shuffle()**

var 1 (x): seq

Shuffle the sequence x in place.

```
Input:      shuffle((1, 2, 3, 4))[0]
Output:     2
```

**sample()**

var 1 (population): seq                      var 2 (k): positive integer  
var 3 (counts, Optional): seq

Return a k length list of unique elements chosen from the population sequence. Used for random sampling without replacement.

Returns a new list containing elements from the population while leaving the original population unchanged. The resulting list is in selection order so that all sub-slices will also be valid random samples. This allows raffle winners (the sample) to be partitioned into grand prize and second place winners (the subslices).

Members of the population need not be hashable or unique. If the population contains repeats, then each occurrence is a possible selection in the sample.

Repeated elements can be specified one at a time or with the optional keyword-only counts parameter. For example, sample(['red', 'blue'], counts=[4, 2], k=5) is equivalent to sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)

```
Input:      sum(sample((1, 1, 2, 2, 3, 4), 4))
Output:     9

Input:      sum(sample((1, 2, 3, 4), 4, (2, 2, 1, 1)))
Output:     8
```

**random()**

Return the next random floating point number in the range [0.0, 1.0)

```
Input:      random()
Output:     0.8070585759119
```

**uniform()**                      var 1 (a): real number                      var 2 (b): real number

Return a random floating point number N such that  $a \leq N \leq b$  for  $a \leq b$  and  $b \leq N \leq a$  for  $b < a$ .  
The end-point value b may or may not be included in the range depending on floating-point rounding in the equation  
 $a + (b-a) * \text{random}()$

Input:            uniform(-2.3, 7.2)  
Output:           -0.7599512525748

**triangular()**                      var 1 (low): real number                      var 2 (high): real number  
var 3 (mode): real number

Return a random floating point number N such that  $\text{low} \leq N \leq \text{high}$  and with the specified mode between those bounds.  
The low and high bounds default to zero and one. The mode argument defaults to the midpoint between the bounds, giving a symmetric distribution

Input:            triangular(-2.1, 5.6, 3.1)  
Output:           4.650596214649

**betavariate()**                      var 1 (alpha): real number                      var 2 (beta): real number

Beta distribution. Conditions on the parameters are  $\alpha > 0$  and  $\beta > 0$ . Returned values range between 0 and 1

Input:            betavariate(2.1, 2.2)  
Output:           0.5409938021106

**expovariate()**                      var 1 (lambd): real number

Exponential distribution. lambd is 1.0 divided by the desired mean. It should be nonzero. (The parameter would be called “lambda”, but that is a reserved word in Python.) Returned values range from 0 to positive infinity if lambd is positive, and from negative infinity to 0 if lambd is negative

Input:            expovariate(2)  
Output:           0.029974233415711

**gammavariate()**                      var 1 (alpha): real number                      var 2 (beta): real number

Gamma distribution. (Not the gamma function!) Conditions on the parameters are  $\alpha > 0$  and  $\beta > 0$

Input:            gammavariate(2.1, 2.2)  
Output:           9.419232015484

**gauss()**                      var 1 (mu): real number                      var 2 (sigma): real number

Normal distribution, also called the Gaussian distribution. mu is the mean, and sigma is the standard deviation. This is slightly faster than the normalvariate() function defined below.

Multithreading note: When two threads call this function simultaneously, it is possible that they will receive the same return value. This can be avoided in three ways. 1) Have each thread use a different instance of the random number generator. 2) Put locks around all calls. 3) Use the slower, but thread-safe normalvariate() function instead

Input:            gauss(0, 1)  
Output:           2.3667486814625

## lognormvariate()

var 1 (mu): real number

var 2 (sigma): real number

Log normal distribution. If you take the natural logarithm of this distribution, you'll get a normal distribution with mean mu and standard deviation sigma. mu can have any value, and sigma must be greater than zero

Input: lognormvariate(0, 1)

Output: 1.615633099343

## normalvariate()

var 1 (mu): real number

var 2 (sigma): real number

Normal distribution. mu is the mean, and sigma is the standard deviation

Input: normalvariate(0, 1)

Output: 1.87181748392

## vonmisesvariate()

var 1 (mu): real number

var 2 (kappa): real number

mu is the mean angle, expressed in radians between 0 and  $2\pi$ , and kappa is the concentration parameter, which must be greater than or equal to zero. If kappa is equal to zero, this distribution reduces to a uniform random angle over the range 0 to  $2\pi$

Input: vonmisesvariate(0, 1)

Output: 0.1528172805248

## paretovariate()

var 1 (alpha): real number

Pareto distribution. alpha is the shape parameter

Input: paretovariate(2.1)

Output: 1.9917223270650

## weibullvariate()

var 1 (alpha): real number

var 2 (beta): real number

Weibull distribution. alpha is the scale parameter and beta is the shape parameter

Input: weibullvariate(2.1, 2.2)

Output: 4.055761225935



**phase()**                      var 1 (x): complex / real number

Return the phase of x

Input:            phase(1 + i)

Output:          0.785398163397

**polar()**                      var 1 (x): complex / real number

Return the representation of x in polar coordinates. Returns a pair (r, phi) where r is the modulus of x and phi is the phase of x. polar(x) is equivalent to (abs(x), phase(x))

Input:            polar(1 + i)[1]

Output:          0.785398163397

**rect()**                      var 1 (r): real number                      var 2 (θ): real number

Return the complex number x with polar coordinates r and θ. Equivalent to  $r(\cos \theta + i \sin \theta)$

Input:            rect(2, 5)

Output:          0.5673243709265 - 1.917848549326 i

**exp()**                      var 1 (x): complex / real number

Return e raised to the power x, where e is the base of natural logarithms

Input:            exp(i\*pi)

Output:          -1

**log()**                      var 1: complex / real number                      var 2 (base, Optional): complex / real number

Logarithm in base 10. There is one branch cut, from 0 along the negative real axis to  $-\infty$ , continuous from above.

