



# H13 - HDA Development Part 03 & 04

VOP HDAs, Python, and Script Nodes

# Agenda

## Part 03

Python SOP

Unix SOP (Python Script)

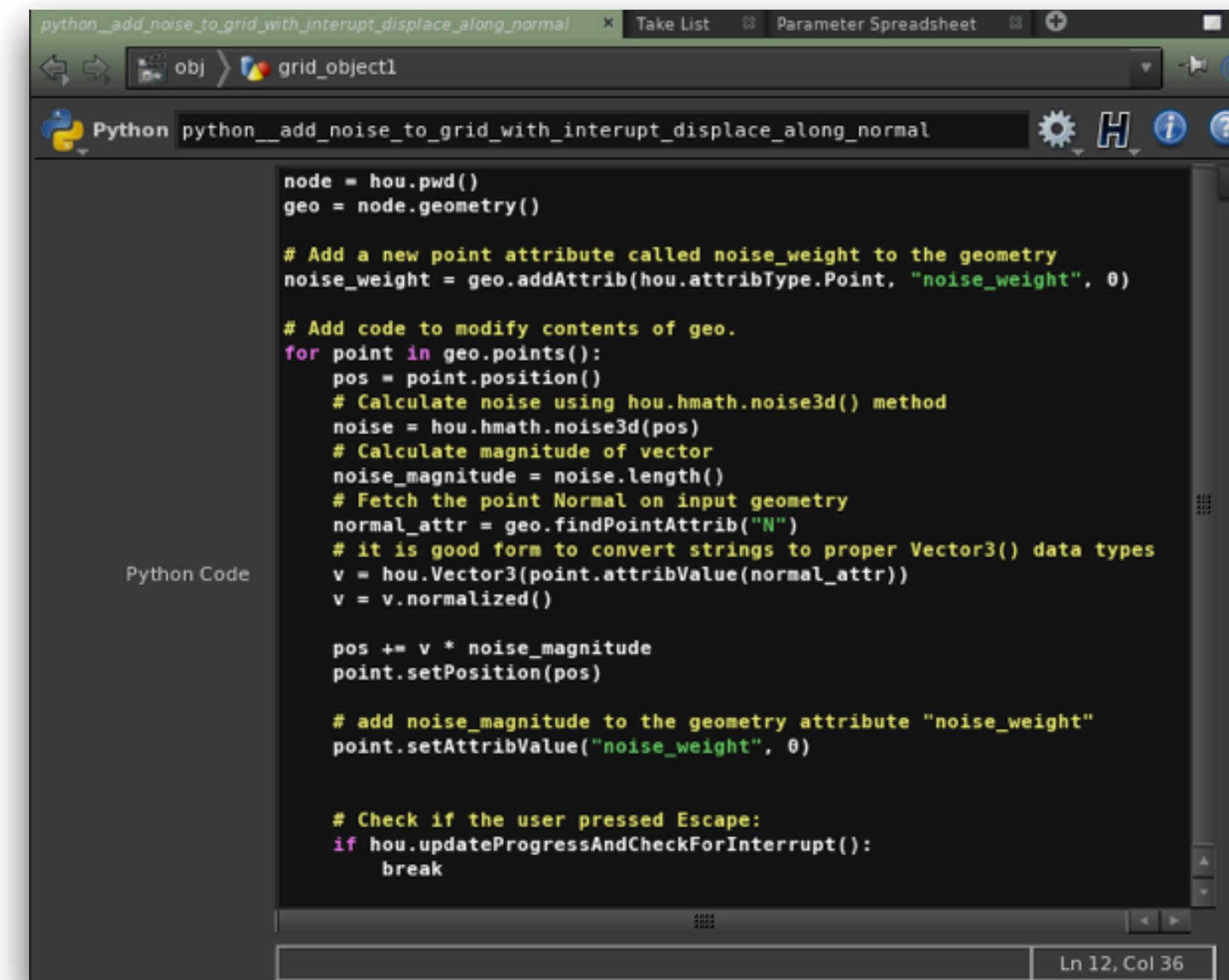
Script SOP (HScript)

## Part 04

Creating a Shelf Tool (Python)

Creating a VOP Node (VEX Script)

Help Cards Part 02

A screenshot of a Python Shell window in Houdini. The window title is 'python\_add\_noise\_to\_grid\_with\_interrupt\_displace\_along\_normal'. The code is written in a dark-themed editor. It defines a node and its geometry, then iterates through points to add a 'noise\_weight' attribute and displace points based on noise. The status bar at the bottom right shows 'Ln 12, Col 36'.

```
node = hou.pwd()
geo = node.geometry()

# Add a new point attribute called noise_weight to the geometry
noise_weight = geo.addAttrib(hou.attribType.Point, "noise_weight", 0)

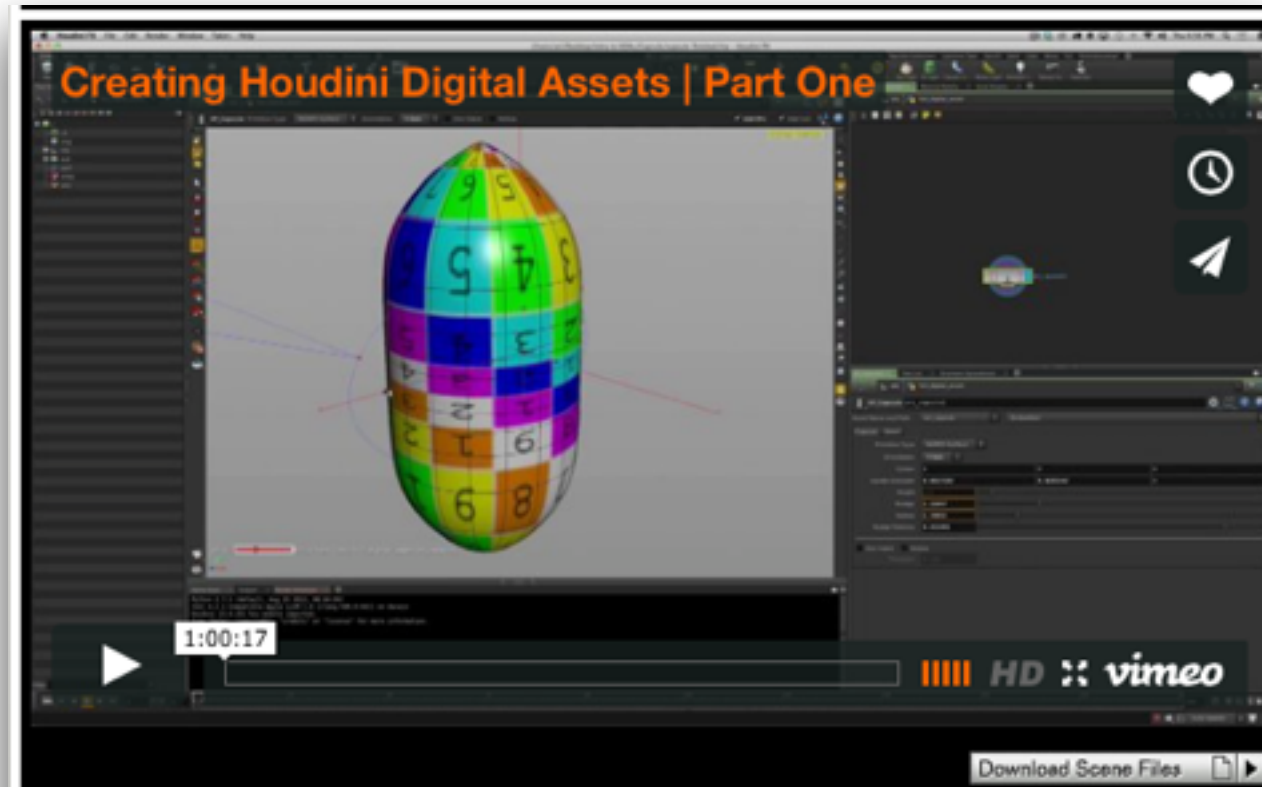
# Add code to modify contents of geo.
for point in geo.points():
    pos = point.position()
    # Calculate noise using hou.hmath.noise3d() method
    noise = hou.hmath.noise3d(pos)
    # Calculate magnitude of vector
    noise_magnitude = noise.length()
    # Fetch the point Normal on input geometry
    normal_attr = geo.findPointAttrib("N")
    # it is good form to convert strings to proper Vector3() data types
    v = hou.Vector3(point.attribValue(normal_attr))
    v = v.normalized()

    pos += v * noise_magnitude
    point.setPosition(pos)

# add noise_magnitude to the geometry attribute "noise_weight"
point.setAttribValue("noise_weight", 0)

# Check if the user pressed Escape:
if hou.updateProgressAndCheckForInterrupt():
    break
```

# Pre-Requisites



## Creating Digital Assets

Published: Feb 14, 2014 for Houdini 13

This lesson is a good introduction to learning how to assetize Houdini networks which is essential for the Technical Director or those using Houdini to create Digital Assets for Houdini Engine. Broken into two, one hour sessions, this lesson gives you an in-depth look at the different aspects of building assets interactively. Using an existing node network that is ready to be converted into a digital asset you will identify parameters that can be promoted to build the asset's artist-friendly user interface.

Once a digital asset is created, you will learn a couple of different methods to link to icons and embed them. You will then create parameters from scratch that are not in the network, and learn how to create range values and default values. You will continue by learning how to create menus, and how to disable parameters when toggles are selected. The video will go into how to create comments, tool, tips, and help cards. Finally you will learn how to create Guide Geometry and Handles to manipulate the digital asset in the viewport.



The scene files for this lesson are available using the button at the top right just under the embedded video.

## Are You New To Houdini?

If you are new to Houdini and would like to follow through these lessons, be sure to download the [FREE Houdini Apprentice](#) edition then go through the [First Steps Intro](#) lessons.

## Instructor | Ari Danesh

Ari Danesh is a Visual FX artist and Interactive Designer with 12 years experience teaching at the University level. He has taught at UCLA Extension, Moorpark College, and most recently CSUN (California State University at Northridge) and Ai Hollywood. His current focus is in Procedural Modeling and Animation, Particle Systems, Dynamics and Plant Simulation. Ari Danesh is a SideFX Houdini instructor.

Familiarity with SOPs and VOPs

Some knowledge of Python, VEX, and HScript

Understanding of contents in:

Creating Houdini Digital Assets Part 01 & Part 02

[http://www.sidefx.com/index.php?option=com\\_content&task=view&id=2677&Itemid=132](http://www.sidefx.com/index.php?option=com_content&task=view&id=2677&Itemid=132)

**SIDE EFFECTS**  
**SOFTWARE**

# Disclaimer

**I am no expert in Python or Shell Scripting**

**Many of you can run circles around me in your knowledge**

**This workshop focuses on how to use Python, VEX, and HScript to create digital assets and inline code for Houdini**





# Part 03

Inline Tools



# Python SOP

Just working are way up to HDAs

# What is the Python SOP?



## Python surface node

*Runs a Python snippet to modify the incoming geometry.*

This node is designed to modify the input geometry. If you just want to run a script whenever the network cooks, see the [Script SOP](#).

This node lets you modify geometry in a network using a quick ad-hoc script. To create a new, reusable geometry node type using python, see [creating a geometry SOP with Python](#).

See [editing geometry using python](#) for information on how to write the script.

### Parameters

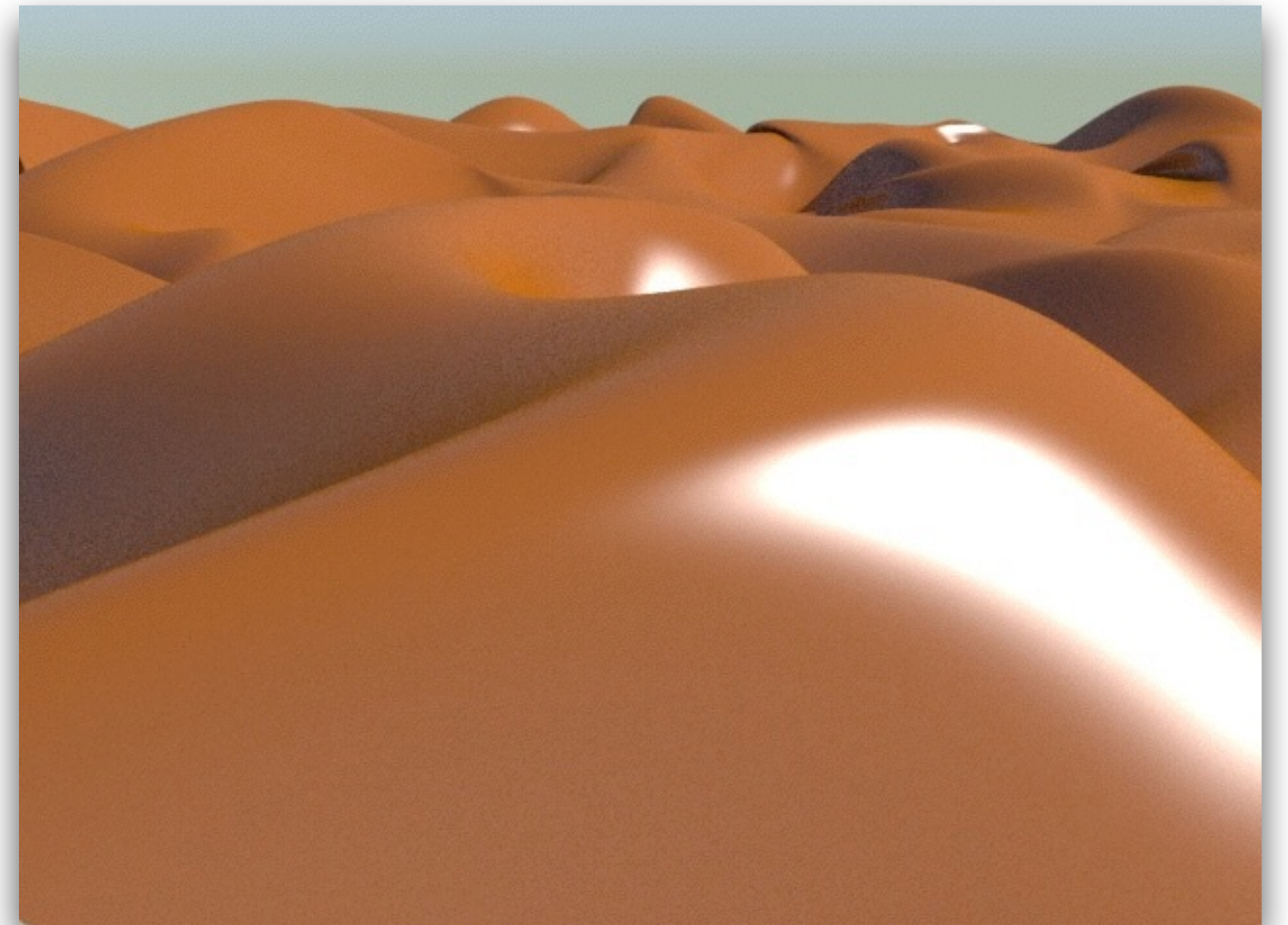
Python Code	
	Python code should modify the incoming geometry.

# What do we want to accomplish with the Python SOP?

Just to get our feet wet with some of the different modules in Python you we will recreate the Mountain SOP

