BrawlAPI guide

This is a quick-start guide to cover some helpful commands and help anyone get started on BrawlCrate scripts using Python.

- **This is not a detailed programming guide.** Feel free to read through and learn as you go! However for more complex scripting, you'll need some understanding of variables, Python lists, and functions, which won't be explained here.
- For questions or discussion, check #brawlapi in the BrawlCrate server. Please do not ping for support. sooper and I regularly check these channels

Everything I learned myself was from soopercool's Sample Plugins:
https://github.com/soopercool101/BrawlCrateSamplePlugins

...as well as the API documentation here:
https://soopercool101.github.io/BrawlCrate

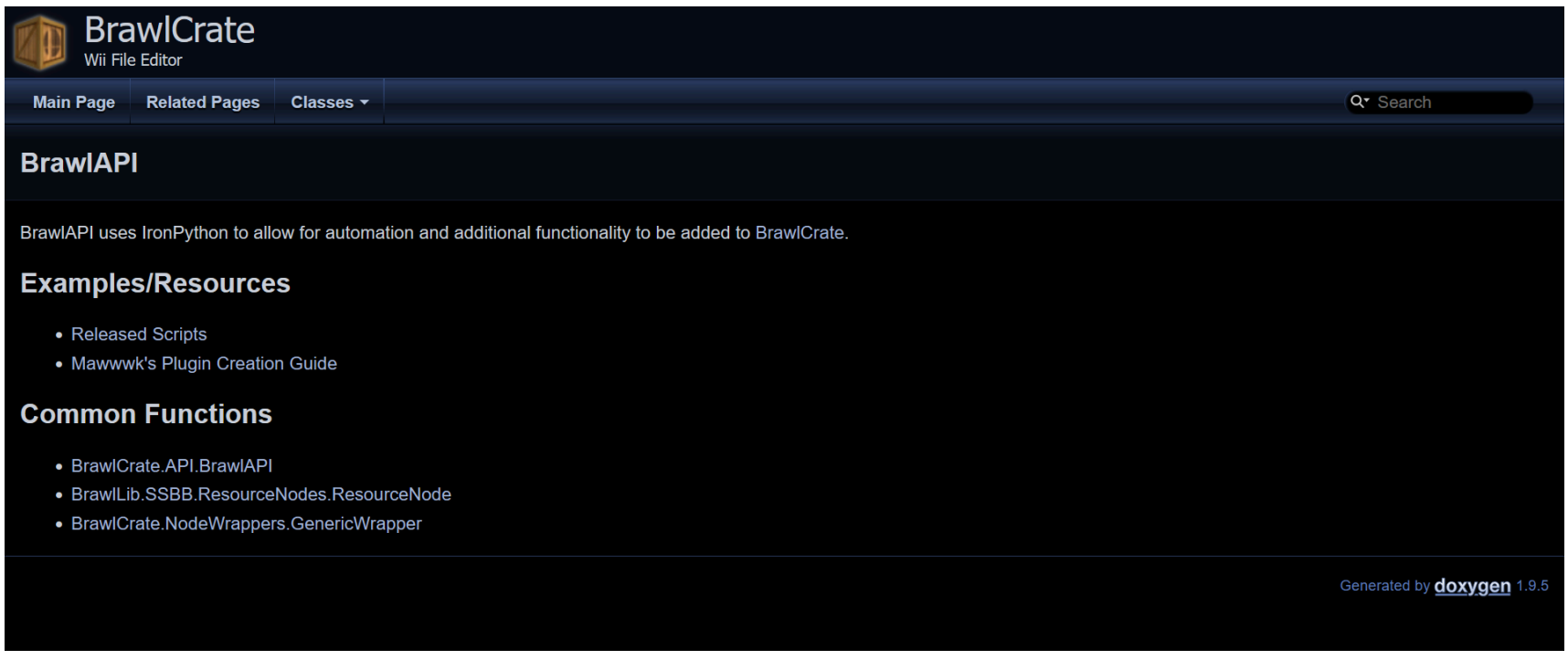
For more examples, my own plugins can be found here:
https://github.com/markymawk/BrawlCratePlugins

For writing plugins, I like to use Notepad++, but you can use any text editor or IDE (even regular Notepad works).