Input:            log(10)

Output:          1

Input:            log(8, 2)

Output:          3

**log10()**                      var 1: complex / real number                      var 2 (base, Optional): complex / real number

Same as log()

**log2()**                      var 1: complex / real number                      var 2 (base, Optional): complex / real number

Logarithm in base 2

Input:            log2(2)

Output:          1

**ln()**                      var 1: complex / real number                      var 2 (base, Optional): complex / real number

Natural logarithm

Input:            ln(e)

Output:          1

**sqrt()**  
Square root

var 1: complex / real number

Input: sqrt(4)

Output: 2

Input: sqrt(8, 3)

Output: 2

var 2 (Optional): complex / real number

**rt()**  
same as sqrt()

**acos()**

var 1: complex / real number

Input: acos(rt(2)/2)

Output: 0.7853981633974

Input: acos(rt(2)/2, d)

Output: 45

var 2 (angle unit, Optional): d

Return the arc cosine of x. There are two branch cuts: One extends right from 1 along the real axis to  $\infty$ , continuous from below. The other extends left from -1 along the real axis to  $-\infty$ , continuous from above

**asin()**

var 1: complex / real number

Input: asin(rt(2)/2)

Output: 0.7853981633974

Input: asin(rt(2)/2, d)

Output: 45

var 2 (angle unit, Optional): d

Return the arc sine of x. This has the same branch cuts as acos()

**atan()**

var 1: complex / real number

Input: atan(1)

Output: 0.7853981633974

Input: atan(1, d)

Output: 45

var 2 (angle unit, Optional): d

Return the arc tangent of x. There are two branch cuts: One extends from 1i along the imaginary axis to  $\infty i$ , continuous from the right. The other extends from -1i along the imaginary axis to  $-\infty i$ , continuous from the left

**asec()**

var 1: complex / real number

Input: asec(2)

Output: 1.0471975511966

Input: asec(2, d)

Output: 60

var 2 (angle unit, Optional): d

Return the arc secant of x

acsc()

Return the arc cosecant of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: acsc(2)

Output: 0.5235987755983

Input: acsc(2, d)

Output: 30

acot()

Return the arc cotangent of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: acot(2)

Output: 0.4636476090008

Input: acot(2, d)

Output: 26.565051177078

cos()

Return the cosine of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: cos(pi)

Output: -1

Input: cos(180, d)

Output: -1

sin()

Return the sine of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: sin(pi)

Output: 0

Input: sin(180, d)

Output: 0

tan()

Return the tangent of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: tan(pi/4)

Output: 1

Input: tan(45, d)

Output: 1

sec()

Return the secant of x

var 1: complex / real number

var 2 (angle unit, Optional): d

Input: sec(pi/4)

Output: 1.414213562373

Input: sec(45, d)

Output: 1.414213562373

40

**csc()**                      var 1: complex / real number                      var 2 (angle unit, Optional): d

Return the cosecant of x

Input:            csc(pi/3)  
Output:           1.1547005383793

Input:            csc(60, d)  
Output:           1.1547005383793

**cot()**                      var 1: complex / real number                      var 2 (angle unit, Optional): d

Return the cotangent of x

Input:            cot(pi/3)  
Output:           0.5773502691896

Input:            cot(60, d)  
Output:           0.5773502691896

**acosh()**                      var 1: complex / real number

Return the inverse hyperbolic cosine of x. There is one branch cut, extending left from 1 along the real axis to  $-\infty$ , continuous from above

Input:            acosh(2)  
Output:           1.3169578969248

**asinh()**                      var 1: complex / real number

Return the inverse hyperbolic sine of x. There are two branch cuts: One extends from 1i along the imaginary axis to  $\infty$  i, continuous from the right. The other extends from -1i along the imaginary axis to  $-\infty$  i, continuous from the left

Input:            asinh(2)  
Output:           1.4436354751788

**atanh()**                      var 1: complex / real number

Return the inverse hyperbolic tangent of x. There are two branch cuts: One extends from 1 along the real axis to  $\infty$ , continuous from below. The other extends from -1 along the real axis to  $-\infty$ , continuous from above

Input:            atanh(0.5)  
Output:           0.5493061443341

**asech()**                      var 1: complex / real number

Return the inverse hyperbolic secant of x

Input:            asech(0.5)  
Output:           1.3169578969248

**acsch()**                      var 1: complex / real number

Return the inverse hyperbolic cosecant of x

Input:            acsch(0.5)  
Output:           1.4436354751788



**acoth()**                      var 1: complex / real number

Return the inverse hyperbolic cotangent of x

Input:            acoth(2)