“I know, I know... How many times will we do a Mountain SOP “

I promise better examples as the workshop progresses





# Setup

At the Object Level drop down a Grid - name: Surface\_To\_Deform

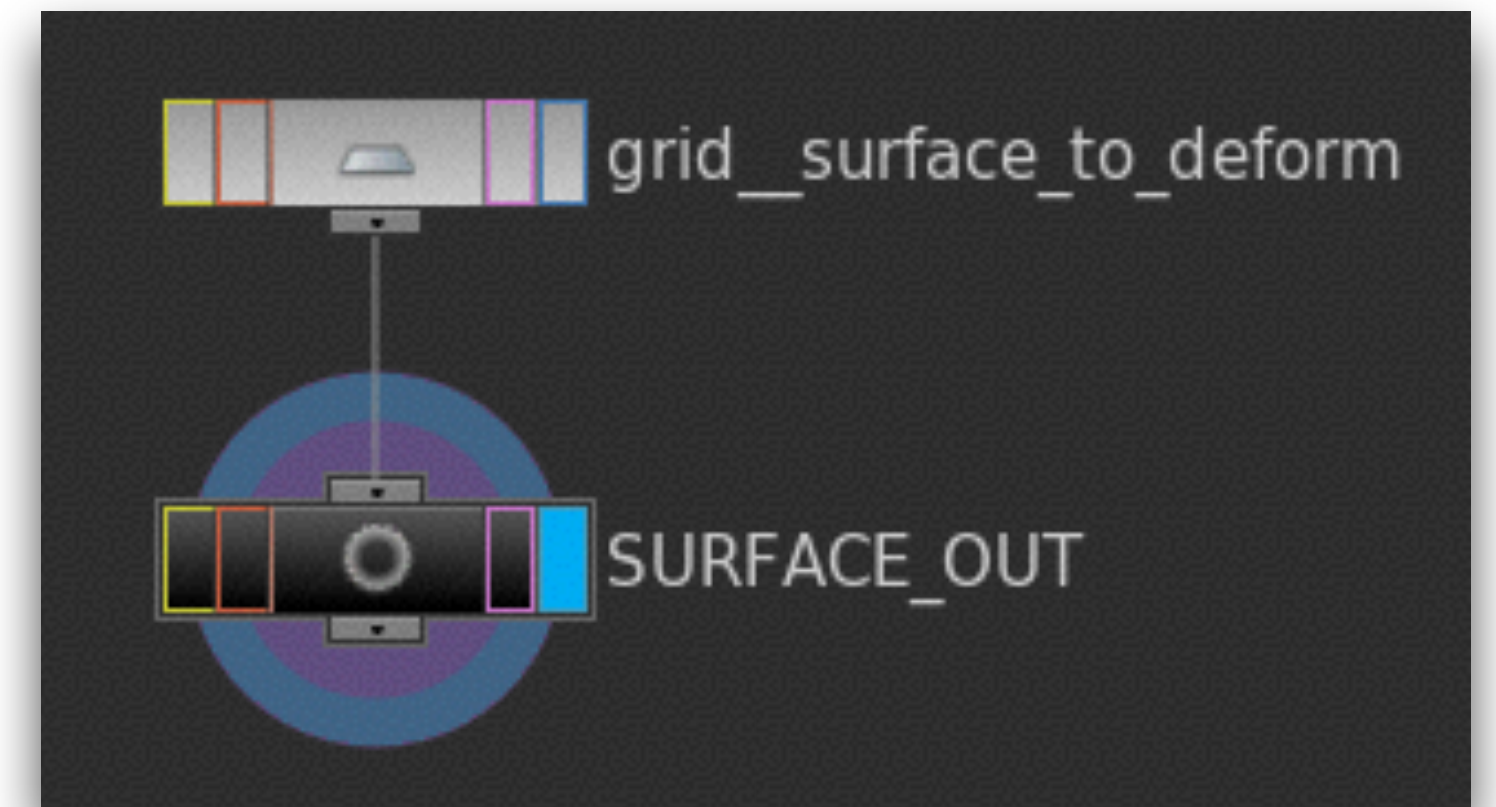
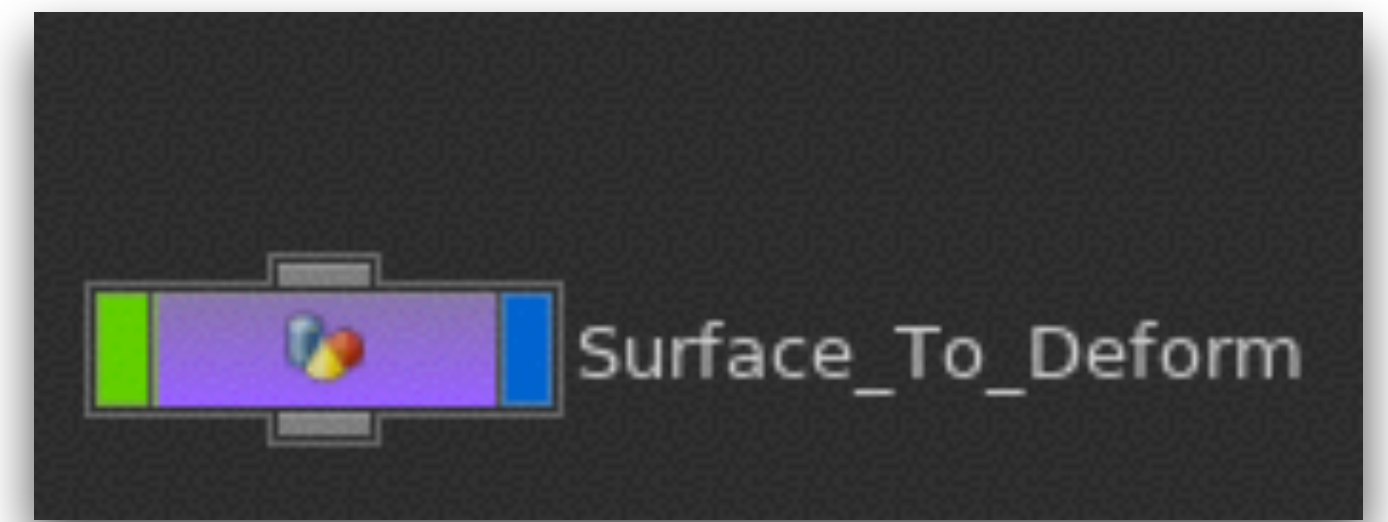
Dive Inside

Grid - ZX plane

Size 10,10

Divisions 100,100

Append a NULL, Name - SURFACE\_OUT



## Setup (cont.)

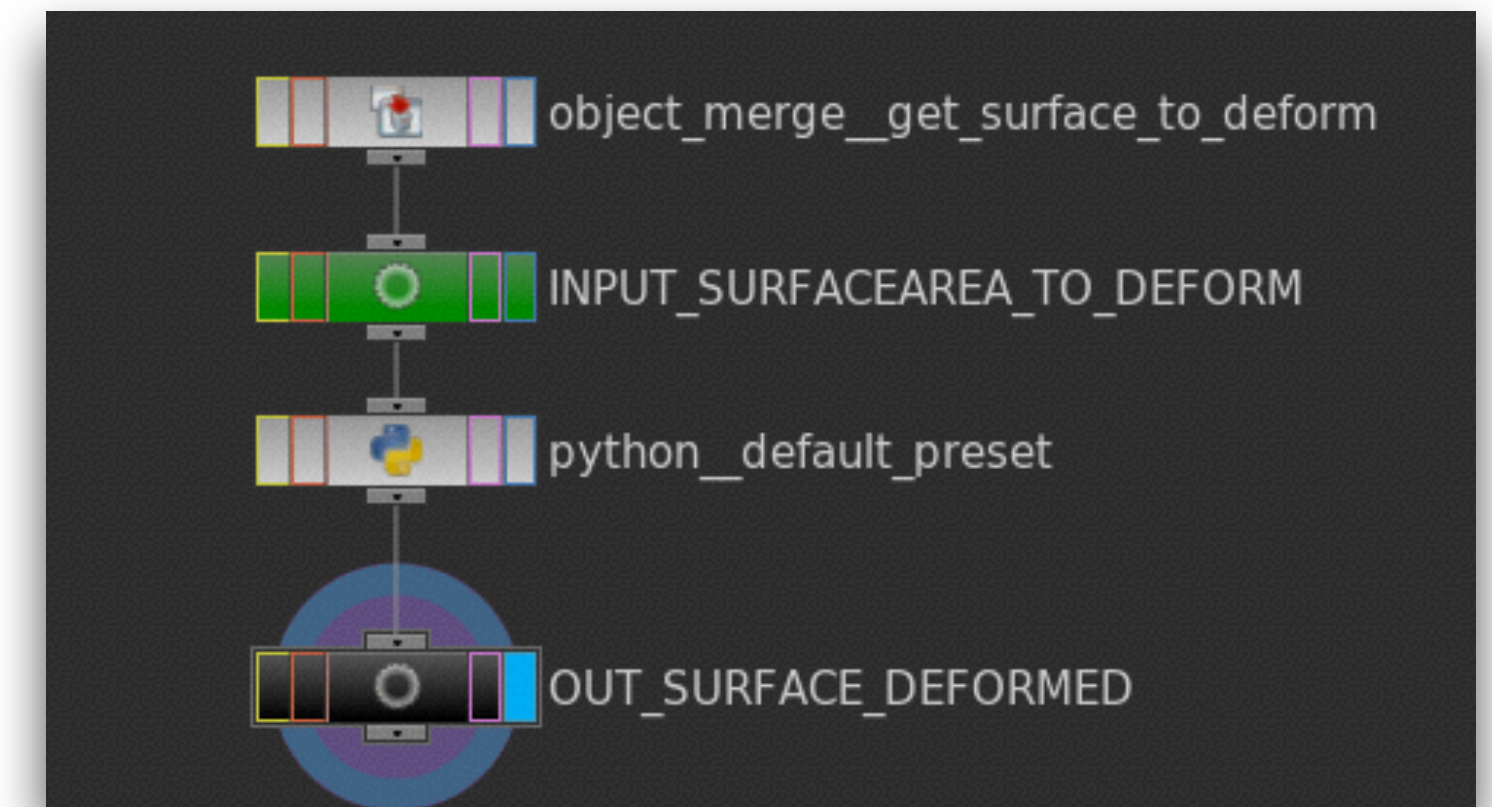
At the Object Level drop down a Geometry Object - name: Surface\_Deformer

Dive Inside

Drop down an object merge - Point to SURFACE\_OUT

Append a Python SOP

(Optional) - Append a Null, Name: OUT\_SURFACE\_DEFORMED



# Looking at the Python SOP

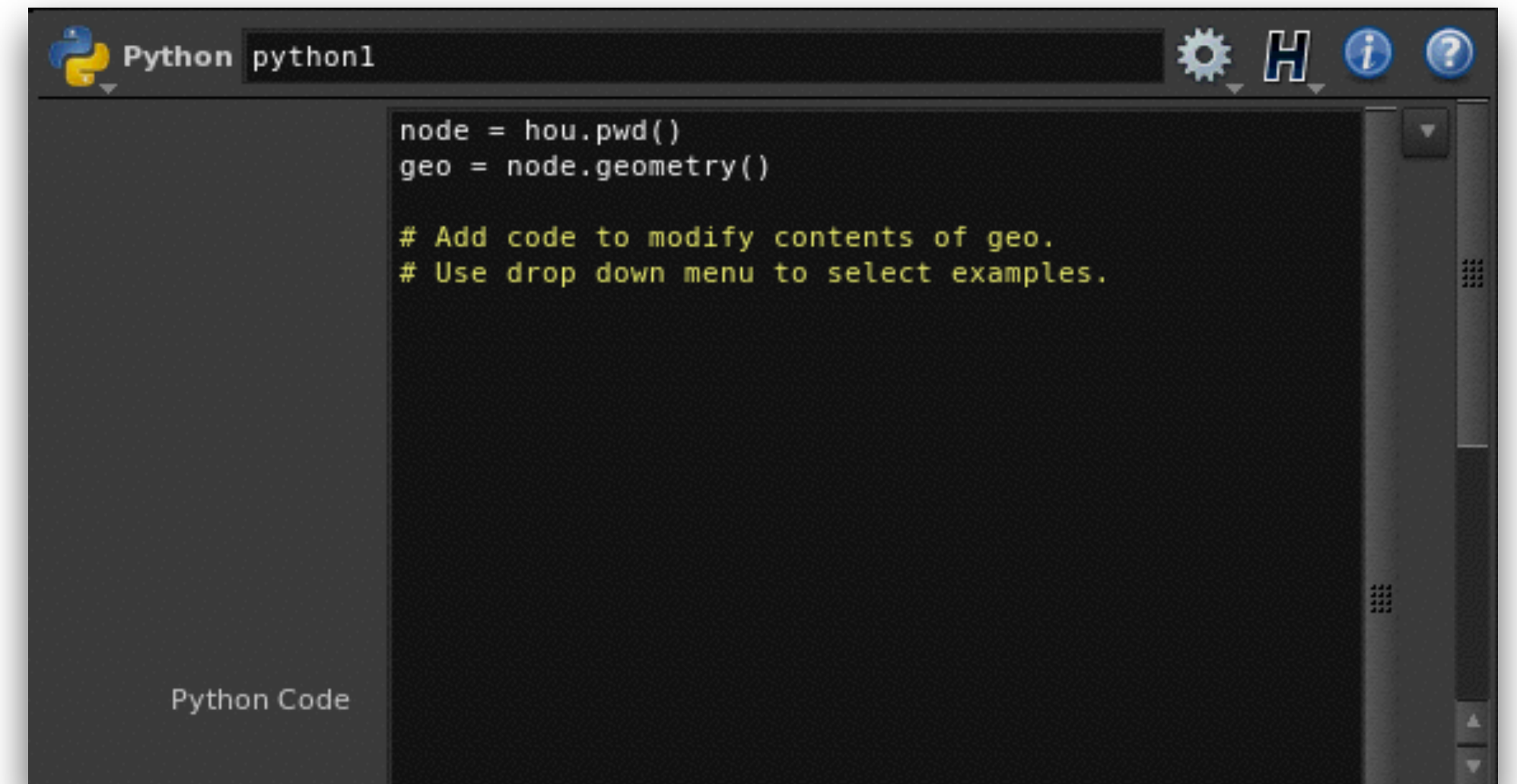
The Python Text Edit area has two lines of code already entered:

```
# get the current node
```

```
node = hou.pwd()
```

```
# get the geometry in the current node
```

```
geo = node.geometry()
```



Module containing all the sub-modules, classes, and functions to access Houdini

`hou.pwd()` - If called from an evaluating parm, return the node containing the parm. Otherwise, return Houdini's global current node. You can change this current node with `hou.cd`

This function is a shortcut for writing `hou.node(".").`



# hou.Geometry



## hou.Geometry class

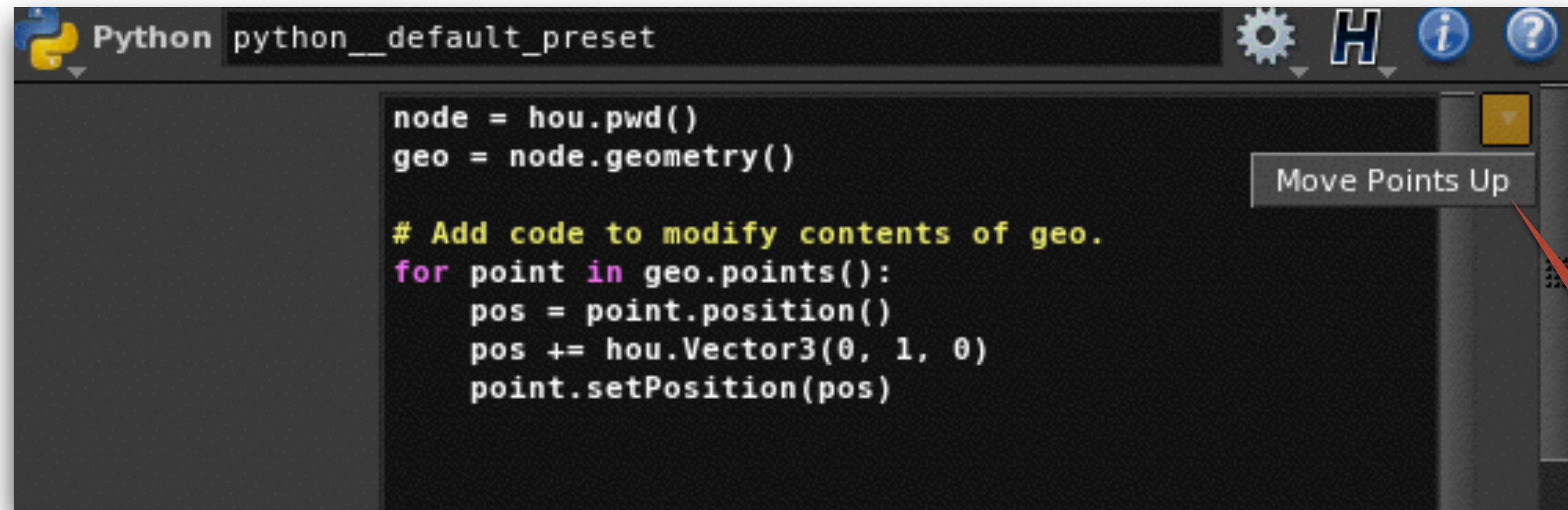
*A Geometry object contains the points and primitives that define a 3D geometric shape. For example, each SOP node in Houdini generates a single Geometry object.*

If you ask a SOP for its geometry via [hou.SopNode.geometry](#), you'll get a read-only reference to it. If the SOP recooks, the corresponding Geometry object will update to the SOP's new geometry. If the SOP is deleted, accessing the Geometry object will raise a [hou.ObjectWasDeleted](#) exception. If you call methods that try to modify the geometry, they will raise a [hou.GeometryPermissionError](#) exception.

If you do not want the geometry to update when the SOP recooks, you can call [hou.Geometry.freeze](#). freeze returns another Geometry object that will not change when the SOP recooks. Accessing frozen Geometry is slightly faster, since Houdini does not need to look up the SOP node for each access, so you may want to use frozen geometry for speed-crucial operations.

If you're writing a SOP using Python, you will have read-write access to the geometry, and it will be frozen. To create a Python-defined SOP, select **File > New Operator Type...** and place the Python code in the **Code** tab.

Finally, you can allocate a new frozen geometry with read-write access by creating an instance of [hou.Geometry](#).



## Step 01 - Preset Menu

Click on the preset menu and select “Move Points Up”

The code in the text editor updates

`geo.points()` # create a tuple of all points in the geometry

`hou.Vector3(0,1,0)` # A sequence of 3 floating point values, with associated mathematical operations.