Section 1 - Basic message box

In BrawlCrate, the items listed on the left side of the program window are called **nodes**.
The values on the right side are its **properties**.
The first example in this guide will show a message box listing the **type** and **name** of the selected node.

Start by opening the API documentation (linked above; or inside the BrawlCrate program, click Help > Documentation).



Under Common Functions, links are listed for the BrawlAPI-specific functions and for **ResourceNode**, which are both widely used in scripts. You can check the **ResourceNode** documentation, and scroll down on the page to see the functions and properties available.

1. Copy an existing plugin .py file (or create a new .py file), and put it in BrawlCrate/BrawlAPI/Plugins. Name it **Example.py** or something similar.
Inside BrawlCrate, click Tools > Reload Plugins (or press Alt, T, R), then the new Example plugin will appear inside the Plugins menu.

2. Open Example.py in a text editor. Per convention, start with these lines to the top of the Python file:

```
__author__ = "yourname"
__version__ = "1.0"

from BrawlCrate.API import *
from BrawlLib.SSBB.ResourceNodes import *
```

The "from" lines (aka *import statements*) allow the script to use **BrawlAPI** commands, which make up the basis of all plug-ins.

3. Going back to the BrawlAPI docs for BrawlCrate.API.BrawlAPI, there's some key properties and functions we'll use for this example: **SelectedNode** and **ShowMessage()**.

SelectedNode returns the node currently selected in BrawlCrate.
As a shortcut, I often like to store it as **selNode** (or sometimes just **node**):

```
selNode = BrawlAPI.SelectedNode
```

4. With the selected node now accessible to us, we can access and edit any property listed in the ResourceNode documentation.

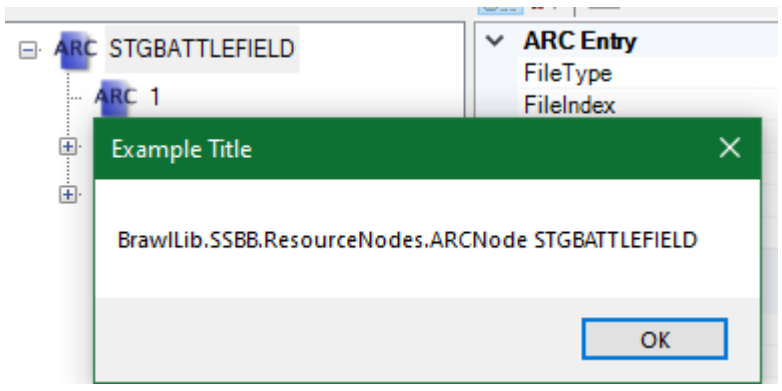
To store strings of the node type and name, use:

```
nodeType = selNode.NodeType
nodeName = selNode.Name

# Alternatively we can use Python type():
# nodeType = str(type(selNode))
```

To show the message box:

```
BrawlAPI.ShowMessage(nodeType + " " + nodeName, "Example Title")
```

5. Save these lines in the Example.py file, then click **Tools > Reload Plugins** to update the list.
I'll select the root node of STGBattlefield.pac, and run the script.
The resulting message box shows the type (ARCNode) and name (STGBATTLEFIELD):



[View the full script]

The above example utilizes properties of ResourceNode, a class that nearly every **node** inherits from. For more specific functions, use the search bar to find more specialized nodes.

If you like, check the documentation for MDL0Node or CHR0Node. These use lots of specialized functions, but also **inherit** functions and properties from their parent classes (ResourceNode).