Output:          0.54930614433405

**cosh()**                      var 1: complex / real number

Return the hyperbolic cosine of x

Input:            cosh(pi)

Output:          11.5919532755215

**sinh()**                      var 1: complex / real number

Return the hyperbolic sine of x

Input:            sinh(pi)

Output:          11.5487393572577

**tanh()**                      var 1: complex / real number

Return the hyperbolic tangent of x

Input:            tanh(pi)

Output:          0.99627207622

**sech()**                      var 1: complex / real number

Return the hyperbolic secant of x

Input:            sech(pi)

Output:          0.08626673833405

**csch()**                      var 1: complex / real number

Return the hyperbolic cosecant of x

Input:            csch(pi)

Output:          0.08658953753005

**coth()**                      var 1: complex / real number

Return the hyperbolic cotangent of x

Input:            coth(pi)

Output:          1.0037418731973

**isfinite()**                  var 1 (x): complex / real number

Return True if both the real and imaginary parts of x are finite, and False otherwise.

Input:            int(isfinite(inf))

Output:          0

## isinf()

var 1 (x): complex / real number

Return True if either the real or the imaginary part of x is an infinity, and False otherwise.

Input:      int(isinf(1j))  
Output:     1

## isnan()

var 1 (x): complex / real number

Return True if either the real or the imaginary part of x is a NaN, and False otherwise.

Input:      int(isnan(1j))  
Output:     1

## isclose()

var 1 (a): complex / real number                      var 2 (b): complex / real number  
var 3 (rel\_tol = 10\*\*-9, Optional): real number  
var 4 (abs\_tol = 0, Optional): real number

Return True if the values a and b are close to each other and False otherwise.

Whether or not two values are considered close is determined according to given absolute and relative tolerances.

rel\_tol is the relative tolerance – it is the maximum allowed difference between a and b, relative to the larger absolute value of a or b. For example, to set a tolerance of 5%, pass rel\_tol=0.05. The default tolerance is 1e-09, which assures that the two values are the same within about 9 decimal digits. rel\_tol must be greater than zero.

abs\_tol is the minimum absolute tolerance – useful for comparisons near zero. abs\_tol must be at least zero.

If no errors occur, the result will be: abs(a-b) <= max(rel\_tol \* max(abs(a), abs(b)), abs\_tol).

The IEEE 754 special values of NaN, inf, and -inf will be handled according to IEEE rules. Specifically, NaN is not considered close to any other value, including NaN. inf and -inf are only considered close to themselves.

Input:      int(isclose(1, 1 + 10^-10))  
Output:     1

## Re()

var 1: complex / real number

Return the real part of a complex number. Same as z.real

Input:      Re(1 + 1j)  
Output:     1

## Im()

var 1: complex / real number

Return the imaginary part of a complex number. Same as z.imag

Input:      Im(1 + 1j)  
Output:     1j

## conj()

var 1: complex / real number

Return the complex conjugate

Input:      conj(1 + 1j)  
Output:     1 - 1j

**ceil()**                      var 1: real

Return the ceiling of x, the smallest integer greater than or equal to x

Input:        ceil(8.1)

Output:       9

**comb()**                      var 1: nonnegative integer                      var 2: nonnegative integer

Combination

Input:        comb(5, 2)

Output:       10

**C()**

same as comb()

**copysign()**                      var 1 (x): real number                      var 2 (y): real number

Return a float with the magnitude (absolute value) of x but the sign of y.

Input:        copysign(2, -1)

Output:       -2

**fabs()**                      var 1: real number

Absolute value, Python version of the math module

Input:        fabs(-1)

Output:       1

**abs()**                      var 1: complex / real number

Absolute value

Input:        abs(1 + i)

Output:       1.414213562373

**factorial()**                      var 1 (n): nonnegative integer

Return n factorial as an integer.

Input:        factorial(5)

Output:       120

**floor()**                      var 1: real

Return the floor of x, the largest integer less than or equal to x

Input:        floor(9.9)

Output:       9

## fmod()

var 1: real number

var 2: real number

Return fmod(x, y), as defined by the platform C library. Note that the Python expression `x % y` may not return the same result. The intent of the C standard is that fmod(x, y) be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer `n` such that the result has the same sign as `x` and magnitude less than `abs(y)`

Input:      fmod(5, 3)  
Output:     2

## frexp()

var 1 (x): real number

Return the mantissa and exponent of `x` as the pair (m, e). `m` is a float and `e` is an integer such that `x == m * 2**e` exactly. If `x` is zero, returns (0.0, 0), otherwise `0.5 <= abs(m) < 1`. This is used to “pick apart” the internal representation of a float in a portable way

Input:      frexp(0.0625)[0]  
Output:     0.5

## fsum()

var 1 (seq): real number

Return an accurate floating point sum of values in the iterable. Avoids loss of precision by tracking multiple intermediate partial sums

Input:      fsum((1, 0.1, 0.01))  
Output:     1.11

## sum()

var 1,2,...,n: complex / real number

Sum of value.

Input:      sum(1, 0.1, 0.01)  
Output:     1.11

## sum()

var 1 (seq): complex / real number

Sum of value.