`point.setPosition(pos)` update the original point position to the new location



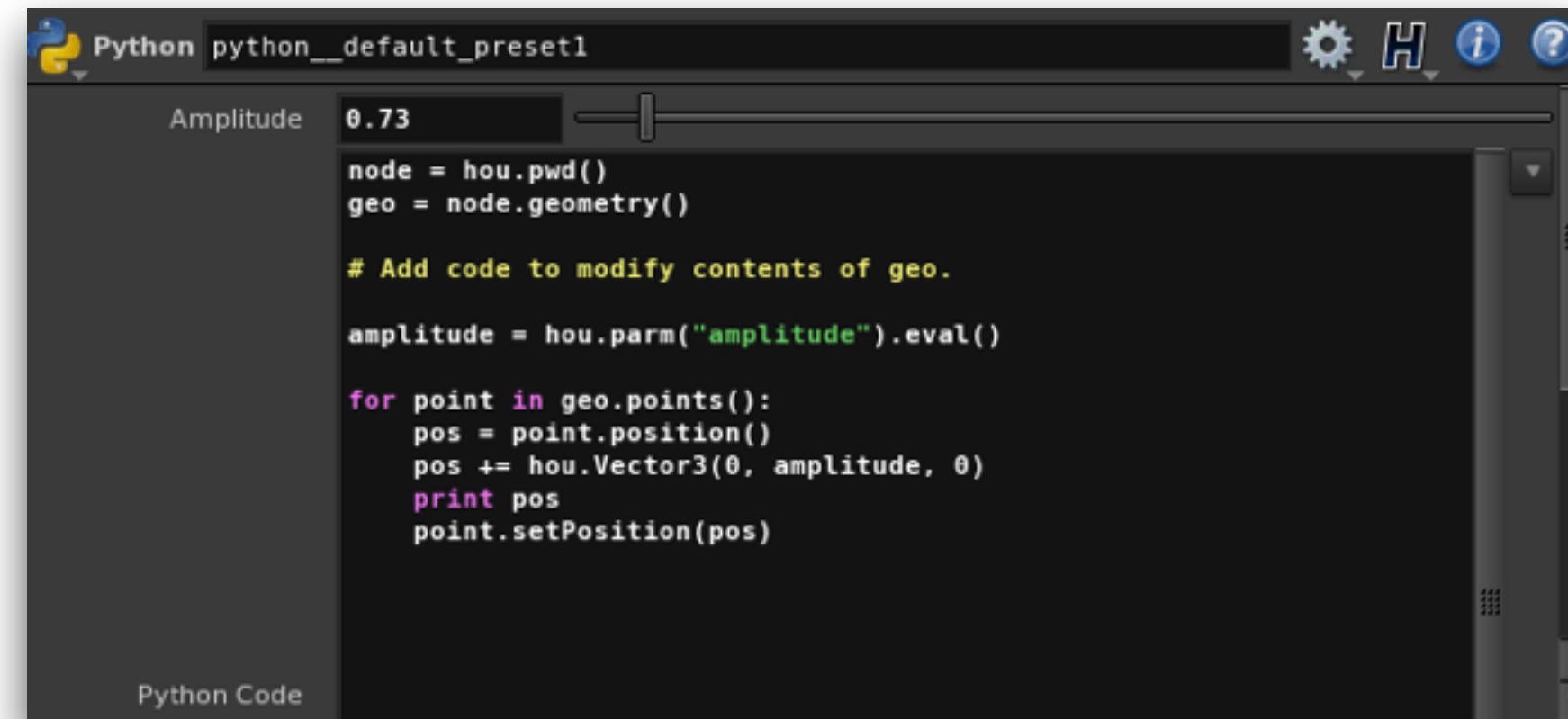
# Step02 - Add User Control

Create a spare parameter

name - amplitude

label - Amplitude

Modify Code - To read the spare parameter



```
# hou.parm() - Given a path string, return a Parm object.
```

```
# Return None if the path does not refer to a parameter.
```

```
amplitude = hou.parm("amplitude").eval()
```

```
# eval() - Evaluates this parameter at the current frame and
returns the result.
```

```
#updating the old position to the new position
```

```
point.setPosition(pos)
```

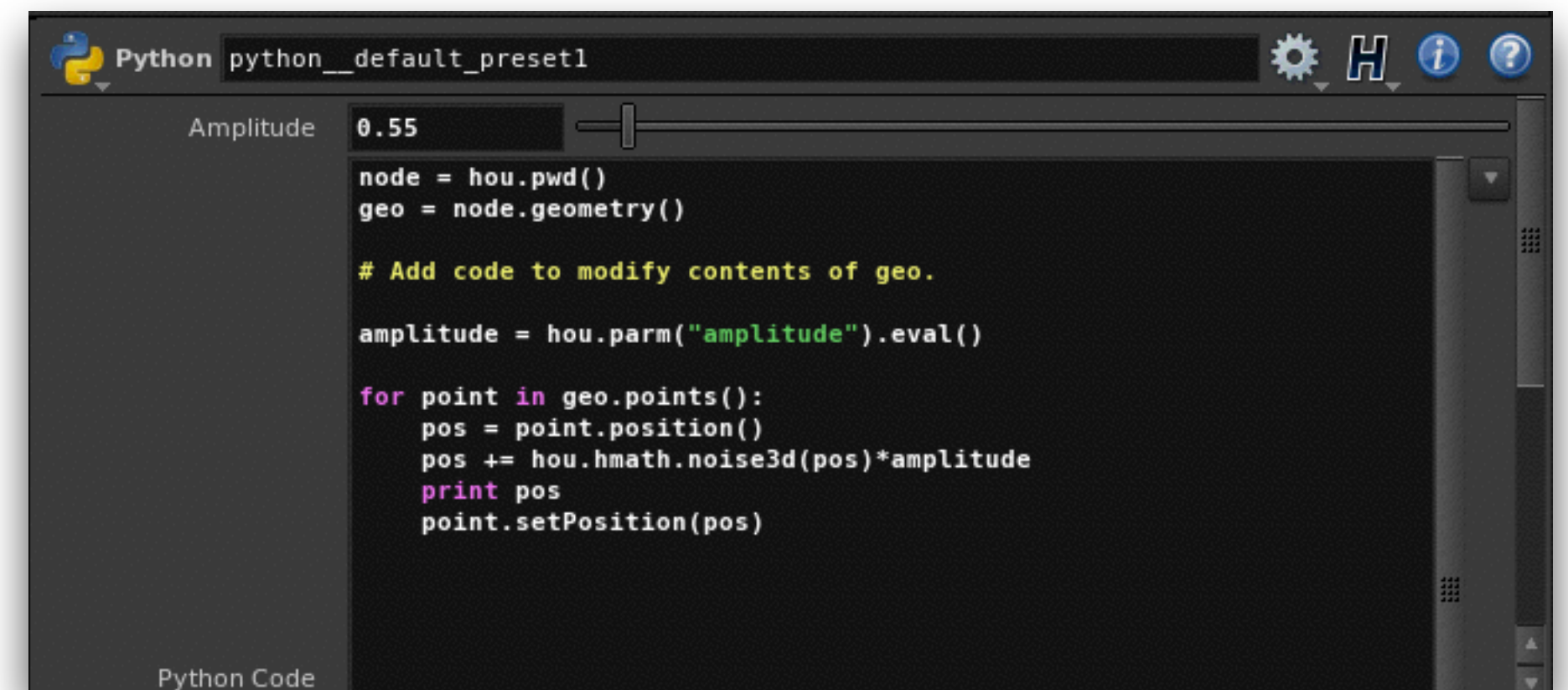
# Step 03 - Adding Noise

Note: The rest of the steps for this example we will be working on creating the equivalent of a Mountain SOP

## Adding Noise to the Position

We will start by modifying the code to update the position by adding 3D Noise and multiplying it by the spare parameter “amplitude”

```
pos += hou.hmath.noise3d(pos) * amplitude
```





## hou.hMath module

Houdini and 3D related math functions. You can still use Python's Math module.

`noise3d(self, pos) → hou.Vector3`

Given a sequence of 1 to 4 floats representing a position in N-dimensional space, return a `hou.Vector3` object representing the vector noise at the given position.

This function matches the output of the noise function from VEX.

# hou.hMath module (cont.)

## Key functions

`buildTranslate(values)` → `hou.Matrix4`

`buildRotate(values, order=xyz)` → `hou.Matrix4`

`buildScale(values)` → `hou.Matrix4`

`buildShear(values)` → `hou.Matrix4`

`buildTransform(values_dict, transform_order="srt", rotate_order="xyz")` → `hou.Matrix4`

`degToRad(degrees)` → `float`

`radToDeg(radians)` → `double`

`clamp(value, min, max)` → `float`

`wrap(value, min, max)`

`sign(value)` → `int`

`smooth(value, min, max)` → `float`

`fit(value, old_min, old_max, new_min, new_max)` → `float`

`fit01(value, new_min, new_max)` → `float`

`fit10(value, new_min, new_max)` → `float`

`fit11(value, new_min, new_max)` → `float`

`rand(seed)` → `float`

`noiseld(self, pos)` → `float`

`noise3d(self, pos)` → `hou.Vector3`

Key Point - All the Math functionality you are used to in HScript is here in `hou.hMath`

For more generic math functions use Python's math module.

# Step 04 - Adding an Interrupt

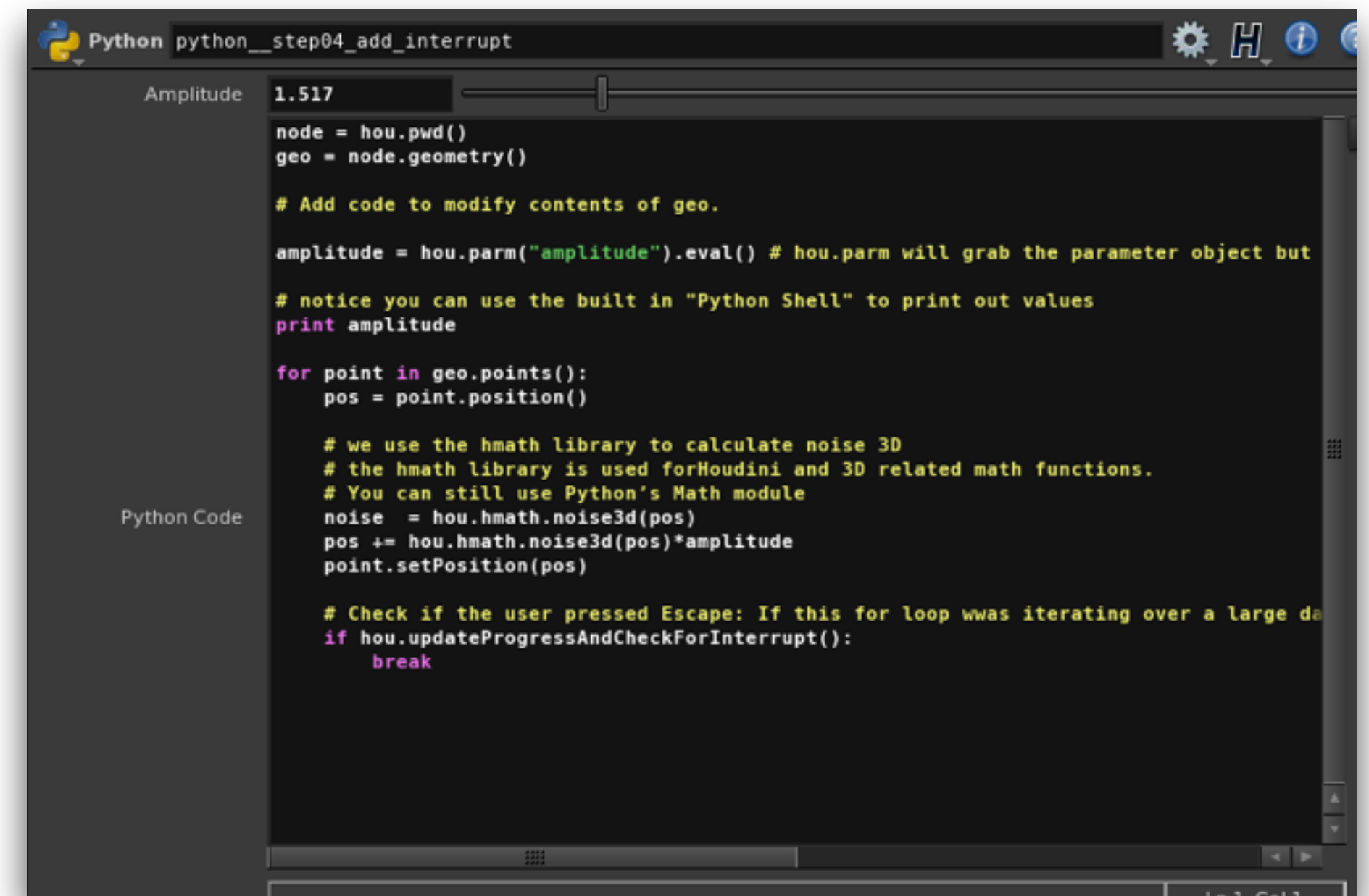
If you are iterating through a large data set Houdini may become sluggish or seem “hung.”

It is a good idea to add logic to be able break out of the for loop.

The Houdini convention to stop cooking is to use the “ESC” key

In the loop add:

```
if hou.updateProgressAndCheckForInterrupt() :  
    break
```



# hou.updateProgressAndCheckForInterrupt



Deprecated



## hou.updateProgressAndCheckForInterrupt function

*Deprecated: Use InterruptableOperation.*

*Check if the user pressed Escape to interrupt the current operation, and optionally update the completion progress.*

### Usage

```
updateProgressAndCheckForInterrupt(percentage=-1) → bool
```

#### Warning

This is deprecated. Use [hou.InterruptableOperation](#) instead.

Return `True` if the user pressed Escape and `False` otherwise.

**percentage** An integer value containing the percentage complete to display in the taskbar. If the percentage is -1, the taskbar will not display the completion percentage.

You would typically call this function from inside a Python SOP, to let the user of the SOP press escape to interrupt a long cook.

See [Define a new SOP type using Python](#) for an example.



# Interrupt - The New Method

```
with hou.InterruptableOperation("Performing Tasks", open_interrupt_dialog=True) as operation:
    count = 0;
    num_points = len(geo.points())
    for point in geo.points():
        pos = point.position()
        count += 1
        # we use the hmath library to calculate noise 3D
        # the hmath library is used for Houdini and 3D related math functions.
        # You can still use Python's Math module
        noise = hou.hmath.noise3d(pos)
        pos += hou.hmath.noise3d(pos)*amplitude
        point.setPosition(pos)
        # Update operation progress.
        percent = float(count) / float(num_points)
        operation.updateProgress(percent)
```

Nest the for...loop inside a "with" statement  
Need to add variable "count" since "point" in  
for loop is a hou.Point object which can not be  
cast to float  
num\_points - geo.points is a list of Point  
Objects so we need a float to calculate  
percentage

# Step 05 - Adding Normals

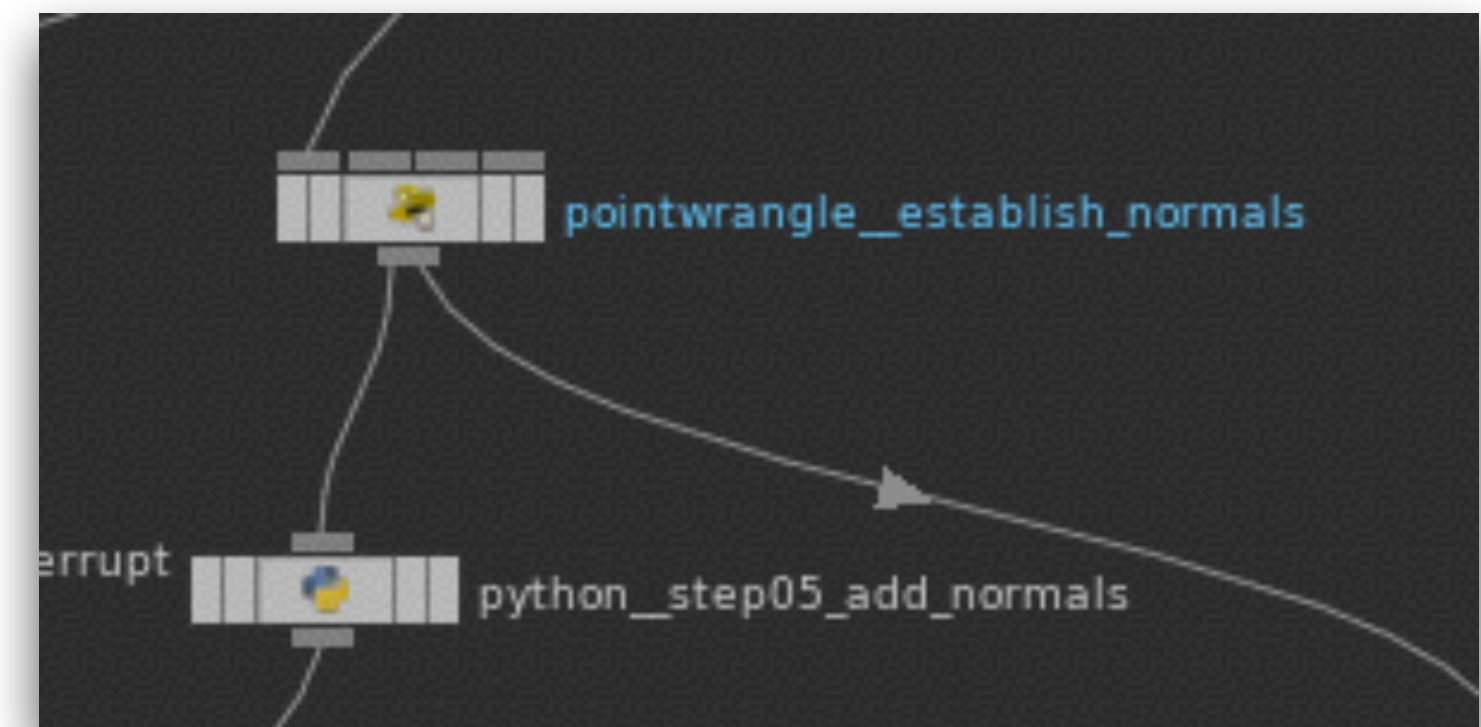
Not everything has to be done in the Python SOP

Drop down a Point Wrangle

```
@N = @N; // establish normals
```

Modify Code to read in the point attribute “N”

```
normal_attr = geo.findPointAttrib("N")
```



```
for point in geo.points():  
    pos = point.position()  
  
    # Fetch the point Normal on input geometry  
    normal_attr = geo.findPointAttrib("N")  
    # it is good form to convert 3 floats to proper Vector3() data types
```

# Some hou.Geometry Functions

**findPointAttrib(*self*, name)** → **hou.Attrib** or **None**

Look up a point attribute by name. Returns the corresponding **hou.Attrib** object, or None if no attribute exists with that name.

Note that the point position attribute is named **P** and is 3 floats in size. Also, the point weight attribute is named **Pw** and is 1 float in size.

These attributes always exist in HOM, even though they are not listed by Houdini's UI.

**findPrimAttrib(*self*, name)** → **hou.Attrib** or **None**

Look up a primitive attribute by name. Returns the corresponding **hou.Attrib** object, or None if no attribute exists with that name.

**deletePoints(*self*, points)**

Delete a sequence of points. You would typically call this method from the code of a Python-defined SOP.