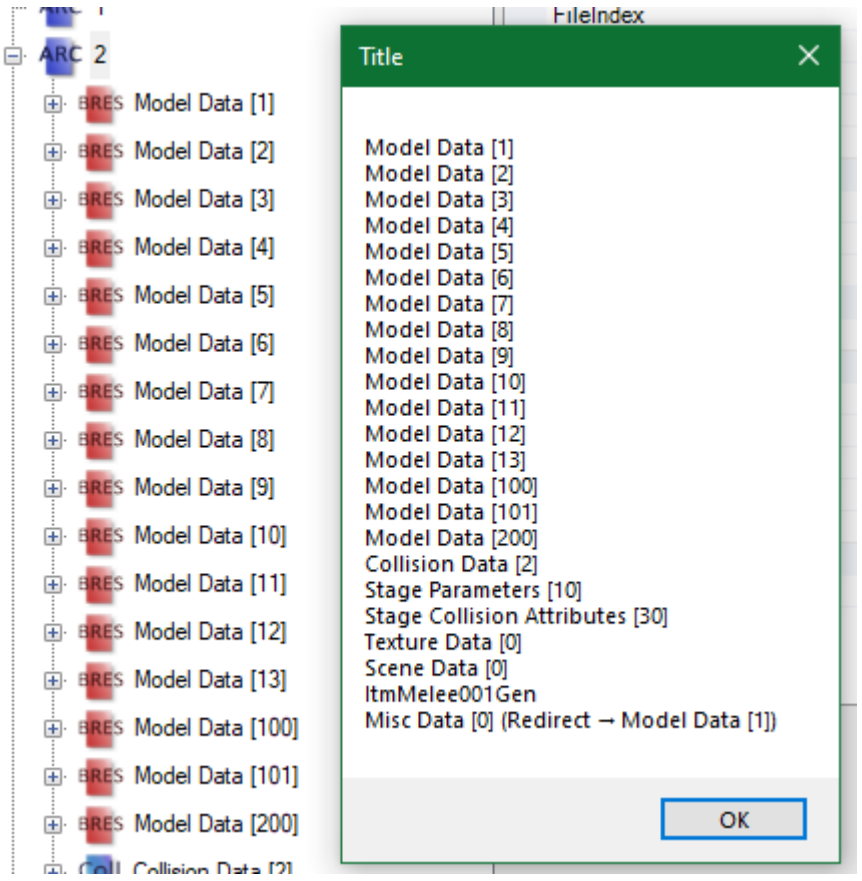
Section 2 - Child & parent nodes

For this example, we'll show a dialog box that lists all child names of the SelectedNode, then edit some node properties.
A node's children can be accessed as a list using `node.Children`.
Similarly, use `node.Parent` to access a parent node.

Using a **for-loop** to iterate through the child list, store each child's name in a single string:

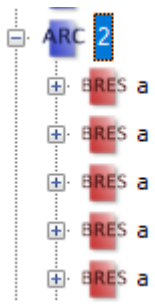
```
selNode = BrawlAPI.SelectedNode
outputMsg = ""
for node in selNode.Children:
     outputMsg += node.Name + "\n"

BrawlAPI.ShowMessage(outputMsg, "Title")
```

Running this with the 2 ARC selected in STGBattlefield.pac shows this message:



Within this list, node properties such as the name can be directly edited.
For example, this loop changes all of the child names at once:

```
for node in BrawlAPI.SelectedNode.Children:
    node.Name = "a"
```



(This can be undone by right-clicking the 2 ARC and clicking Restore, or Ctrl+T)

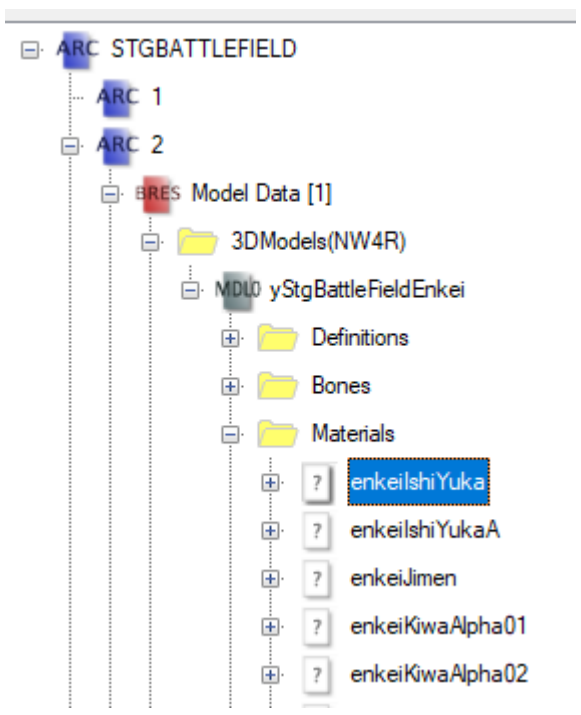
To rename only BRRES nodes, add an if-statement before the rename:

```
if isinstance(node, BRRESNode):
```

To avoid errors, check for the existence of a node's children using `if node.HasChildren` before looping through them.