Input:      sum((1, 0.1, 0.01))  
Output:     1.11

## gcd()

var 1,2,...,n: nonnegative integer

Greatest common divisor

Input:      gcd(6, 12, 18)  
Output:     6

## isqrt()

var 1: nonnegative integer

Return the integer square root of the nonnegative integer `n`. This is the floor of the exact square root of `n`, or equivalently the greatest integer `a` such that  $a^2 \leq n$

Input:      isqrt(65)  
Output:     8



**lcm()**                      var 1,2,...,n: nonnegative integer  
Least common multiple

Input:        lcm(6, 12, 18)  
Output:       36

**ldexp()**                      var 1 (x): real number                      var 2 (y): integer  
Return  $x * (2^{**}y)$ . This is essentially the inverse of function frexp()

Input:        ldexp(0.5, -3)  
Output:       0.0625

**modf()**                      var 1 (x): real number  
Return the fractional and integer parts of x. Both results carry the sign of x and are floats.

Input:        modf(5.2)[0]  
Output:       0.2

**nextafter()**                      var 1 (x): real number                      var 2 (y): real number  
Return the next floating-point value after x towards y  
If x is equal to y, return y

Input:        nextafter(5, 1)  
Output:       4.999999999999

**perm()**                      var 1: nonnegative integer                      var 2: nonnegative integer  
Permutation

Input:        perm(5, 2)  
Output:       20

**P()**  
same as perm()

**prod()**                      var 1 (seq): real number                      var 2 (start = 1, Optional): real number  
Calculate the product of all the elements in the input iterable. The default start value for the product is 1.

When the iterable is empty, return the start value. This function is intended specifically for use with numeric values and may reject non-numeric types

Input:        prod((4,5,6))  
Output:       120

**prod()**                      var 1,2,...,n: real number

Calculate the product of the inputs.

Input:            prod(4,5,6)  
Output:          120

**remainder()**              var 1 (x): real number                      var 2 (y): real number

Return the IEEE 754-style remainder of x with respect to y. For finite x and finite nonzero y, this is the difference  $x - n*y$ , where n is the closest integer to the exact value of the quotient  $x / y$ . If  $x / y$  is exactly halfway between two consecutive integers, the nearest even integer is used for n. The remainder  $r = \text{remainder}(x, y)$  thus always satisfies  $\text{abs}(r) \leq 0.5 * \text{abs}(y)$

Special cases follow IEEE 754: in particular,  $\text{remainder}(x, \text{math.inf})$  is x for any finite x, and  $\text{remainder}(x, 0)$  and  $\text{remainder}(\text{math.inf}, x)$  raise ValueError for any non-NaN x. If the result of the remainder operation is zero, that zero will have the same sign as x

Input:            remainder(5, -3)  
Output:          -1

**trunc()**                      var 1: real number

Return x with the fractional part removed, leaving the integer part. This rounds toward 0:  $\text{trunc}()$  is equivalent to  $\text{floor}()$  for positive x, and equivalent to  $\text{ceil}()$  for negative x. If x is not a float, delegates to  $x.\_\text{trunc}\_\_$ , which should return an Integral value

Input:            trunc(pi)  
Output:          3

**ulp()**                      var 1: real number

Return the value of the least significant bit of the float x:

If x is a NaN (not a number), return x.

If x is negative, return  $\text{ulp}(-x)$ .

If x is a positive infinity, return x.

If x is equal to zero, return the smallest positive denormalized representable float (smaller than the minimum positive normalized float,  $\text{sys.float\_info.min}$ ).

If x is equal to the largest positive representable float, return the value of the least significant bit of x, such that the first float smaller than x is  $x - \text{ulp}(x)$ .

Otherwise (x is a positive finite number), return the value of the least significant bit of x, such that the first float bigger than x is  $x + \text{ulp}(x)$ .

ULP stands for “Unit in the Last Place”

Input:            ulp( $10^{12}$ )  
Output:          0.0001220703

**cbrt()**                      var 1 (x): real number

Return the cube root of x

Input:            cbrt(8)  
Output:           2

**exp2()**                      var 1 (x): real number

Return 2 raised to the power x

Input:            exp2(5)  
Output:           32

**expm1()**                      var 1: real number

Return e raised to the power x, minus 1. Here e is the base of natural logarithms. For small floats x, the subtraction in  $\exp(x) - 1$  can result in a significant loss of precision; the `expm1()` function provides a way to compute this quantity to full precision

Input:            expm1(3)  
Output:           19.085536923188

**log1p()**                      var 1: real number

Return the natural logarithm of 1+x (base e). The result is calculated in a way which is accurate for x near zero

Input:            log1p(e-1)  
Output:           1

**atan2()**                      var 1 (y): real number                      var 2 (x): real number

Return  $\text{atan}(y / x)$ , in radians. The result is between  $-\pi$  and  $\pi$ . The vector in the plane from the origin to point (x, y) makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both  $\pi/4$ , but `atan2(-1, -1)` is  $-3\pi/4$

Input:            atan2(-1, -1)  
Output:           -2.356194490192

**dist()**                      var 1: coordinate                      var 2: coordinate

Return the Euclidean distance between two points p and q, each given as a sequence (or iterable) of coordinates. The two points must have the same dimension

Input:            dist((5,0,0), (8,0,0))  
Output:           3

**hypot()**                      var 1,2,...,n: real number

Return the Euclidean norm,  $\sqrt{\text{sum}(x^2 \text{ for } x \text{ in coordinates})}$ . This is the length of the vector from the origin to the point given by the coordinates.