**findPointGroup(*self*, name)** → **hou.PointGroup** or **None**

Return the point group with the given name, or None if no such group exists.

**pointGroups(*self*)** → **tuple** of **hou.PointGroup**

Return a tuple of all the point groups in the geometry.

The following function returns the names of all the groups in the geometry:

```
def pointGroupNames(geometry):  
    return [group.name() for group in geometry.pointGroups()]
```

**createPointGroup(*self*, name, is\_ordered=False)** → **hou.PointGroup**

Create a new point group in this geometry.

## Adding Normals (cont.)

```
# Fetch the point Normal on input geometry
    normal_attr = geo.findPointAttrib("N")
    # it is good form to convert strings to proper Vector3() data types
    v = hou.Vector3(point.attribValue(normal_attr))
    v = v.normalized()
```





## hou.Vector3 class

A sequence of 3 floating point values, with associated mathematical operations.

A Vector3 might be used to represent a position in 3D space, or a 3D direction with a length.

See also [hou.Vector2](#) and [hou.Vector4](#).

**length(*self*)** → float

Interpret this vector as a direction vector and return its length. The result is the same as `math.sqrt(self[0]**2 + self[1]**2 + self[2]**2)`.

**lengthSquared(*self*)** → float

Interpret this vector as a direction vector and return the square of its length. The result is the same as `self[0]**2 + self[1]**2 + self[2]**2`.

**normalized(*self*)** → [hou.Vector3](#)

Interpret this vector as a direction and return a vector with the same direction but with a length of 1. If the vector's length is 0 (or close to it), the result is the original vector.

For vectors with non-zero lengths, this method is equivalent to `self * (1.0/self.length())`.

**distanceTo(*self*, vector3)** → float

Interpret this vector and the argument as 3D positions, and return the distance between them. The return value is equivalent to `(self - vector3).length()`.

**dot(*self*, vector3)** → float

Return the dot product between this vector and the one in the parameter. This value is equal to `self[0]*vector3[0] + self[1]*vector3[1] + self[2]*vector3[2]`, which is also equal to `self.length() * vector3.length() * math.cos(hou.hmath.degToRad(self.angleTo(vector3)))`

See [Wikipedia's dot product page](#).

**cross(*self*, vector3)** → [hou.Vector3](#)

Return the cross product of this vector with another vector. The return value is a vector that is perpendicular to both vectors, pointing in the direction defined by the right-hand rule, with length `self.length() * vector3.length() * math.sin(hou.hmath.degToRad(self.angleTo(vector3)))`.

See [Wikipedia's cross product page](#).

**angleTo(*self*, vector3)** → float

Interprets this Vector3 and the parameter as directions and returns the angle (in degrees) formed between the two vectors when you place the origins at the same location

# Code So far...

```
node = hou.pwd()
geo = node.geometry()

# Add code to modify contents of geo.

# hou.parm will grab the parameter object but will not evaluate the value.
# You need the eval() function to grab the value
amplitude = hou.parm("amplitude").eval()

# notice you can use the built in "Python Shell" to print out values
# print amplitude

for point in geo.points():
    pos = point.position()

    # Fetch the point Normal on input geometry
    normal_attr = geo.findPointAttrib("N")
    # it is good form to convert 3 floats to proper Vector3() data types

    v = hou.Vector3(point.attribValue(normal_attr))
    v = v.normalized()
    print v
    # we use the hmath library to calculate noise 3D
    # the hmath library is used for Houdini and 3D related math functions.
    # You can still use Python's Math module
    noise = hou.hmath.noise3d(pos)
    pos += v*noise*amplitude
    point.setPosition(pos)

# Check if the user pressed Escape: If this for loop was iterating
# over a large data set the artist might want to escape out of it
if hou.updateProgressAndCheckForInterrupt():
    break
```

# Step06 - Calculating the Magnitude of the Vector

```
for point in geo.points():  
    pos = point.position()  
    # Calculate noise using hou.hmath.noise3d() method  
    noise = hou.hmath.noise3d(pos)  
    # Calculate magnitude of vector  
    noise_magnitude = noise.length()  
    noise_magnitude = hou.hmath.fit01(noise_magnitude, -1, 1)
```

# hou Fit Functions

Fit01

**fit(value, old\_min, old\_max, new\_min, new\_max) → float**

Returns a number between `new_min` and `new_max` that is relative to the `value` between the range `old_min` and `old_max`. If the value is outside the `old_min` to `old_max` range, it will be clamped to the new range.

```
>>> hou.hmath.fit(3, 1, 4, 5, 20)  
15.0
```

**fit01(value, new\_min, new\_max) → float**

Returns a number between `new_min` and `new_max` that is relative to the `value` between the range 0 and 1. If the value is outside the 0 to 1 range, it will be clamped to the new range.

This function is a shortcut for `hou.hmath.fit(value, 0.0, 1.0, new_min, new_max)`.

**fit10(value, new\_min, new\_max) → float**

Returns a number between `new_min` and `new_max` that is relative to the `value` between the range 1 to 0. If the value is outside the 1 to 0 range, it will be clamped to the new range.

This function is a shortcut for `hou.hmath.fit(value, 1.0, 0.0, new_min, new_max)`.

**fit11(value, new\_min, new\_max) → float**

Returns a number between `new_min` and `new_max` that is relative to the `value` between the range -1 to 1. If the value is outside the -1 to 1 range, it will be clamped to the new range.

This function is a shortcut for `hou.hmath.fit(value, -1.0, 1.0, new_min, new_max)`.



# Step 06 - Putting it All Together

```
node = hou.pwd()
geo = node.geometry()
# Add code to modify contents of geo.

# Add a new point attribute called noise_weight to the geometry
noise_weight = geo.addAttrib(hou.attribType.Point, "noise_weight", 0)

# hou.parm will grab the parameter object but will not evaluate the value.
# You need the eval() function to grab the value
amplitude = hou.parm("amplitude").eval()

for point in geo.points():
    pos = point.position()
    # Calculate noise using hou.hmath.noise3d() method
    noise = hou.hmath.noise3d(pos)
    # Calculate magnitude of vector
    noise_magnitude = noise.length()
    noise_magnitude = hou.hmath.fit01(noise_magnitude, -1, 1)
    # Fetch the point Normal on input geometry
    normal_attr = geo.findPointAttrib("N")
    # convert 3 floats to proper Vector3() data types
    v = hou.Vector3(point.attribValue(normal_attr))
    v = v.normalized()
    pos += v * noise_magnitude * amplitude
    point.setPosition(pos)
    print amplitude
    # add noise_magnitude to the geometry attribute noise_weight
    point.setAttribValue(noise_weight, 1)

# Check if the user pressed Escape:
if hou.updateProgressAndCheckForInterrupt():
    break
```

Position is updated to the normalized normal  
\* noise length \* user defined amplitude

# point.attribValue



## hou.Point class

*Each Point object resides inside a Geometry object and stores a 3D position. Points may be shared between primitives (such as polygons), and the set of points and primitives describes a 3D shape.*

The set of points may also store arbitrary data in the form of attributes, and each point instance stores a unique attribute value.

**attribValue(self, name\_or\_attr)** → **int , float , str or tuple**

Return value stored in this point for a particular attribute. The attribute may be specified by name or by [hou.Attrib](#) object.

Looking up an attribute value using a [hou.Attrib](#) object is slightly faster than looking it up by name. When looking up attribute values inside a loop, look up the [hou.Attrib](#) object outside the loop, and pass it into this method.

Note that the point position attribute is named **P** and is 4 floats in size. This attribute always exists.

When looking up the attribute values of all points, it is faster to call [hou.Geometry.pointFloatAttribValues](#) or [hou.Geometry.pointFloatAttribValuesAsString](#) than to call this method for each point in the geometry.

Raises [hou.OperationFailed](#) if no attribute exists with this name.



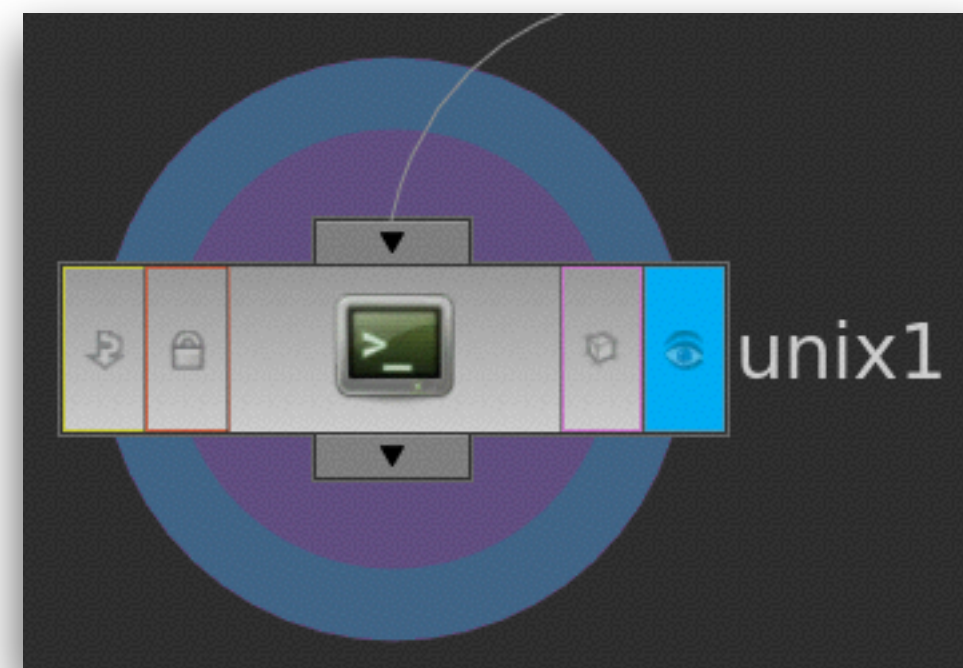
**Unix Node**

# Unix Node

Processes geometry using an external program.

This operator lets you write an external program/script that reads in .geo or .bgeo formatted data|/io/geo (Houdini automatically detects the format), manipulates the data, and writes out modified .geo or .bgeo (set using the Output format parameter).

Not used that often anymore  
Good for sending geometry data to Houdini stand alone programs and getting data back



Despite its name, the unix command works on non-UNIX operating systems.



## **In the Old Days...**

**Perl scripts were used extensively with the Unix SOP to convert geometry into other formats or allow Command Line Tools to process the data**

**We will be more modern....**

**We will create a Python Script to be executed within a Unix SOP**

# Unix Node

Unix Command

unix1

Take List

Parameter Spreadsheet

obj

geol

Unix unix1

gconvert -l 0.5 /Library/Frameworks/Houdini.framework/Versions/13.0.405/Resourc

Output Format

Ascii

Force Recook

Ascii or Bin

Force to Execute Command

Where does convert come from?

Answer: Next Slide

# Side Note - \$HFS/bin

```
ari — Houdini Shell — bash — 120x36
Last login: Tue May  6 11:58:28 on ttys000
Oriel:~ ari$ cd /Library/Frameworks/Houdini.framework/Versions/13.0.405/Resources;source houdini_setup; cd -
The Houdini 13.0.405 environment has been initialized.
/Users/ari
Oriel:~ ari$ ls $HFS/bin
abccconvert  gconvert      hescape      i3diso      mcp
abcecho      gdx           hexpand      iautocrop  mids
abcinfo      geodiff       hexper       ic          minfo
aclip        geps         hhalo-ng     icineon    mplay
acombine     gfont        hidx         icomposite mview
aconvert     gfont.app    hkey         iconvert   orbolt_url
ainfo        gicon        hmaster-ng   icp        pcfiler
aplay        giges        hotl         idiff      pcmerge
arecord      ginfo        hotlview     iflip      proto_install
areverse     glightwave   houdini      ihot       rscript
builduicode  gpdb         houdinifx    iinfo      sdl2otl.py
chchan       gplay        hremove      ilut       sesitag
chcp         gp           hrender      ilutcomp   slo2otl.py
chinfo       gptex        hrmanpipe    ilutinfo   spiff
claudio      greduce      hrmanshader  ilutiridas spy
clchan       gstat        hscript      imdisplay  uiparse
clchn        gstl         hserver      iprint     vcc
clcp         gwavefront   hview        iquantize  vcc11
cldiff       hbatch       hwatermark   isixpack   vcc12
clinfo       hbrickmap    hython       itilestitch vcrypt
clphone      hcollapse    hython2.6    izg        vexexec
dsmconvert   hcommand     hython2.7    izmatte    vmantra
dsmmerge     hcompile     i3dconvert   mantra     wren
dsparse      hconfig      i3ddsmgen    mcacclaim  zmore
easy         hcpio        i3dgen       mcbiovision
gabc         hcustom      i3dinfo      mcmotanal
Oriel:~ ari$
```

Houdini stand alone commands are located at \$HFS/bin

You can call some of these commands from the Unix SOP

On your own try:

**gconvert**

**gdx**

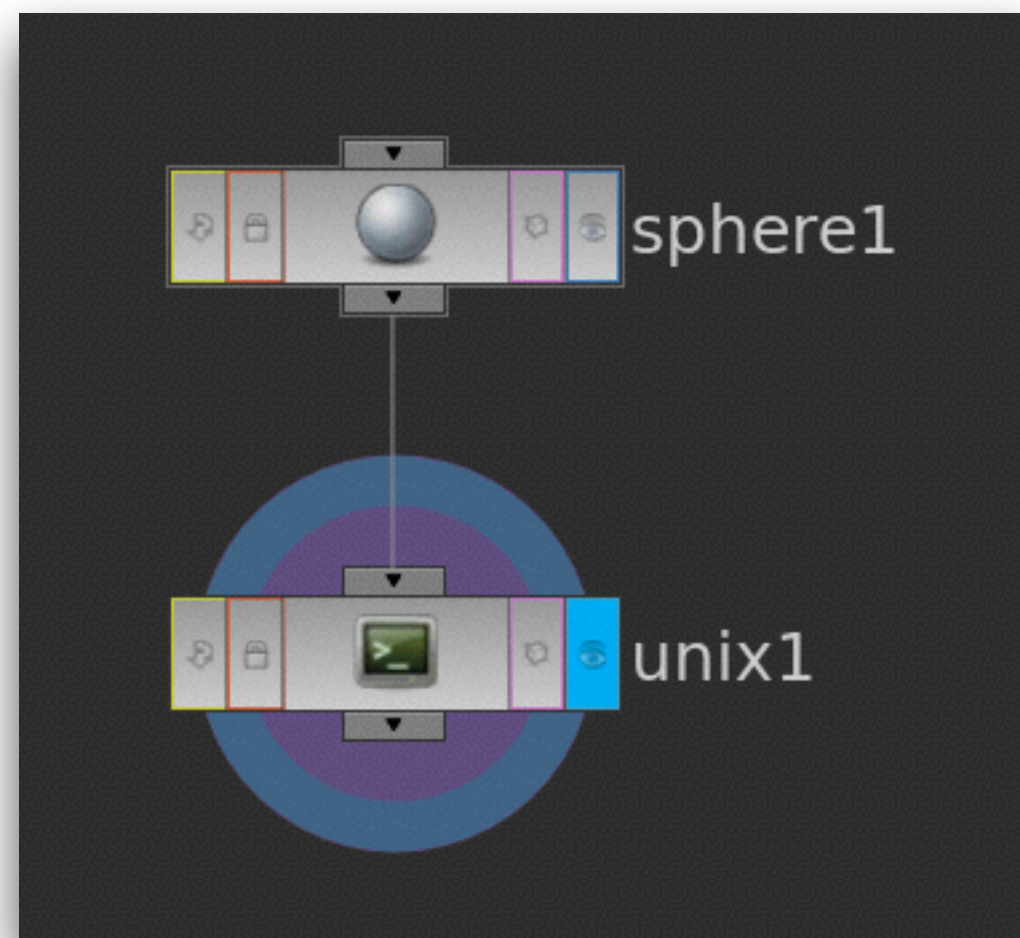
**gwavefront**



# What Will the Python Script Do?

Take any geometry and save out its point positions to a spreadsheet so the point positions can be analyzed by another program

We will write a Python Shell command to take the geometry from the node above the Unix SOP and write out a CSV file



	A	B	C	D	E
1	0	0	-1	1	
2	0	0.360728621483	-0.93267083168	1	
3	0.343073219061	0.111471429467	-0.93267083168	1	
4	0	0.672883570194	-0.739748418331	1	
5	0.35682207346	0.491123735905	-0.79465430975	1	
6	0.639950215816	0.207932755351	-0.739748477936	1	
7	0	0.894427359104	-0.44721314311	1	
8	0.343073278666	0.78435498476	-0.51680546999	1	
9	0.639950215816	0.56866145134	-0.516805529594	1	
10	0.850650846958	0.276393681765	-0.447213202715	1	
11	0.212030917406	-0.291835516691	-0.93267083168	1	
12	0.577350318432	-0.187592312694	-0.794654369354	1	
13	0.395510971546	-0.544374167919	-0.739748477936	1	
14	0.851981341839	-0.0839028730989	-0.516805648804	1	
15	0.738584518433	-0.43290284276	-0.516805708408	1	
16	0.52573120594	-0.723606944084	-0.44721326232	1	
17	-0.212030917406	-0.291835516691	-0.93267083168	1	
18	0	-0.607062101364	-0.794654369354	1	
19	-0.395510971546	-0.544374167919	-0.739748477936	1	



# Telling the Shell We Are Running a Python Script

```
#!/usr/bin/env python
```

If you have several versions of Python installed, `/usr/bin/env` will ensure the interpreter used is the first one on your environment's `$PATH`. The alternative would be to hardcode something like `#!/usr/bin/python` or the like -- that's OK but less flexible.

In Unix, an executable file that's meant to be interpreted must indicate what interpreter to use by having a `#!` at the start of the first line, followed by the interpreter (and any flags it may need); otherwise, I believe the default is `/bin/sh`.