Section 2.5 - Accessing a specific node

This example mashes together what's been covered so far. We'll access specific nodes inside Python, but using methods other than SelectedNode.
This example script will rename each material in a given model to have the same name as its texture reference.



First, access this "Materials" BRES group via code.
It can be retrieved through one of the following methods:

1. Navigate by **children indices**:
   (Succinct, but not very flexible)

```
brres = BrawlAPI.RootNode.Children[1].Children[0]
mdl0 = MODEL_DATA_BRRES.Children[0].Children[0]
matsGroup = MODEL.Children[2]
```

2. Navigate by **node name** using FindChild():
   (More exact, but much lengthier to type)

```
brres = BrawlAPI.RootNode.FindChild("2").FindChild("Model Data [1]")
mdl0 = brres.FindChild("3DModels(NW4R)").FindChild("yStgBattleFieldEnkei")
matsGroup = mdl0.FindChild("Materials")
```

3. Assume the Materials group is the **selected node**:

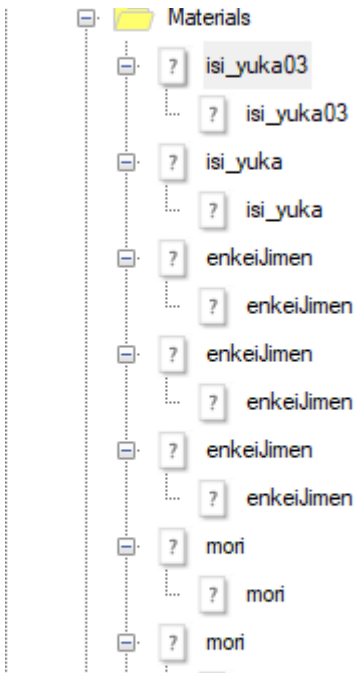
```
matsGroup = BrawlAPI.SelectedNode
```

   Out of these three methods, the shortest and easiest to write is clearly option #3. However, it only works if the Materials group is selected every time we run it, which adds a step at runtime. This option isn't ideal for user-friendliness; I'd only recommend this method for quick personal scripts. Use whichever method best fits your use case.

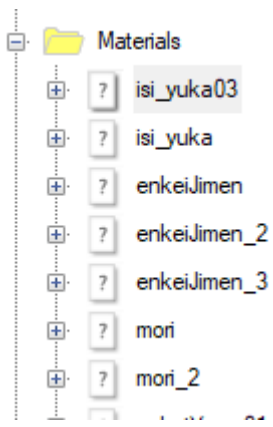
Next, fill out the for-loop.
Any materials that don't use textures should be skipped, so add a HasChildren check for each material:

```
for mat in matsGroup.Children:
     if mat.HasChildren:
          mat.Name = mat.Children[0].Name
```

Reload plugins, then run the script to change all material names at once!



While this works without errors, having multiple mats with identical names can be a problem (or at the very least, is sloppy).
I'll fix that by numbering duplicate materials, using a list to store any repeating names.



View the full script

Section 3 - NodeListOfType

To get a Python list of all nodes of a certain type (models, materials, objects, textures, you name it), use **BrawlAPI.NodeListOfType[]()**.
In this example, I'll set all materials in Battlefield to Cull_Inside.

1. Material nodes exist as type **MDL0MaterialNode**, so start by storing a list of all materials in the file:

```
matList = BrawlAPI.NodeListOfType[MDL0MaterialNode]()
```