For a two dimensional point (x, y), this is equivalent to computing the hypotenuse of a right triangle using the Pythagorean theorem,  $\sqrt{x^2 + y^2}$ .

Input:            hypot(5, 6)  
Output:           7.810249675907

**degrees()**                      var 1: real

Convert angle x from radians to degrees

Input:            degrees(pi)  
Output:           180

**radians()**                      var 1: real

Convert angle x from degrees to radians

Input:            radians(180)  
Output:           3.1415926535898

**erf()**                              var 1: real number

Return the error function at x

Input:            erf(2)  
Output:           0.995322265019

**erfc()**                              var 1: real number

Return the complementary error function at x. The complementary error function is defined as 1.0 - erf(x). It is used for large values of x where a subtraction from one would cause a loss of significance

Input:            erfc(2)  
Output:           0.004677734981047

**gamma()**                              var 1: real number

Return the Gamma function at x

Input:            gamma(2.2)  
Output:           1.1018024908797

**lgamma()**                              var 1: real number

Return the natural logarithm of the absolute value of the Gamma function at x

Input:            lgamma(2.2)  
Output:           0.09694746679064

**round()**                              var 1: real number                      var 2 (places, Optional): integer

Returns a number that is a rounded version of the specified number, with the specified number of decimals

Input:            round(5.5)  
Output:           6

Input:            round(14.5, -1)  
Output:           10



pow()

Power

var 1 (base): complex / real number

var 2 (power): complex / real number

Input:pow(2, 3)

Output:8

min()

Return the smallest item

var 1,2,...,n: real number

Input:min(1, 2, 3)

Output:1

max()

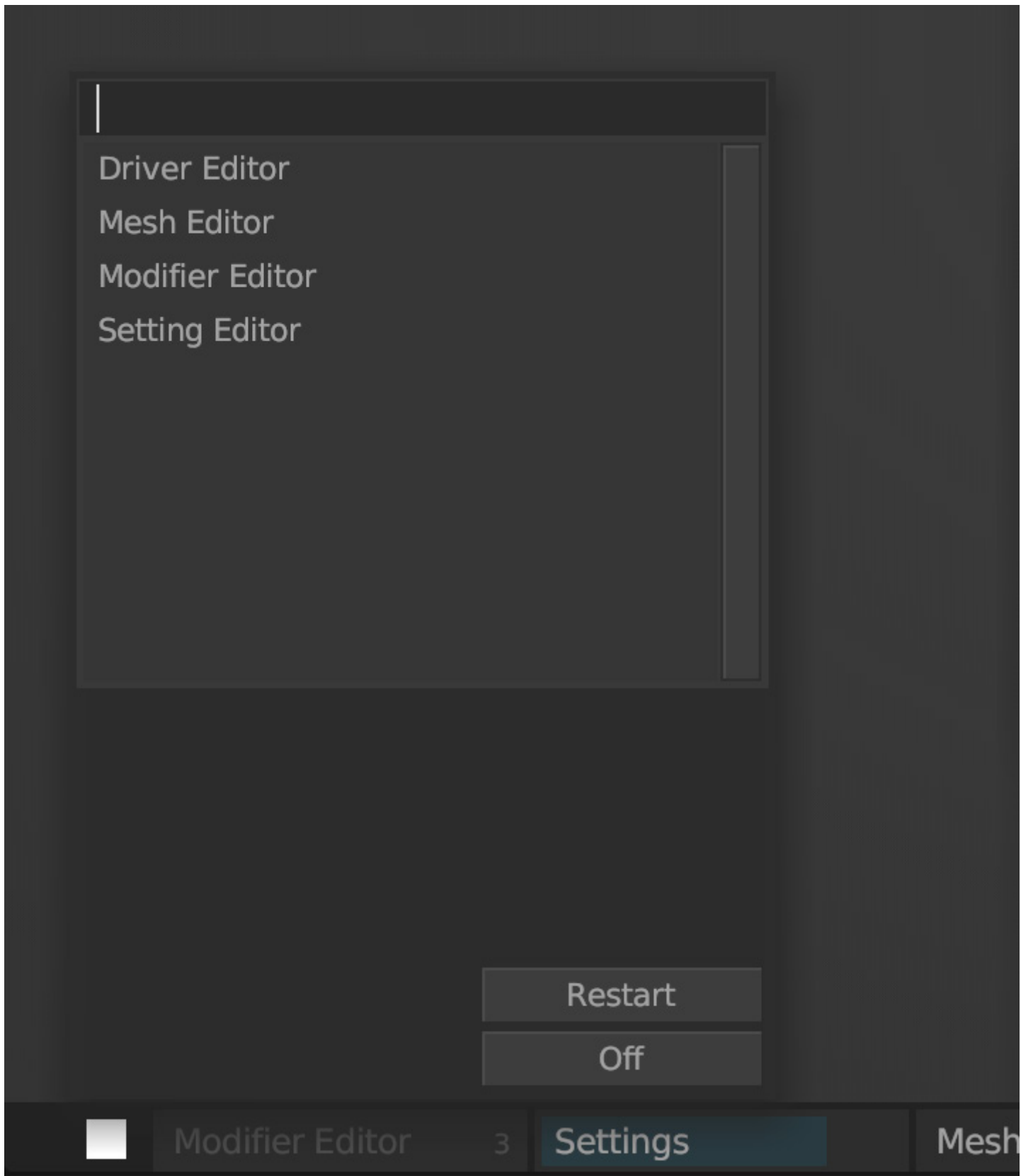
Return the largest item

var 1,2,...,n: real number

Input:max(1, 2, 3)