Just to add: this applies when you run it in Unix by making it executable (`chmod +x myscript.py`) and then running it directly: `./myscript.py`, rather than just `python myscript.py`.



Comments from  
[StackOverflow.com](https://stackoverflow.com)

# Importing Libraries

```
import os, sys, json  
  
import csv
```

**OS** - The OS module in Python provides a way of using operating system dependent functionality.

The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.

Comments from

[www.pythonforbeginners.com](http://www.pythonforbeginners.com)

**SIDE EFFECTS  
SOFTWARE**

# OS Functions

Executing a shell command

**os.system()**

Get the users environment

**os.environ()**

#Returns the current working directory.

**os.getcwd()**

Return the real group id of the current process.

**os.getgid()**

Return the current process's user id.

**os.getuid()**

Returns the real process ID of the current process.

**os.getpid()**

Set the current numeric umask and return the previous umask.

**os.umask(mask)**

Return information identifying the current operating system.

**os.uname()**

Change the root directory of the current process to path.

**os.chroot(path)**

Return a list of the entries in the directory given by path.

**os.listdir(path)**

Create a directory named path with numeric mode mode.

**os.mkdir(path)**

Recursive directory creation function.

**os.makedirs(path)**

Remove (delete) the file path.

**os.remove(path)**

Remove directories recursively.

**os.removedirs(path)**

Rename the file or directory src to dst.

**os.rename(src, dst)**

Remove (delete) the directory path.

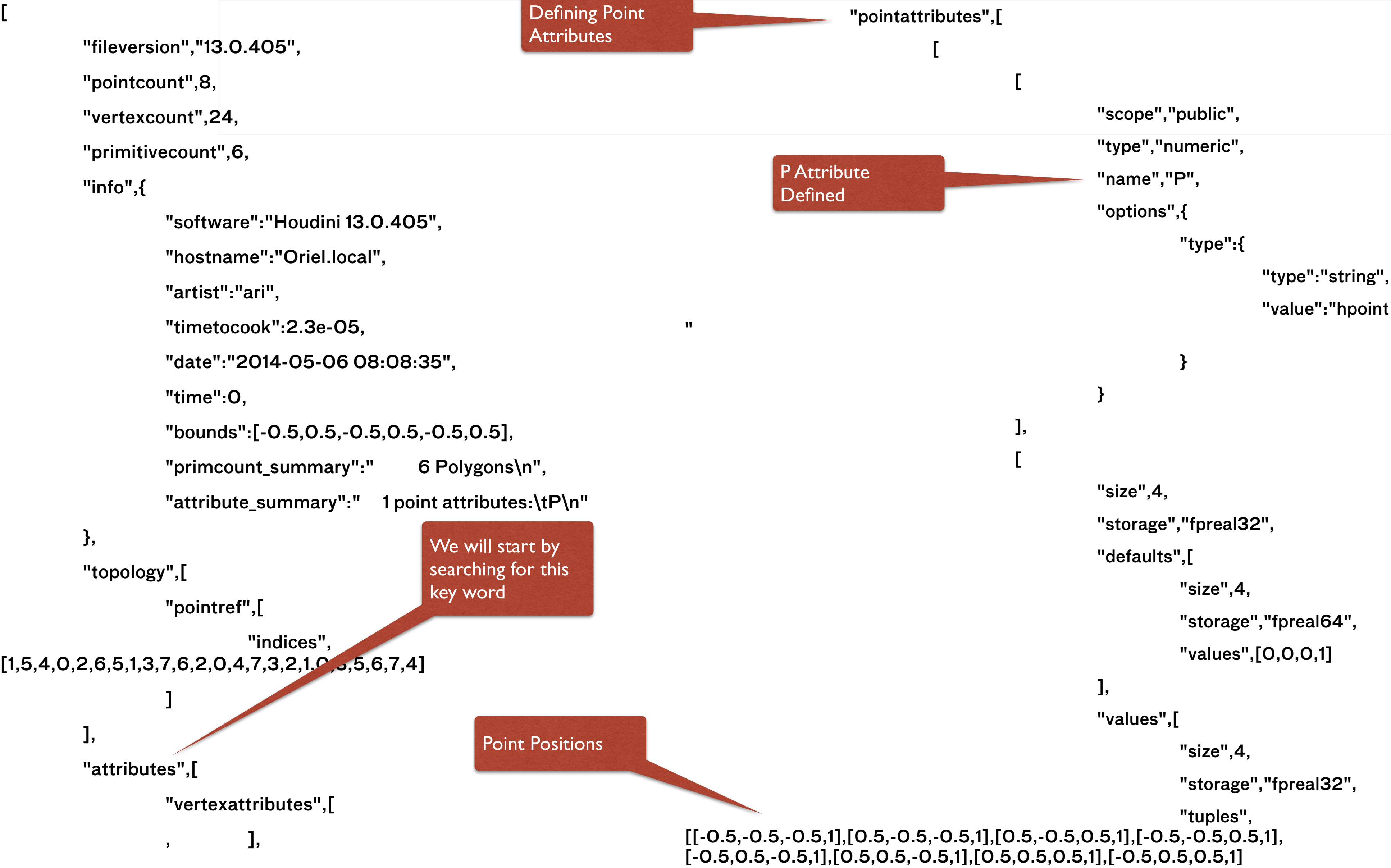
**os.rmdir(path)**

**sys** - This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment.



**JSON** - JavaScript Object Notation is a lightweight data interchange format based on a subset of JavaScript syntax (ECMA-262 3rd edition).

More importantly when you save a geo file in Houdini it is using the JSON format



## CSV - Comma Separated Values.

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. There is no “CSV standard”, so the format is operationally defined by the many applications which read and write it. The lack of a standard means that subtle differences often exist in the data produced and consumed by different applications. These differences can make it annoying to process CSV files from multiple sources. Still, while the delimiters and quoting characters vary, the overall format is similar enough that it is possible to write a single module which can efficiently manipulate such data, hiding the details of reading and writing the data from the programmer.

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

## Code So Far...

```
#!/usr/bin/env python  
import os, sys, json  
import csv
```



# Main Function

```
def main():  
  
    # Read the incoming geometry  
    j = json.load(sys.stdin)  
    # Process the input geometry  
    process(j)  
  
    # Save the geometry out so the Unix SOP can read it back. We could, of  
    # course, save different geometry if we wanted.  
    json.dump(j, sys.stdout)  
  
if __name__ == "__main__":  
    main()
```

# Process Function

```
def process(j):  
    '''  
    Process the JSON by looking for the appropriate items in the JSON arrays  
    First find the "attributes" item, then in that array, we find the  
    "pointattributes". We iterate over the attributes, looking for "P". When  
    we find that, we save out the values to the spreadsheet.  
    '''  
  
    # Find the "attributes" item  
    attributes = findItem(j, "attributes")  
  
    # Find the point attributes in the "attributes" block  
    pointattributes = findItem(attributes, "pointattributes")  
  
    for a in pointattributes:  
        name = findItem(a[0], "name")  
  
        if name == "P":  
            values = findItem(a[1], "values")  
            tuples = findItem(values, "tuples")  
            saveP(tuples)
```

# findItem Function

```
def findItem(list, key):  
    '''  
        The geometry format stores items in a list of key/value pairs.  This  
        function finds an item in the list and returns the value (or None if  
        not found).  
    '''  
    for i in xrange(0, len(list), 2):  
        if list[i] == key:  
            return list[i+1]  
    return None
```

# saveP Function

```
def saveP(values):  
    ''' Save the point positions to a file.  The file can be specified as the  
    first argument to the script. '''  
    ofile = hou.expandString('$HIP/thePoint_P.csv')  
  
    if len(sys.argv) > 1:  
        ofile = sys.argv[1]  
  
    ''' Use the csv module to save the positions to a spreadsheet '''  
    fp = csv.writer(open(ofile, 'w'))  
    for p in values:  
        fp.writerow(p)
```





# Script SOP

Inline HScript & Python Scripts

**SIDE EFFECTS  
SOFTWARE**

## Script SOP Does Not Modify Geometry

## Script in HScript or Python



### Script surface node

*Runs scripts when cooked.*

This operation runs a list of scripts whenever it is cooked. This allows you to tie scripts to be performed at certain points in the cooking process. A Script SOP inside a For Each could be used to run a script for every group on an object, for example.

Unlike the Unix SOP, the script is a Python or Hscript and does not modify the actual geometry.

This SOP passes through its input untouched. Adding a Script SOP does not increase memory usage.

### Parameters

<b>Make Time Dependent</b>	If this flag is turned on, the Script SOP will become timedependent even if its input isn't time dependent. This is a way to make a script that is executed on frame change.
<b>Script</b>	Either a Python or an Hscript script that is run whenever this SOP would be cooked.

# \$HFS/Houdini/Scripts

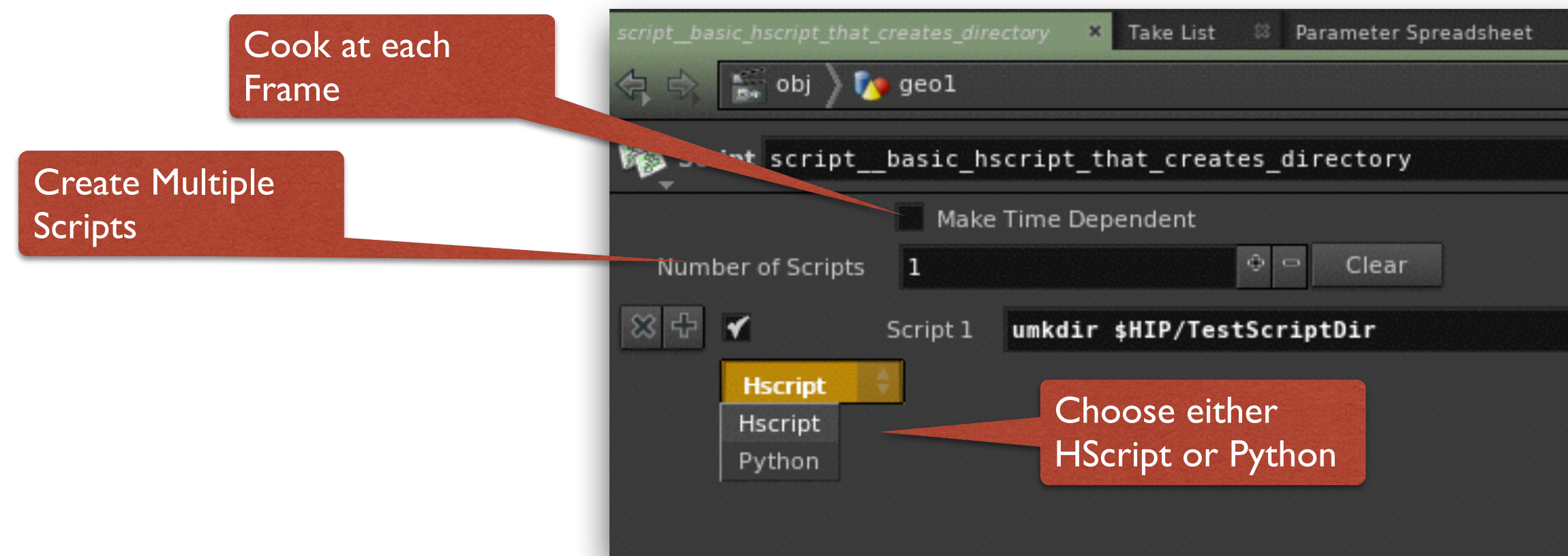
Example Scripts can be found at \$HFS/Houdini/Scripts

Script Ends in - .cmd

```
Oriel:~ ari$ ls $HFS/Houdini/Scripts
123.cmd          hqueue          pop
alfred           ipr             scene
chop             java            shop
cop2            loadHelpcardExample.cmd  sop
data            loadHelpcardOTLExample.cmd  tcl8.0
defaultscene.cmd macro           tk8.0
defaultscene_old.cmd mvexport        traverse.cmd
dop             obj            uniquename.cmd
dophints.cmd   opmacros       vex
fixnodename.cmd out            vop
hescape.cmd    pickandcenter.cmd
Oriel:~ ari$
```



# Step 01 - Script Basics



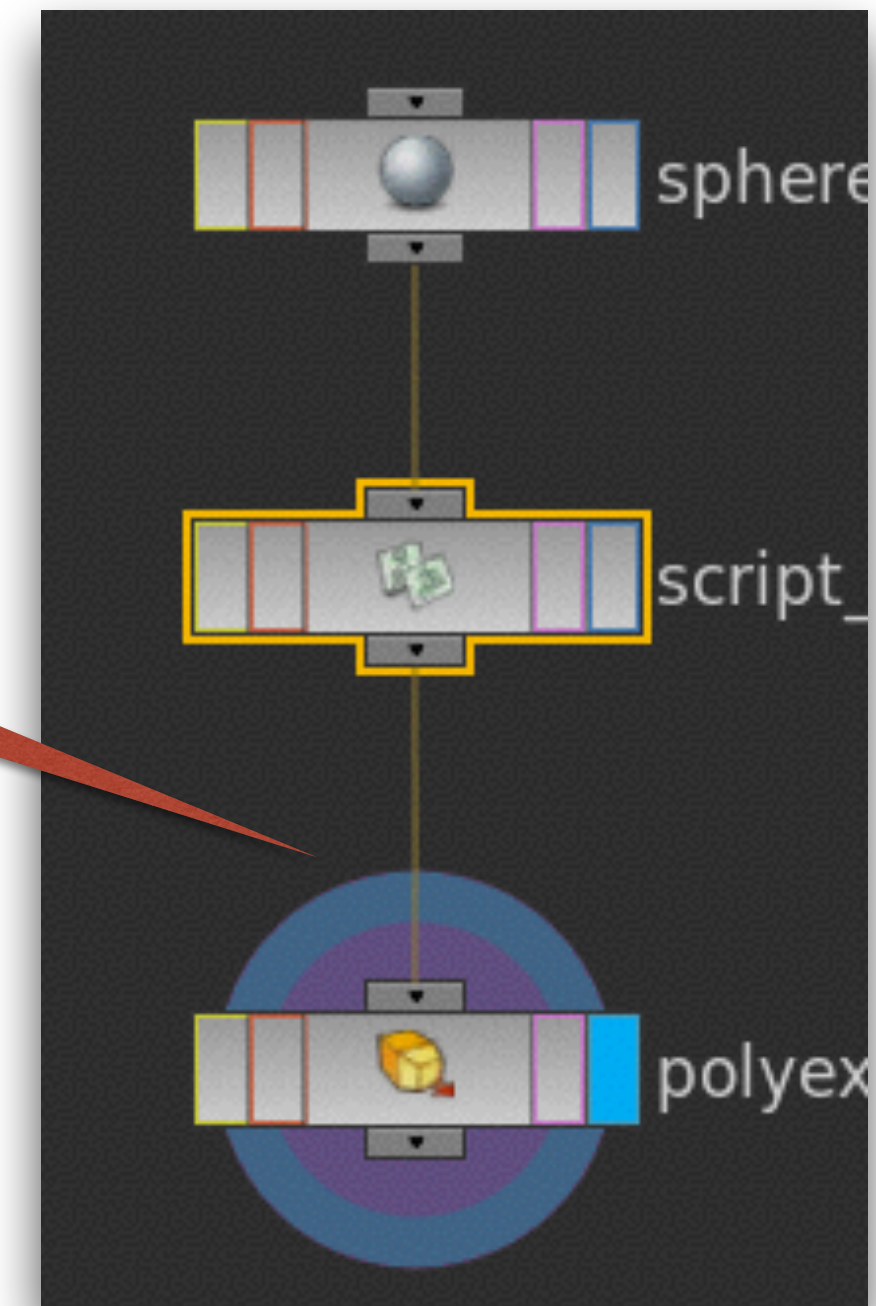
While you can call Unix commands by using the unix command (e.g., `unix mkdir foo`) it is better to use the Houdini equivalent so it will work on all OS platforms

```
umkdir foo
```

You can force the script to cook at each frame by selecting the “Make Time Dependent” Toggle

```
umkdir $HIP/TestScriptDir_$F4
```

Geometry passes through Script node without being altered





## Step 02 - Reading From File

Create a Text document and save it in \$HIP/scripts/SOP

Name it - createFolder.cmd

In the text document type the following script:

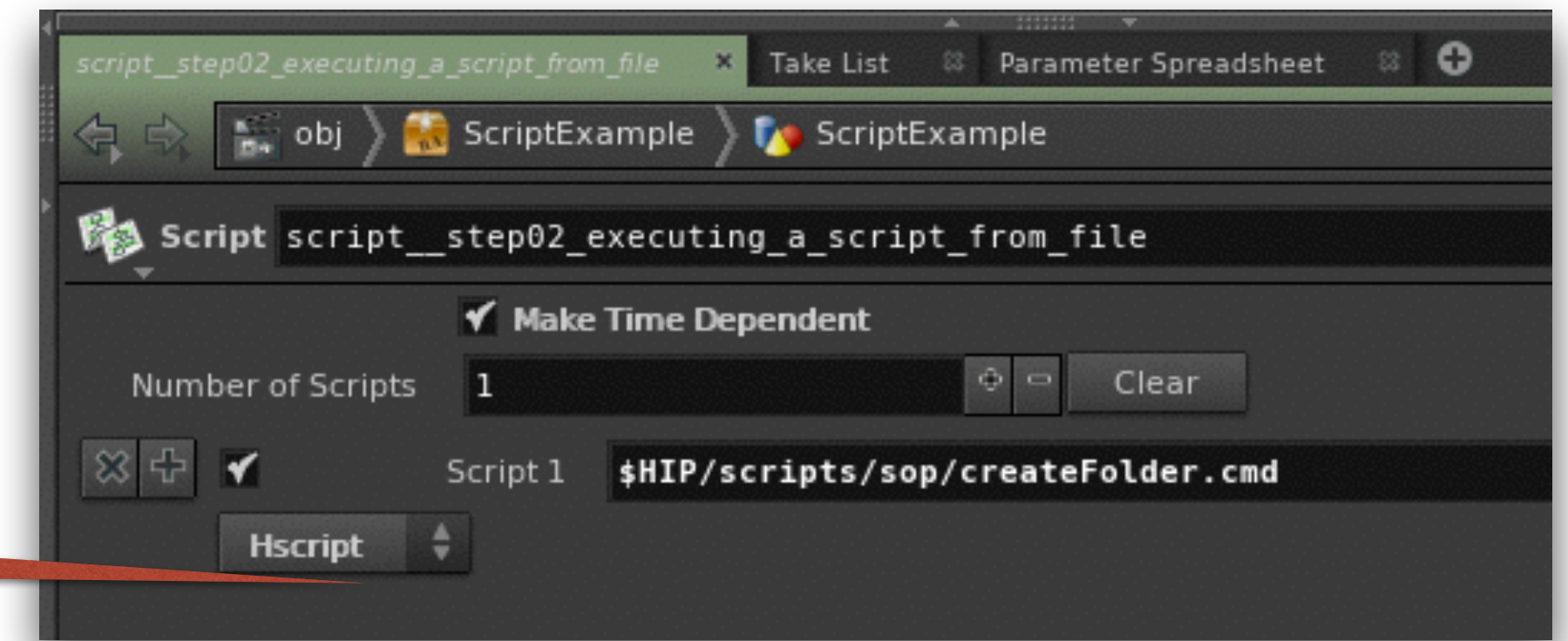
```
umkdir $HIP/TestScriptDir_$F4
```

```
message "I have written a folder TestScriptDir_" `padzero(4,$F4)`
```

In the Script SOP type:

```
$HIP/scripts/sop/createFolder.cmd
```

Make sure it is set  
to HScript



# Step 03 - Placing the Script in the Extras Tab

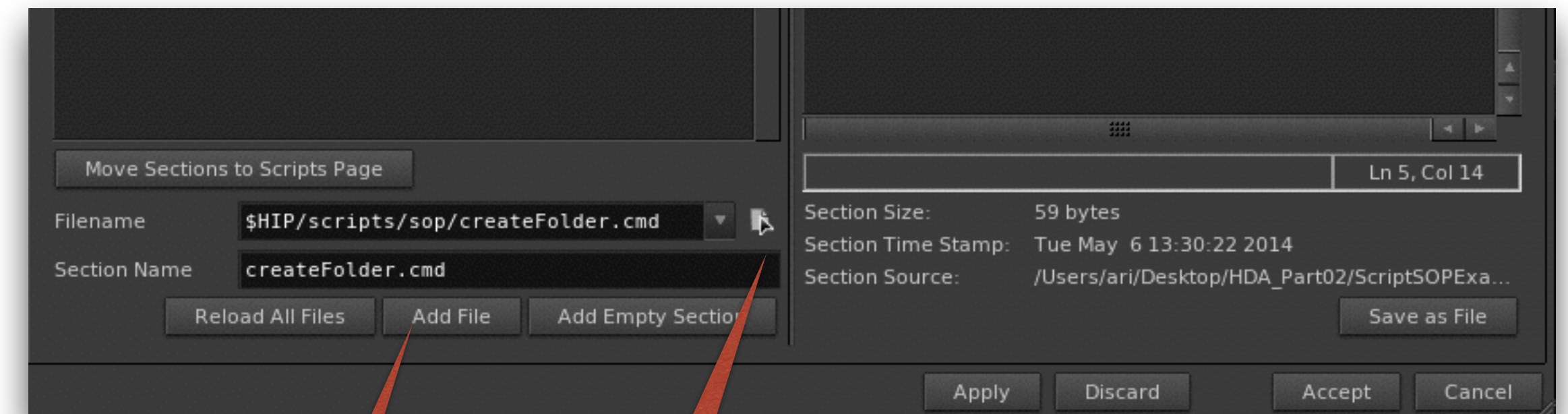
Open the Type Properties for the Script SOP

In the Extra Files Tab load

`$HIP/scripts/sop/createFolder.cmd`

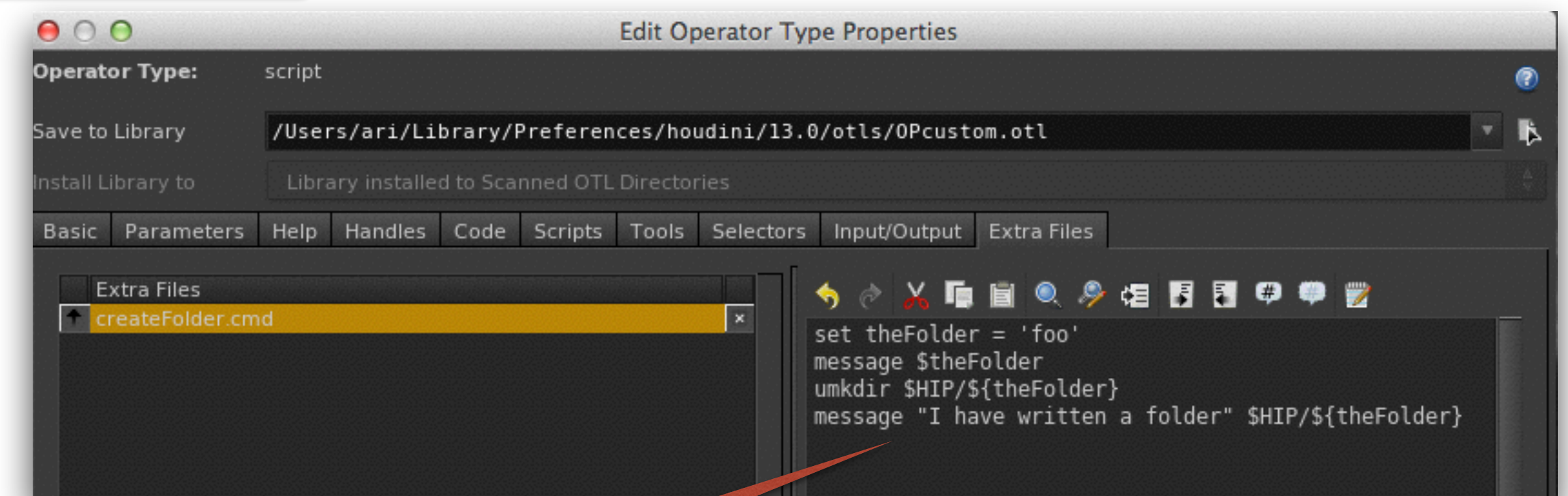
Once loaded. Click on “Add File”

You will see the script in the right side of the panel



Once the file is loaded. Add it.

Click on this and navigate to the file

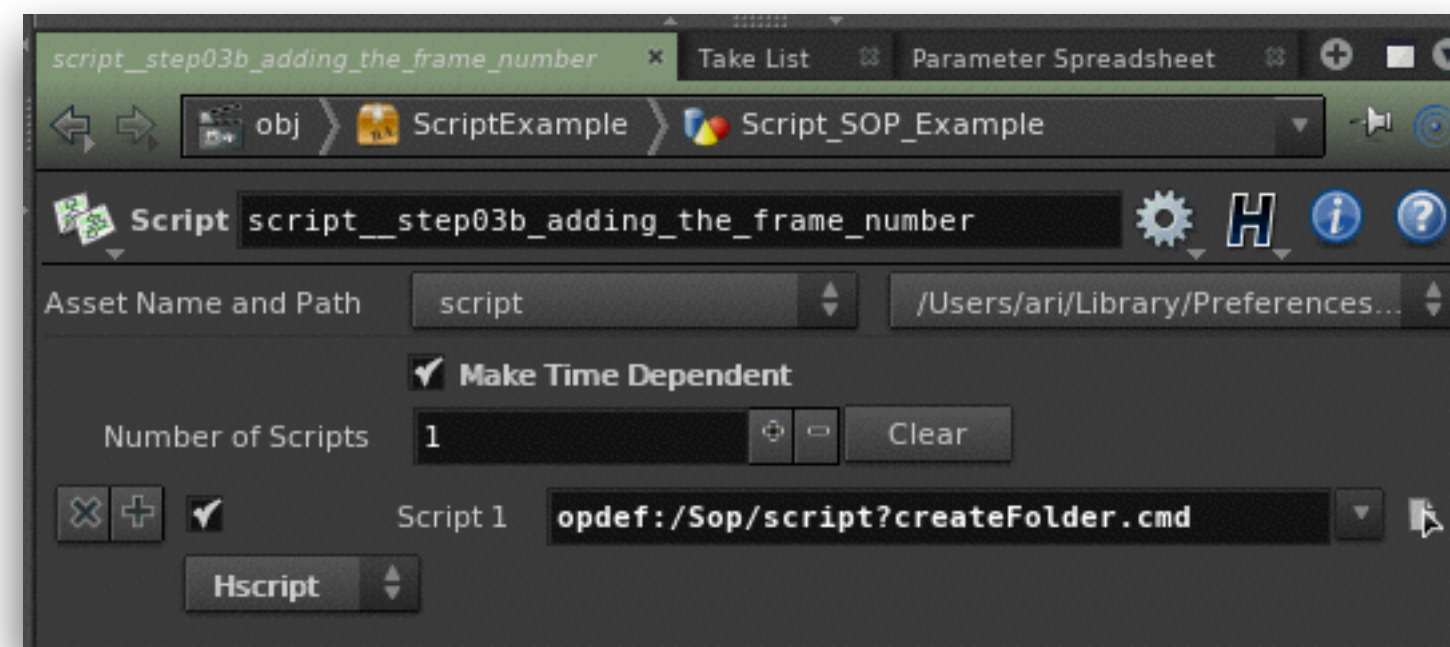


Once “Add File” is clicked you will see the script

# Step 03 - Placing the Script in the Extras Tab (cont.)

In the Script SOP type:

```
opdef:/Sop/script?createFolder.cmd
```



# Step 03b - Adding the Frame Number

Adding the Frame Number and File Name

Go to the Script Tab

Event Handler - Custom Script

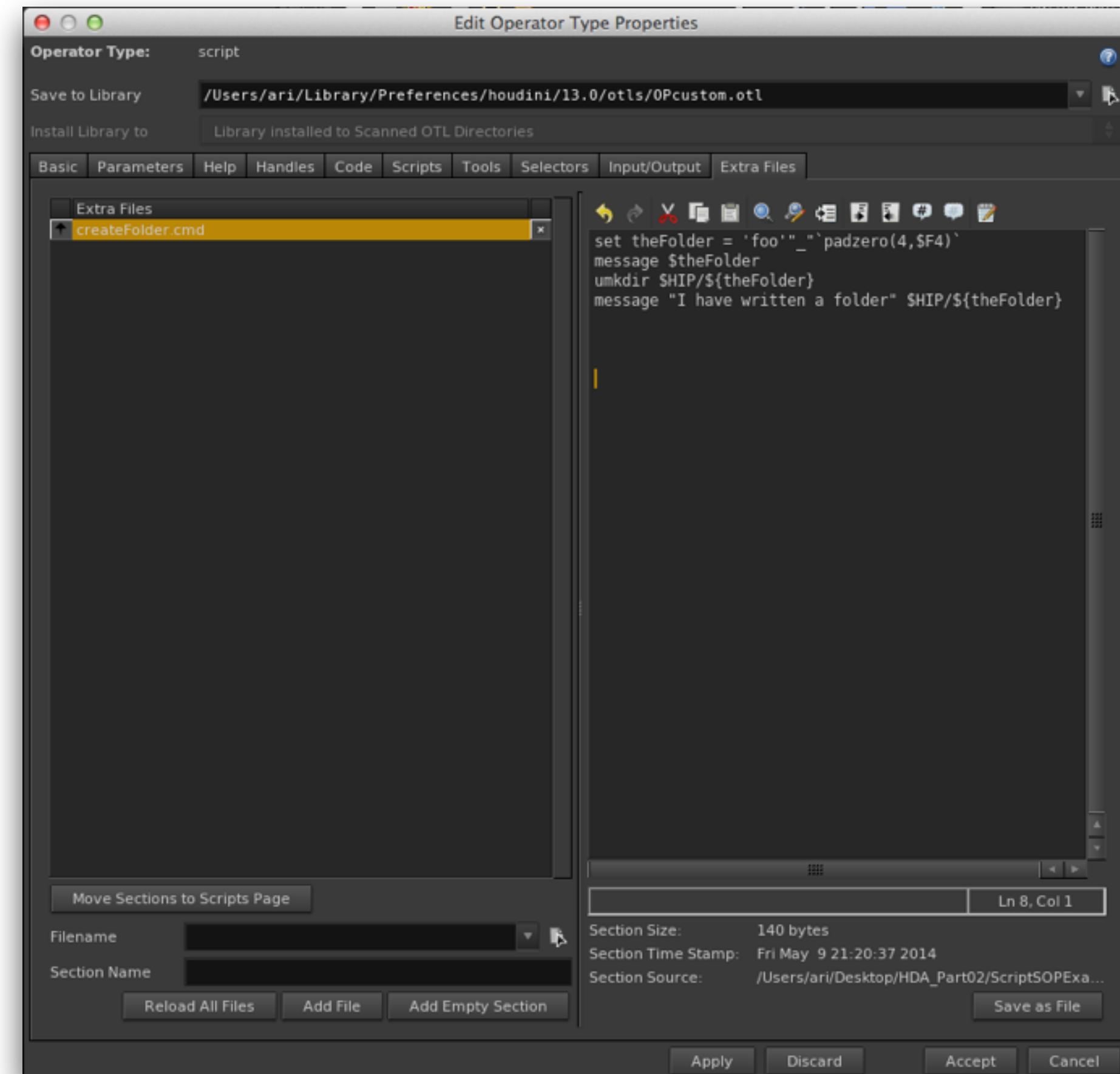
Section Name - CreateFolder

Language - HScript

Click - Add Empty Section

Type the Following Code

```
set theFolder = 'foo' "_" `padzero(4,$F4) `
message $theFolder
umkdir $HIP/${theFolder}
message "I have written a folder" $HIP/${theFolder}
```





# Step 04 - Creating an OTL with Button Script

Start a new .hip file and save it into the same Project folder as the last example

Name - CreateButtonProjectFolder.hip

Drop down a Geometry SOP

Name - Create\_Folder

Dive Inside

Delete the File SOP

Drop down a Null SOP

Go back to the Object Level

Select Create\_Folder and Make it into a Subnet

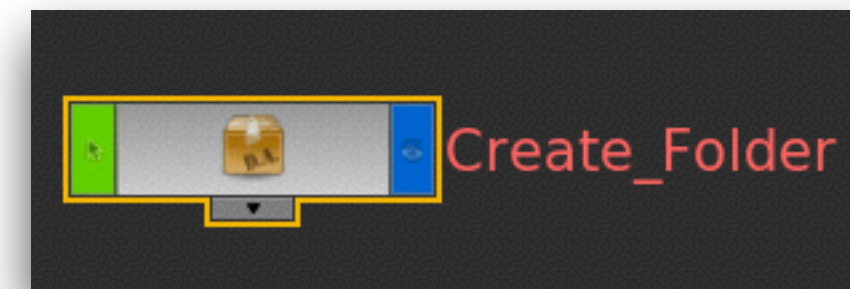
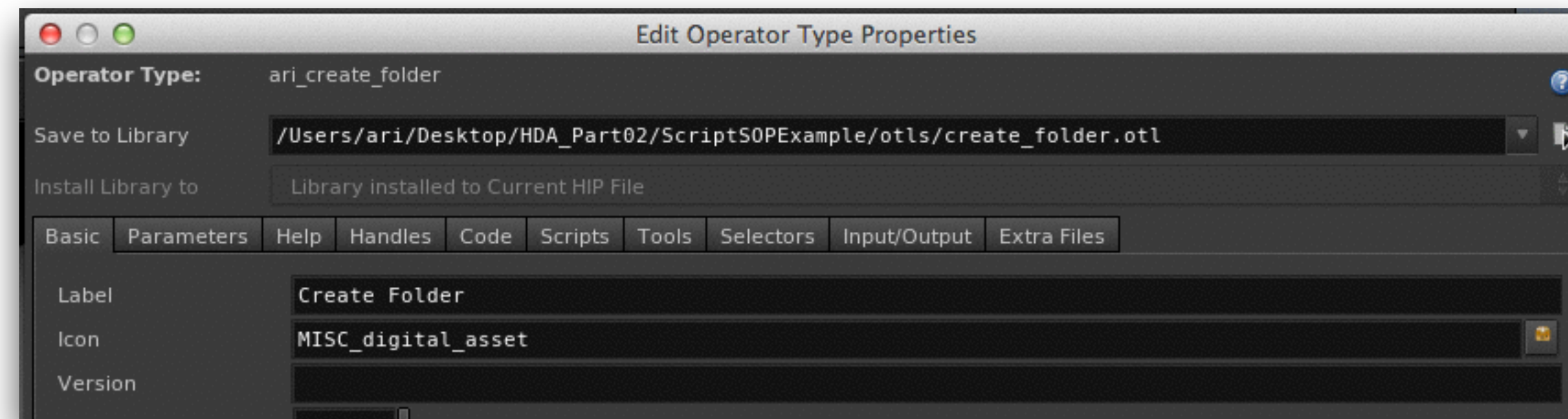
Now convert it into a Digital Asset