2. Next, loop through every mat to set its culling:

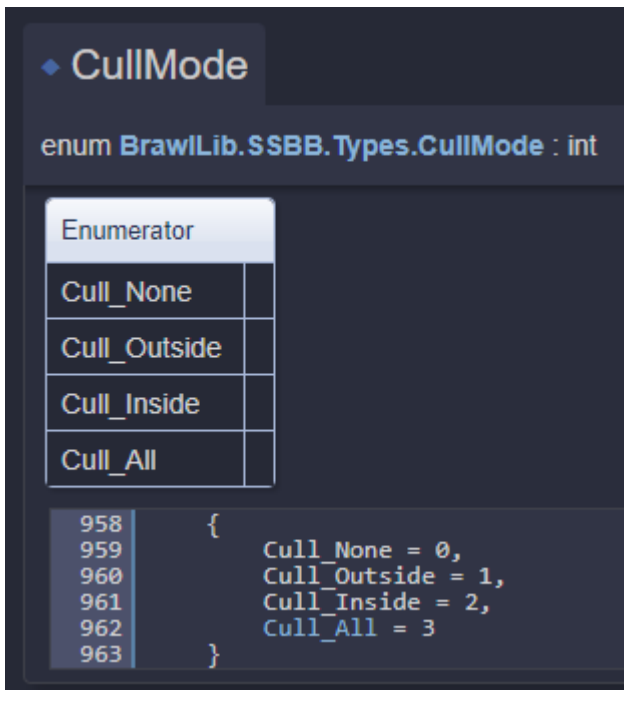
```
for mat in matList:
     mat.CullMode = "Cull_Inside"
```

For confirmation, I'll put a dialog box to say how many materials were checked:

```
BrawlAPI.ShowMessage(str(len(matList)) + " materials set to Cull Inside", "Title")
```

3. Running this gives an error… The problem here is that Cull settings aren't stored as a string, but as an enumerator.

Searching "CullMode" in the API docs will show this table:



To access the above CullMode settings, add the necessary import statement at the top of the script:

```
from BrawlLib.SSBB.Types import CullMode
```

Then change the loop to use:

```
mat.CullMode = CullMode.Cull_Inside
```

[View the full script]
To test this, set a material to Cull_All, then run the script and check if it changes back to Cull_Inside.

For bonus points, figure out how to change the dialog box to count how many materials *weren't* Cull_Inside before running the script, by using an if-statement inside the for-loop.

Section 4 - Handling external files

For external file checking operations, start by adding the following import statement to the top of the script file:

```
from System.IO import *
```

(Avoid using Python's `import os` for BrawlCrate scripts, as this often has compatibility errors)
It's recommended to use the C# System.IO namespace for scripts, which is what's used above.

To check for the existence of a file or folder, use the following line with a folder path string as `DIR_PATH`, and filename string as `FileName`:

```
if File.Exists(DIR_PATH + "\\" + FileName):
```

To store a list of files, use CreateDirectory() as such:

```
FILES_LIST = Directory.CreateDirectory(DIR_PATH).GetFiles()
```

This stores multiple Python *File* objects in a list. For most purposes, we'll use `File.Name` to refer to the file as a string.

To iterate through .pac files inside this folder, use a for-loop as such:

```
for file in FILES_LIST:
    if file.Name.lower().endswith(".pac"):
        # ...
```

For writing to external text files, declare an object with a statement such as:

```
TEXT_FILE = open(TEMP_TEXT_FILE_PATH,"w+", encoding="utf-8")
```

Write a string to this text file using:

```
outputString = "abc"
TEXT_FILE.write(outputString)
```

Many (if not all) strings in Brawl are encoded as UTF-8, so be sure to declare `encoding` like above.

For exporting temporary files, I use AppPath to serve as a temp folder, and use `File.Delete()` to clear them at the end of a script.
The Stock Icon Exporter script below uses this functionality.

Here's some examples that check and/or write data to external files:
Verify Param File Data
Info.pac Stock Icon Exporter

Section 5 - Modifying CHR0 animations

To retrieve a specific frame in a CHR0, use GetKeyframe().
GetKeyframe() takes two arguments, where the first, ArrayIndex, is a digit that represents the following:

0	Scale X
1	Scale Y
2	Scale Z
3	Rotation X
4	Rotation Y
5	Rotation Z
6	Translation X
7	Translation Y
8	Translation Z

The second argument of GetKeyframe() is the frame of the chr0 animation, **zero-indexed**. (For frame 1, use 0)

Example using GetKeyframe() to get the X translation at frame 500:

```
x = chrNode.GetKeyframe(6, 499)
# To get the value as a float, use:
xValue = x._value
```

If no keyframe is set at the given frame index, then GetKeyframe() will return null (**None** in Python). To get the value as a float regardless of whether a keyframe is set, use **GetFrameValue()** with the same arguments.