Output:3

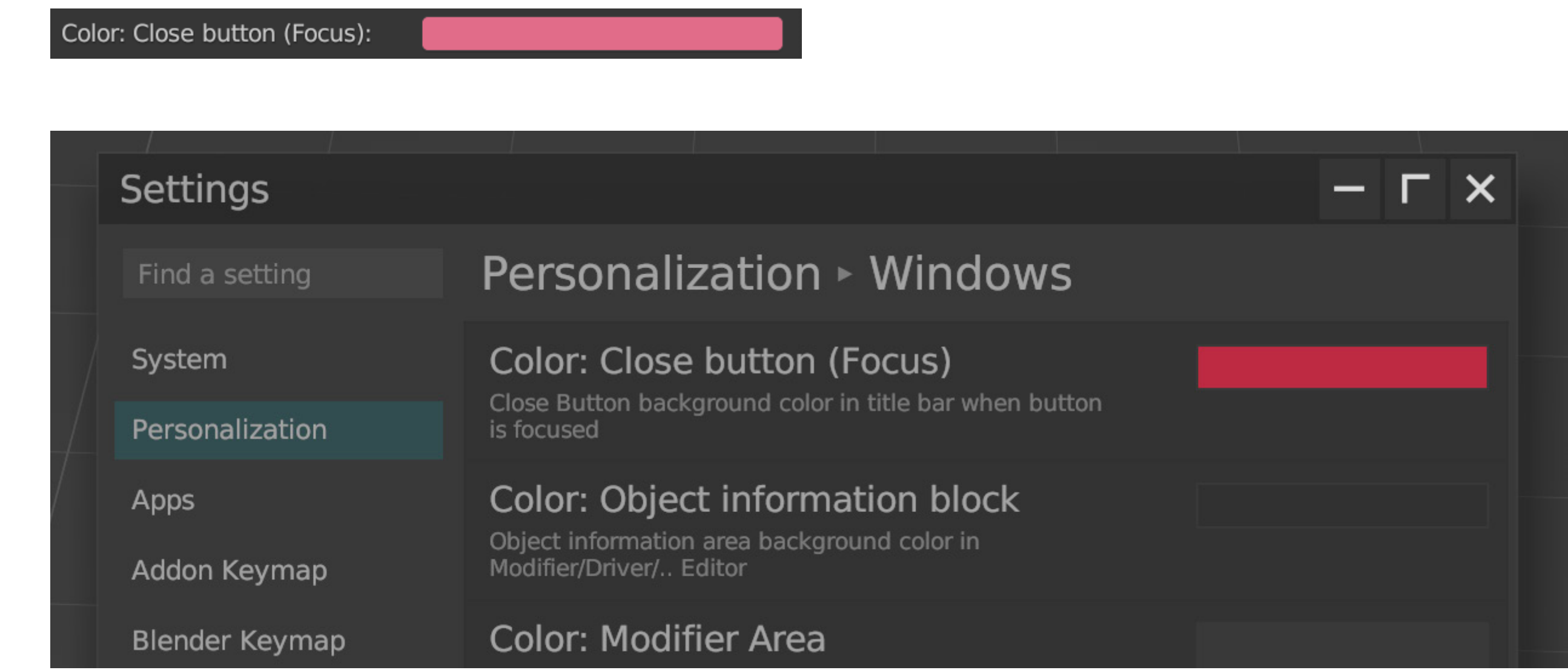
# TASK BAR



When no tasks are running, the subsystem will automatically close. You can disable this setting in Settings.

# THEME AND COLOR

The color of the interface in N-panel will not be displayed correctly, please use Settings Editor instead.



# ABOUT

## Bug Report

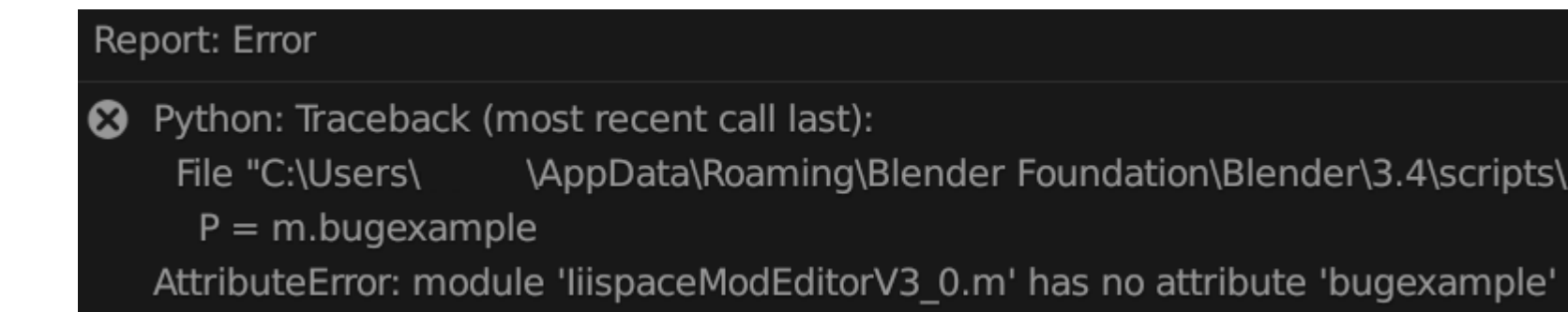
E-mail:  
oorcer@gmail.com

### Report Example

1. Blender version: 3.4
2. Addon version: 22.11
3. OS: Windows 11
4. CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
5. RAM: 32.0 GB
6. GPU: GeForce RTX 3060
7. Description: Error when click the “Call Settings” button in N-panel
8. Traceback:

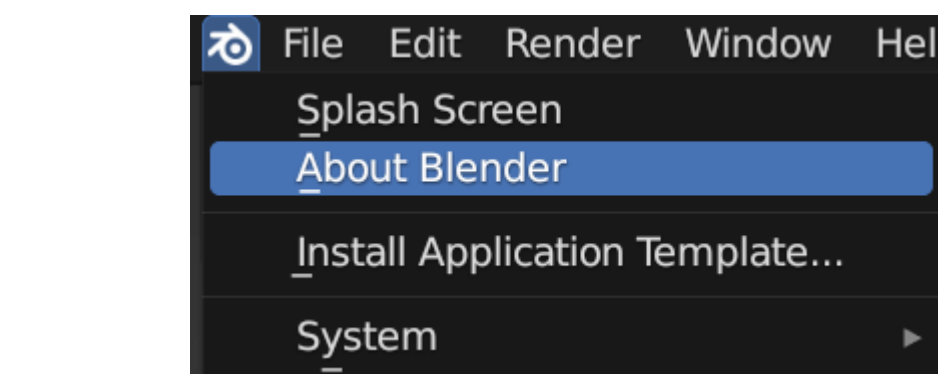
```
Read prefs: C:\Users\ \AppData\Roaming\Blender Foundation\Blender\3.4\config\userpref.blend
-- reg --
-- new file after --
-- unit system changed --
-- unit length/scale changed --
-- subscribe --
[[[ update_link_data ]]]
[[[ update_link_data ]]]
Traceback (most recent call last):
  File "C:\Users\ \AppData\Roaming\Blender Foundation\Blender\3.4\scripts\addons\IiispaceModEditorV3_0\debug_draw.py", line
632, in invoke
    P = m.bugexample
AttributeError: module 'IiispaceModEditorV3_0.m' has no attribute 'bugexample'
Error: Python: Traceback (most recent call last):
  File "C:\Users\ \AppData\Roaming\Blender Foundation\Blender\3.4\scripts\addons\IiispaceModEditorV3_0\debug_draw.py", line
632, in invoke
    P = m.bugexample
AttributeError: module 'IiispaceModEditorV3_0.m' has no attribute 'bugexample'
```

Attachment: Image/Video (If available). Note that Blender version, Addon version and Traceback are important info to the developer.



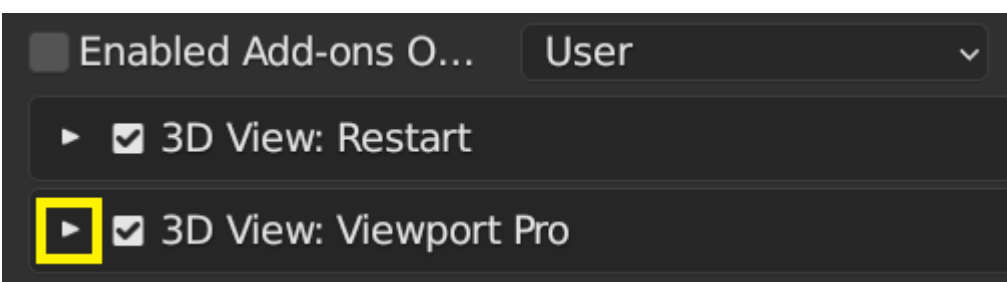
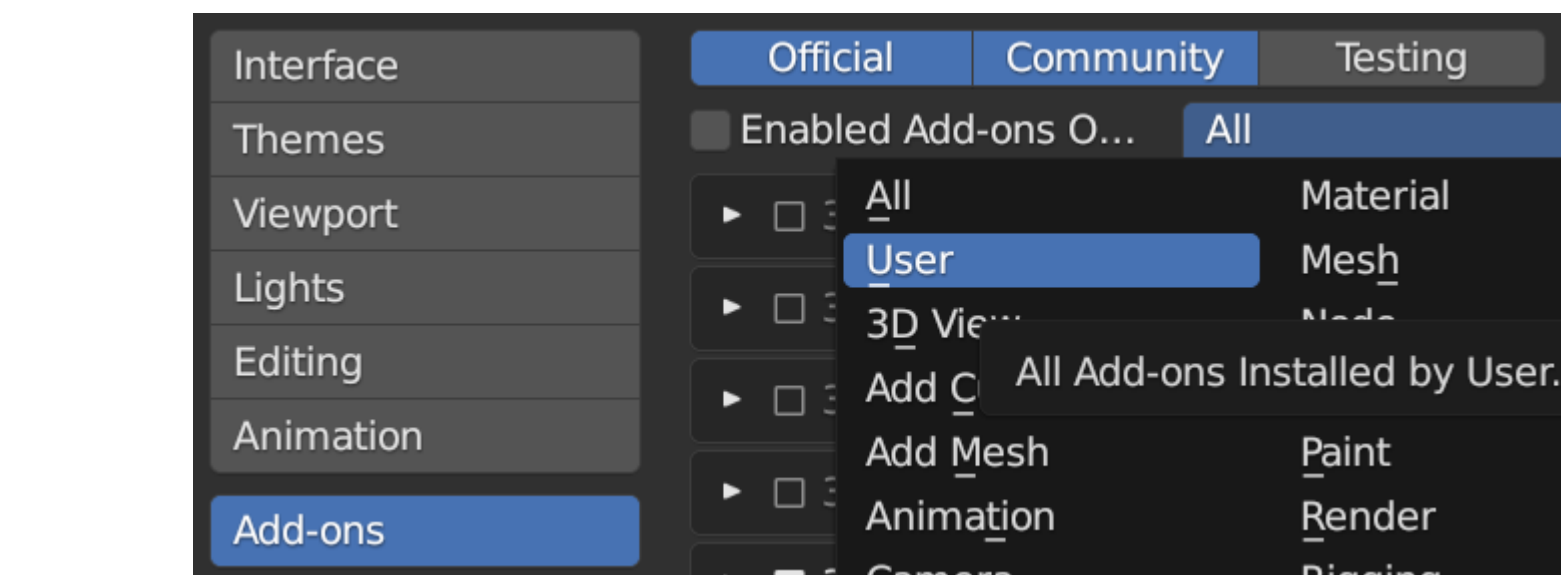
### Get Blender version