Name - CreateFolder

Label - Create Folder

Location - \$HIP/otls

OTL Name - create\_folder.otl



# Step 04 - Creating an OTL with Button Script (cont.)

Got to the Parameters Tab

Create a Button Parameter

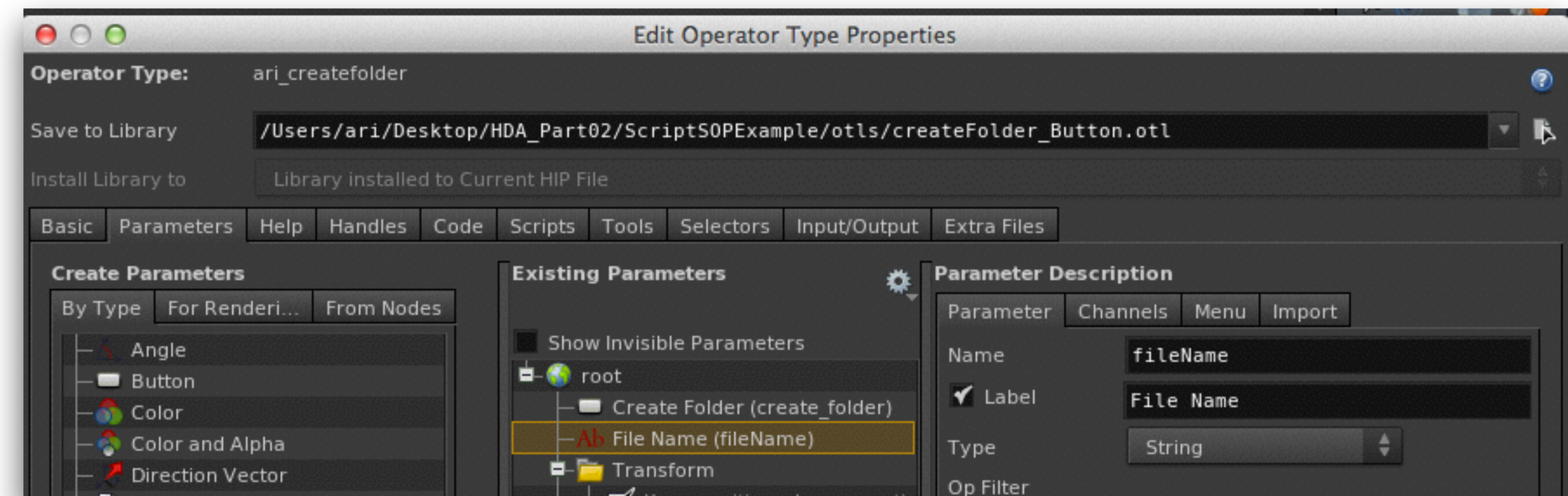
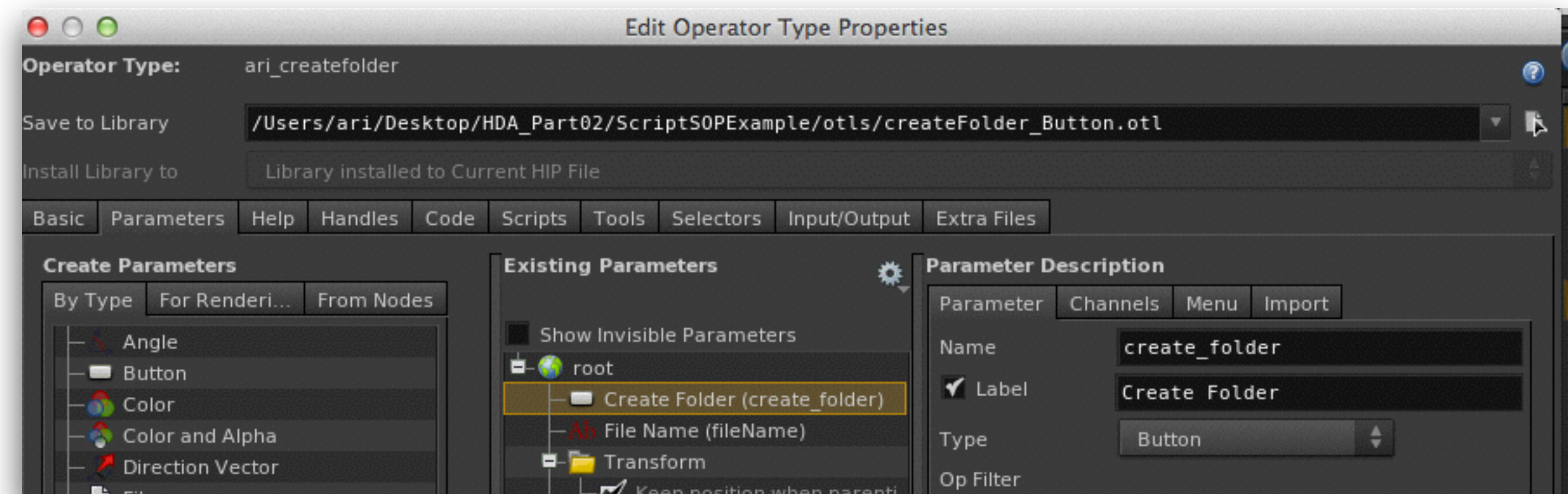
Name - create\_folder

Label - Create Folder

Create a String Parameter

Name - file\_name

Label - File Name



# Step 04 - Creating an OTL with Button Script (cont.)

Go to the Script Tab

Event Handler - Custom Script

Section Name - CreateFolder01

Language - HScript

Click - Add Empty Section

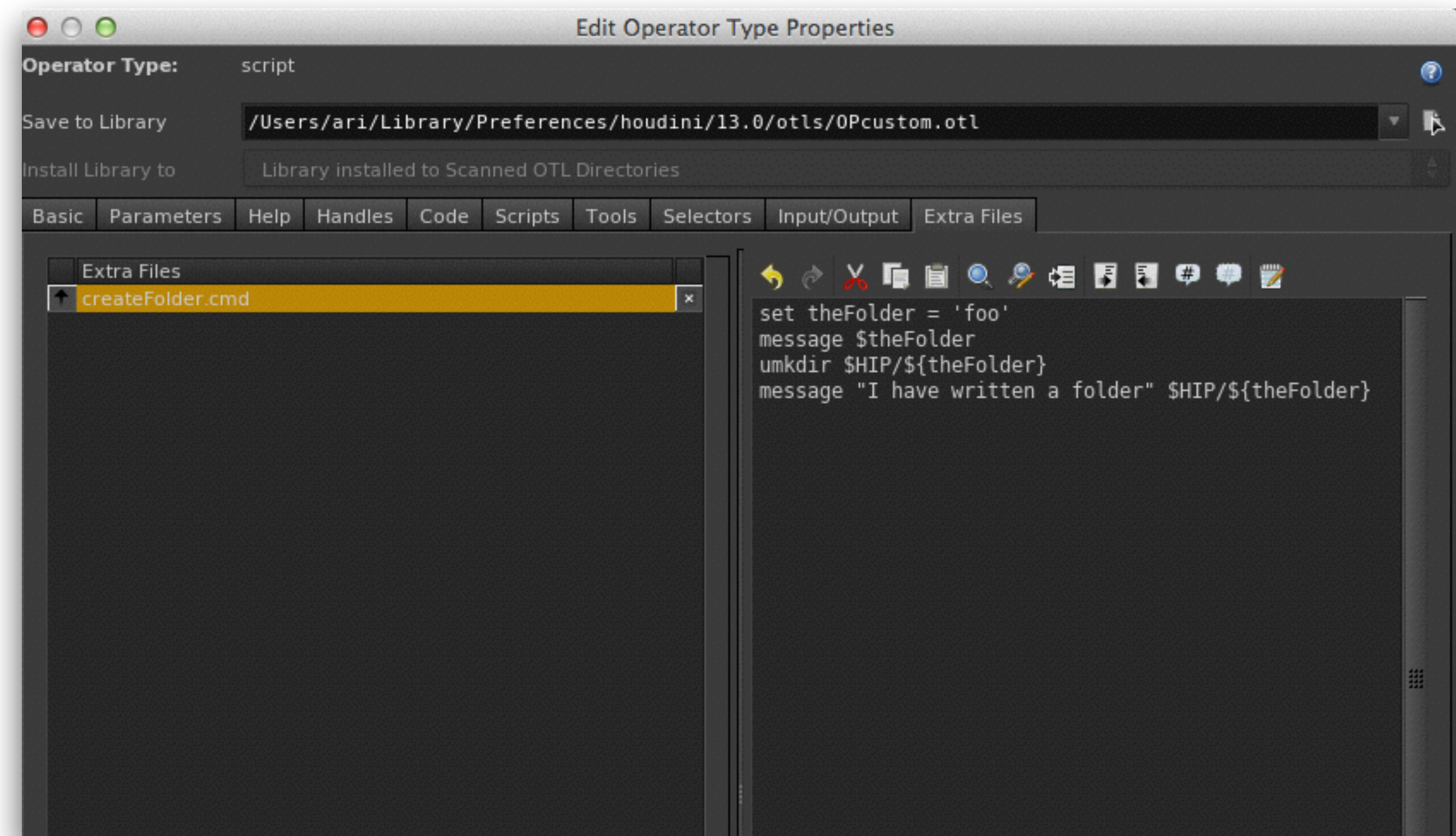
Type the Following Code

```
set theFolder = 'foo'
```

```
message $theFolder
```

```
umkdir $HIP/${theFolder}
```

```
message "I have written a folder" $HIP/${theFolder}
```





# Step 04 - Creating an OTL with Button Script (cont.)

Go back to the Parameters Tab

In the callback script section type:

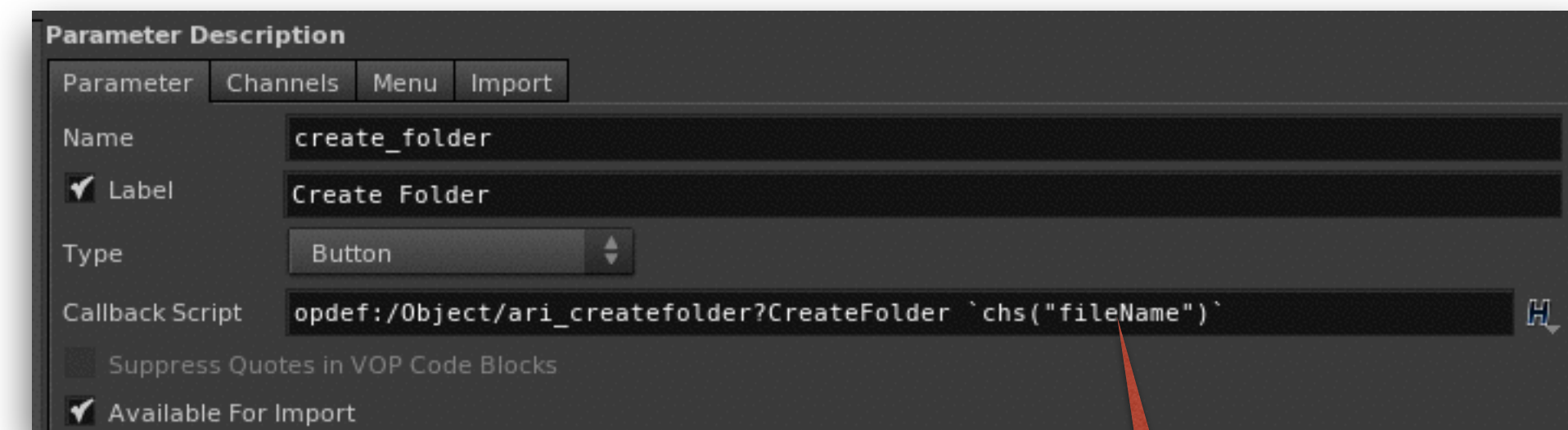
```
opdef:/Object/ari_createfolder?CreateFolder `chs("fileName")`
```

Click Apply

By clicking the “Create Folder” Button you the callback script will be invoked

The callback script calls the HScript we just created

The last part of the command ``chs("fileName")`` will send the string parameter “filename” to the script as arg1

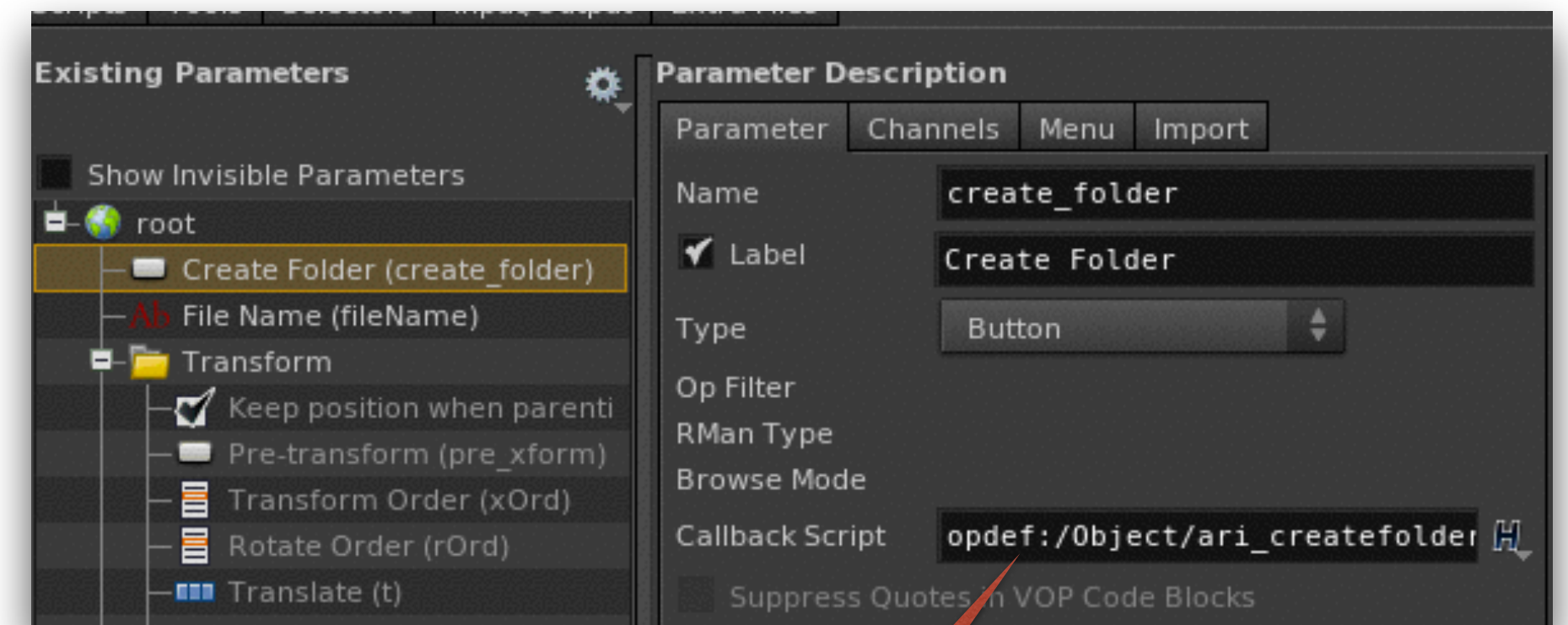




## Step 04 - Creating an OTL with Button Script (cont.)

Now go back to the Scripts tab and modify the Script to take advantage of arg1

```
set theFolder = $arg1 "_" `padzero(4,$F4) `  
message $theFolder  
umkdir $HIP/${theFolder}  
message "I have written a folder" $HIP/${theFolder}
```



Callback Script



# End Part 03

Inline Tools



# Part 04

Digital Assets



# Creating a VOP Operator

Or... Making a little Mid Level VOPs to help the artist



# Why Create a VOP Operator?

There are many times you are repeating the same steps in a VOP Network.