Example: using **SetKeyframe()** to move all keyframes backward in the Z-axis by 1.0

```
for k in range(node.FrameCount):
     frame = node.GetKeyframe(8, k)
     # 8 = Translation Z

     # If keyframe is empty, skip
     if frame is None:
          continue

     # Get new value as float
     newZ = frame._value - 1.0
     node.SetKeyframe(8, k, newZ)
```

SetKeyframe() can also be used to add new keyframes, even if one wasn't there prior.
To loop through all frames and values in a CHR0, use a nested loop such as:

```
for k in range(node.FrameCount):

  # Loop through scale, rotation, translation
  for i in range (0, 9):
      frame = node.GetKeyframe(i, k)

      # If keyframe is empty, skip
      if frame is None:
           continue

     # Operations here
     # ...
```

Section 6 - Utilizing NodeWrappers

Some API functions are not directly accessible via ResourceNode, but instead by its **node wrapper**. Wrapper functions often have less to do with the node data itself and are often used for UI functions, most commonly Duplicate().

With a wrapper stored as **wrapper**, use **wrapper.Resource** to get the node inside. There's no method to get a wrapper from a node, though.

A wrapper must be visible in the node list for it to be accessed, meaning any parent nodes must be expanded out.
Node wrappers can be expanded via API using `yourWrapper.Expand()`

Below are a few common node functions and their corresponding wrapper equivalent:

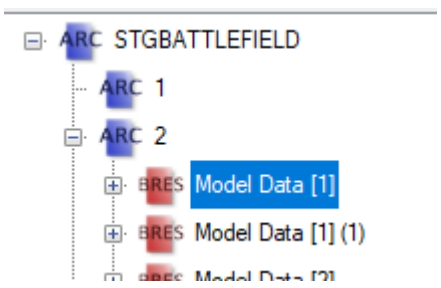
Node	**Wrapper**
BrawlAPI.SelectedNode()	BrawlAPI.SelectedNodeWrapper()
NodeListOfType[*yourType*]()	NodeWrapperListOfType[*yourType*]()
BrawlAPI.RootNode	BrawlAPI.RootNodeWrapper
node.ReplaceFromFolder()	wrapper.ReplaceAll()
node.Children[i] *Returns child node at index i*	wrapper.Nodes[i] *Returns child wrapper at index i*

To search for a child wrapper by node name, feel free to use this custom function:

```
# ---
# findChildWrapperByName()
# Given any nodeWrapper, return its child wrapper whose Resource.Name contains the given nameStr
def findChildWrapperByName(wrapper, nameStr, EXACT_NEEDED=False):
    if wrapper.Nodes:
        for child in wrapper.Nodes:
            if EXACT_NEEDED and child.Resource.Name == str(nameStr):
                return child
            elif str(nameStr) in child.Resource.Name:
                return child
    return 0   # If not found, return 0
# ---
```

If I wanted to duplicate the Model Data [1] BRRES inside STGBattlefield.pac, the full code would be:

```
arcWrapper = BrawlAPI.RootNodeWrapper.Nodes[1]
brresWrapper = arcWrapper.Nodes[0]
# Alternatively, use findChildWrapperByName() from above
brresWrapper = findChildWrapperByName(arcWrapper, "Model Data [1]")
newBrresNode = brresWrapper.Duplicate()
```



Section 7 - Modifying vertices

Vertex data is stored in a MDL0VertexNode using a Vector3[] list. However, this list seems to be immutable via API edits (in other words, it can't be modified).
This means we must replace it with a new list entirely, making a new list using a C# Array object.

Below is an example script to set the Z value of all vertices to 0, using an added import statement for the Array class:

```
from BrawlCrate.API import *
from BrawlLib.SSBB.ResourceNodes import *
from BrawlLib.Internal import * # Vector3 etc
from System import Array

selNode = BrawlAPI.SelectedNode
oldVerts = selNode.Vertices
newVerts = []

# Example to set Z values to 0. Need to save and re-open to get proper effect in a model
preview
for vertex in oldVerts:
    newVert = Vector3(vertex._x, vertex._y, 0)
    newVerts += [newVert]

selNode.Vertices = Array[Vector3](newVerts)
```