Icon > About Blender



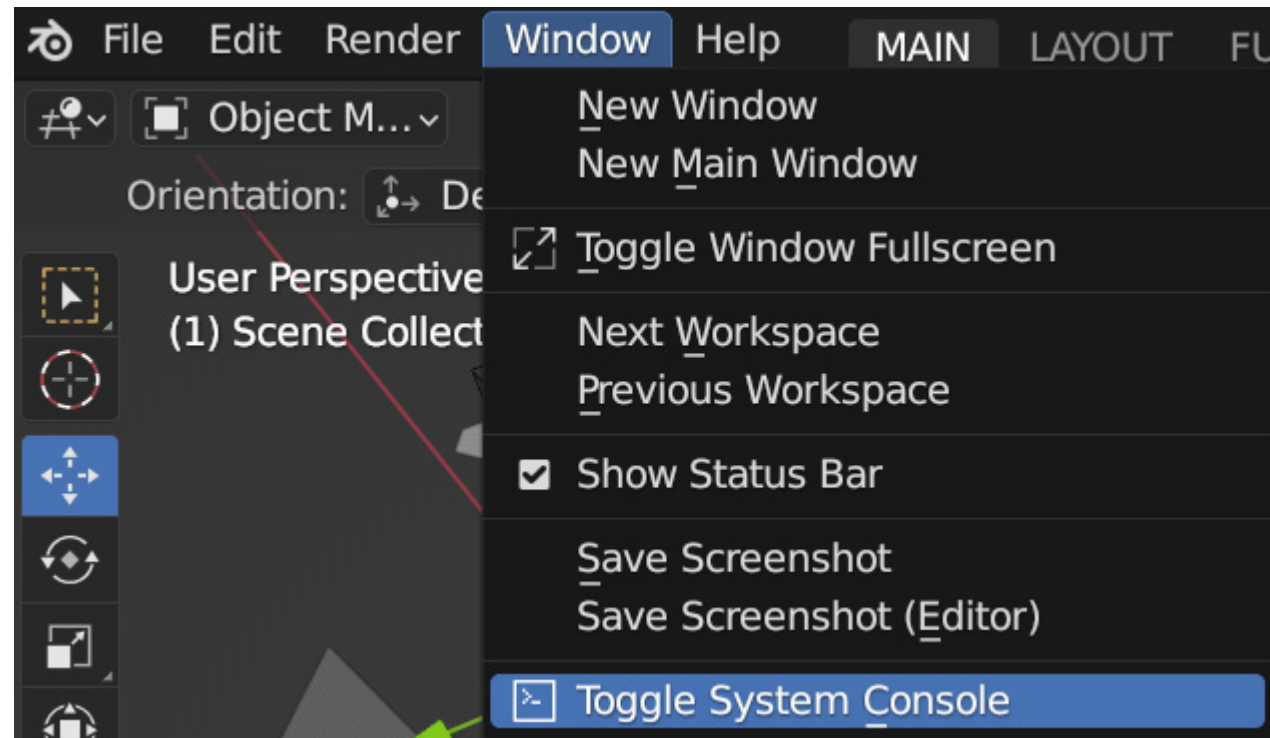
### Get Addon version

Edit > Preferences  
Add-ons > Filter > User  
Expand button



## Get Traceback

Window > Toggle System Console



Copy all text in the Console

## Known issues and limitations

This is the first experimental version of this Add-on, it may be unstable and is for testing only.

1. Unexpected result when editing Included Angle from the Mesh Editor in Blender 3.6.
2. Undo function are not available in N-panel preferences properties. Use the Settings Editor instead.
3. The current version does not support Quad View