A few months pass before you need the same network and by then you forgot how to do it

Artists coming from other packages might not have the technical acumen to create full network

**Create a VOP Operator for a SHOP Context that does Noise Shaping**

**The new operator will start with the code from the AANoise OP and add:**

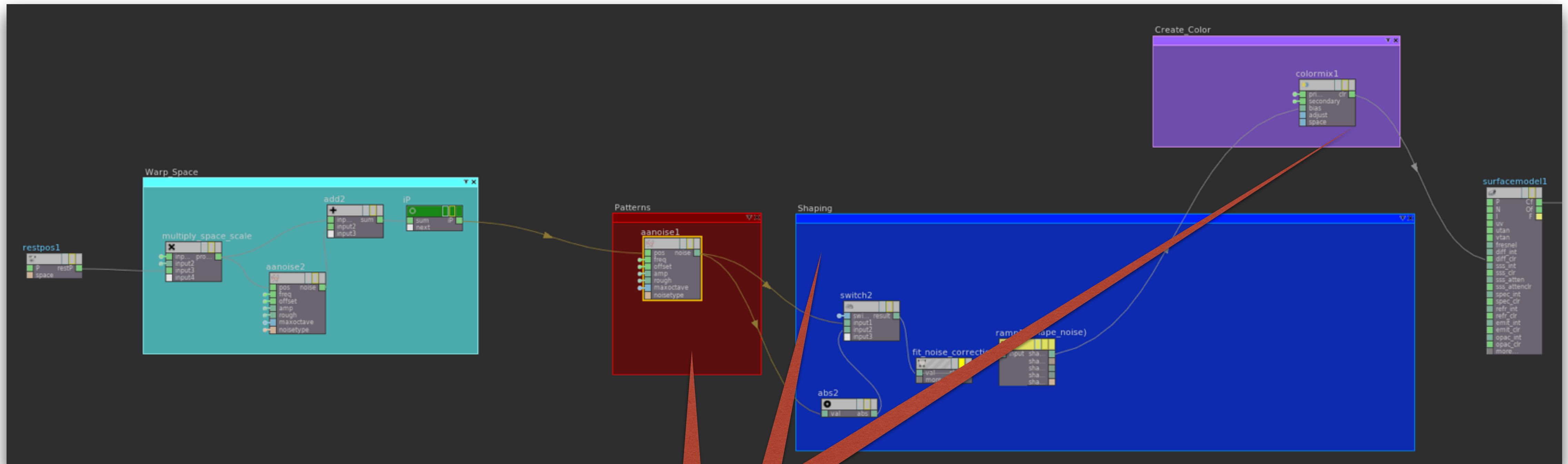
**Folding**

**Color mixing based on noise**

**Scalar multiplying the noise based on a spline ramp**

**Output color and noise for displacement**

# The Original Network



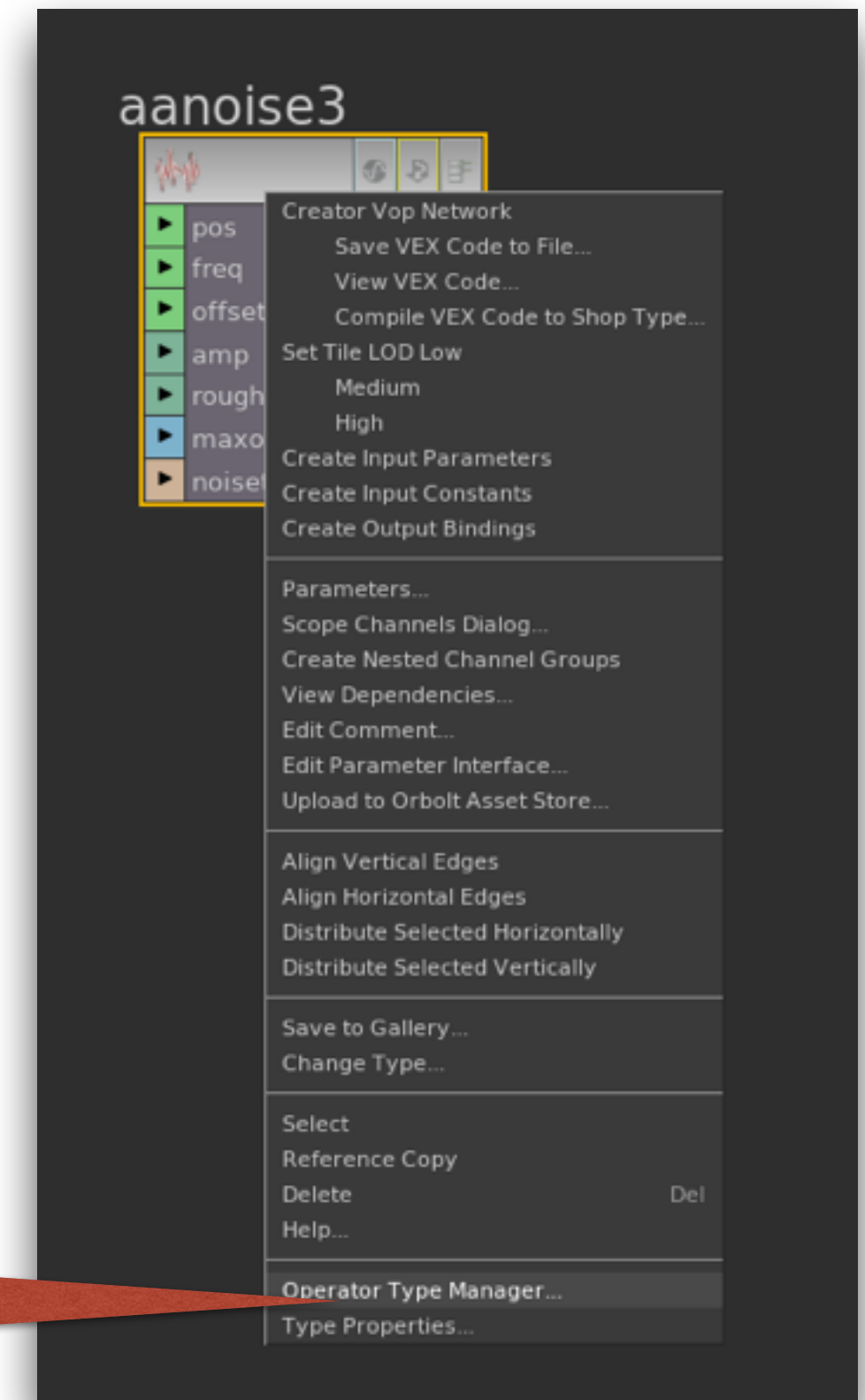
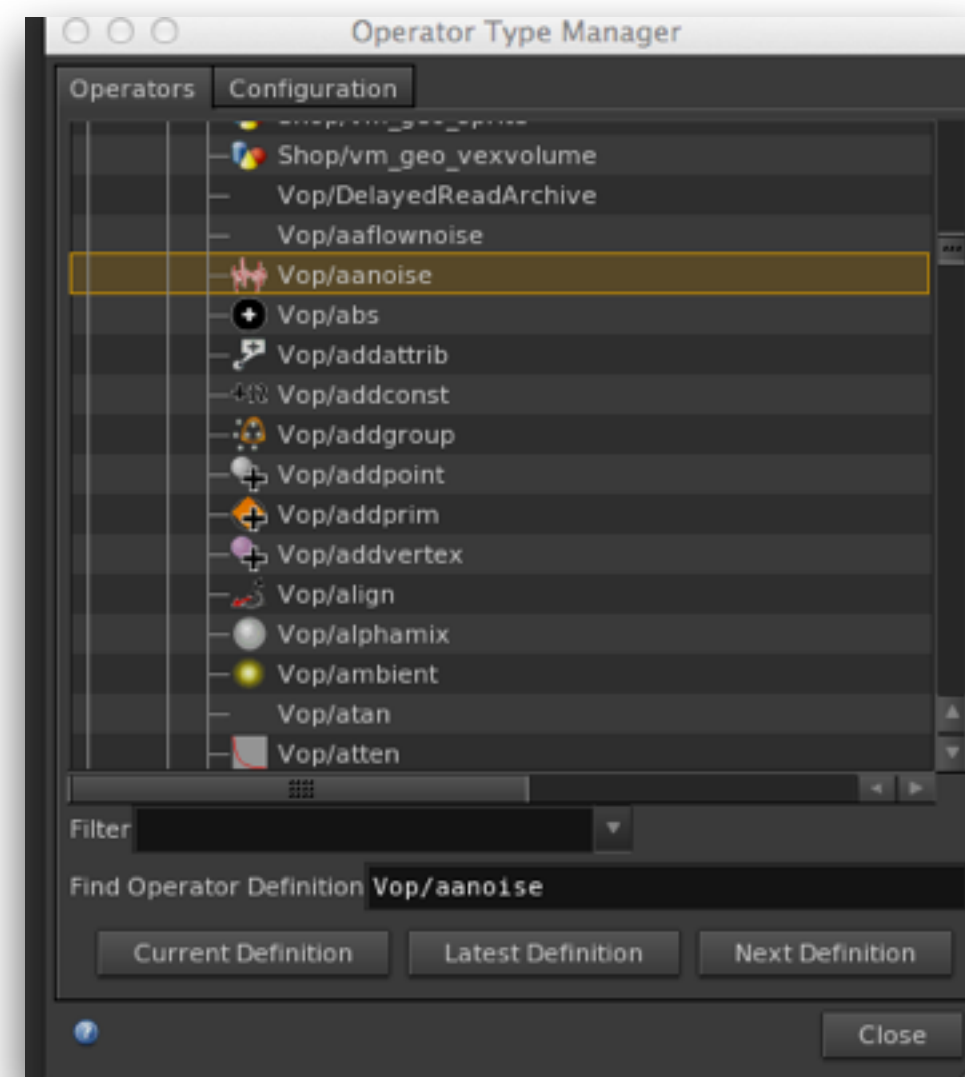
We want to collapse Patterns, Shaping, and Create Color boxes into one VOP

# Step 01 - Duplicating the OTL

Drop down an anti-alias noise VOP

Right click on the VOP and select Operator Type Manager

A New Dialog Box “Operator Type Manager” should pop up with the VOP/aanoise highlighted



Operator Type  
Manager



# Step 01 - Operator Type Manager to Duplicate Noise

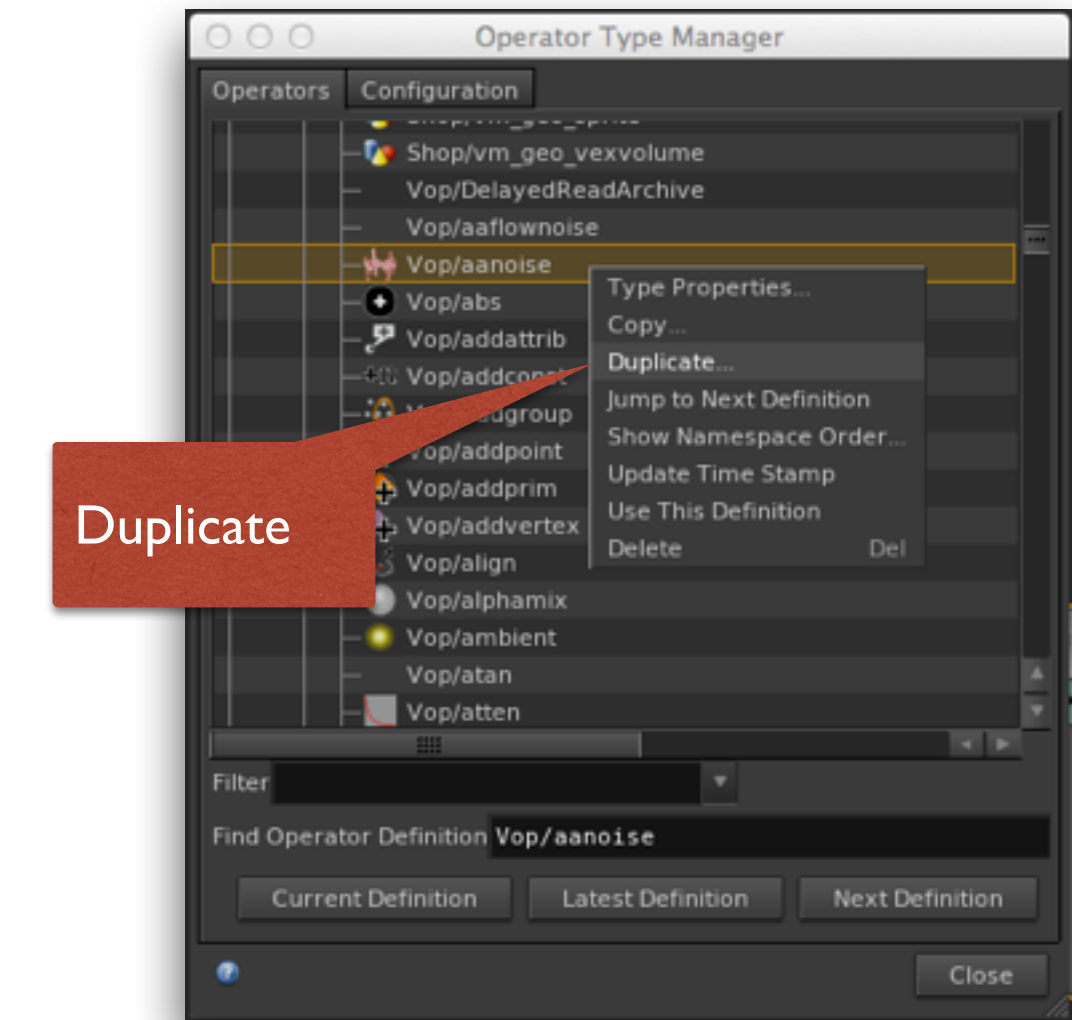
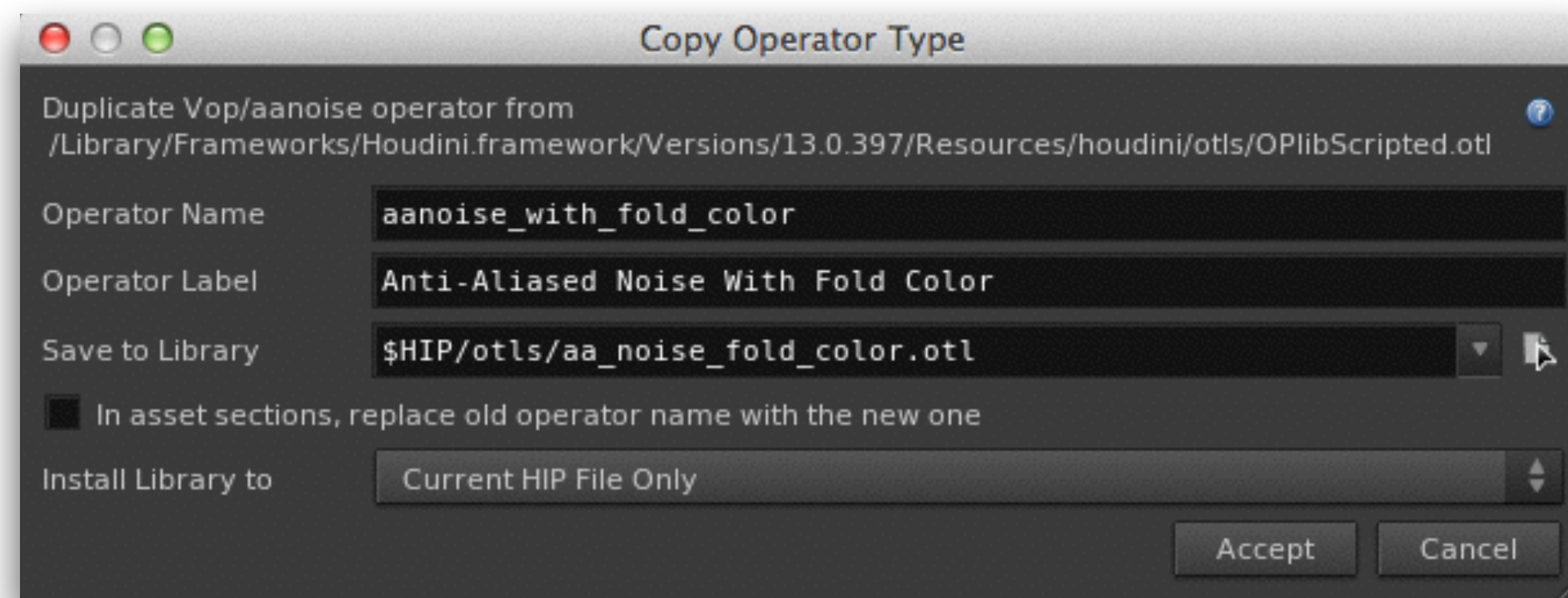
Right Click on the highlighted operator and select “Duplicate”

In the dialog that appears create a new name and label

Save to your project folder/ otl folder

Deselect - In asset sections

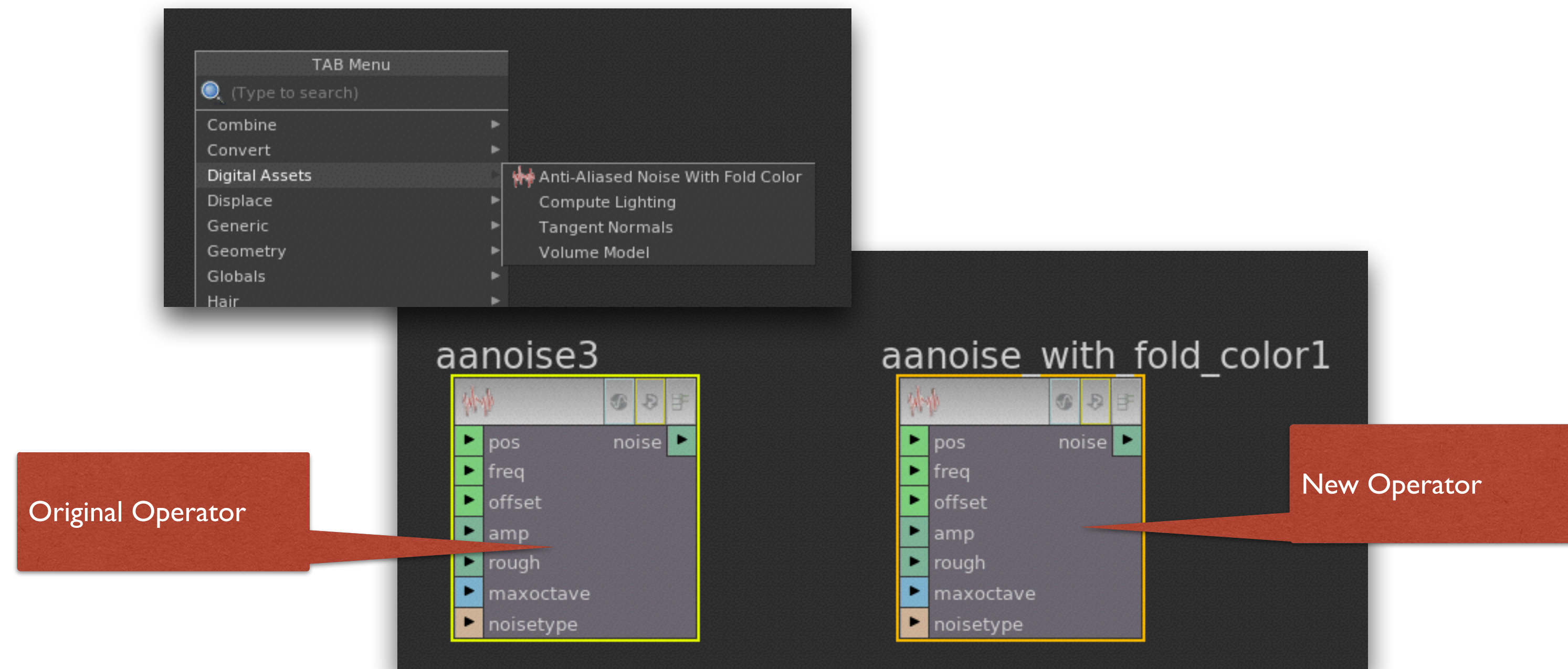
Click “Accept”



# Step 01 - Seeing the New Operator

Click the Tab Key and under Digital Assets

Select “Anti-Alias Noise With Fold Color”



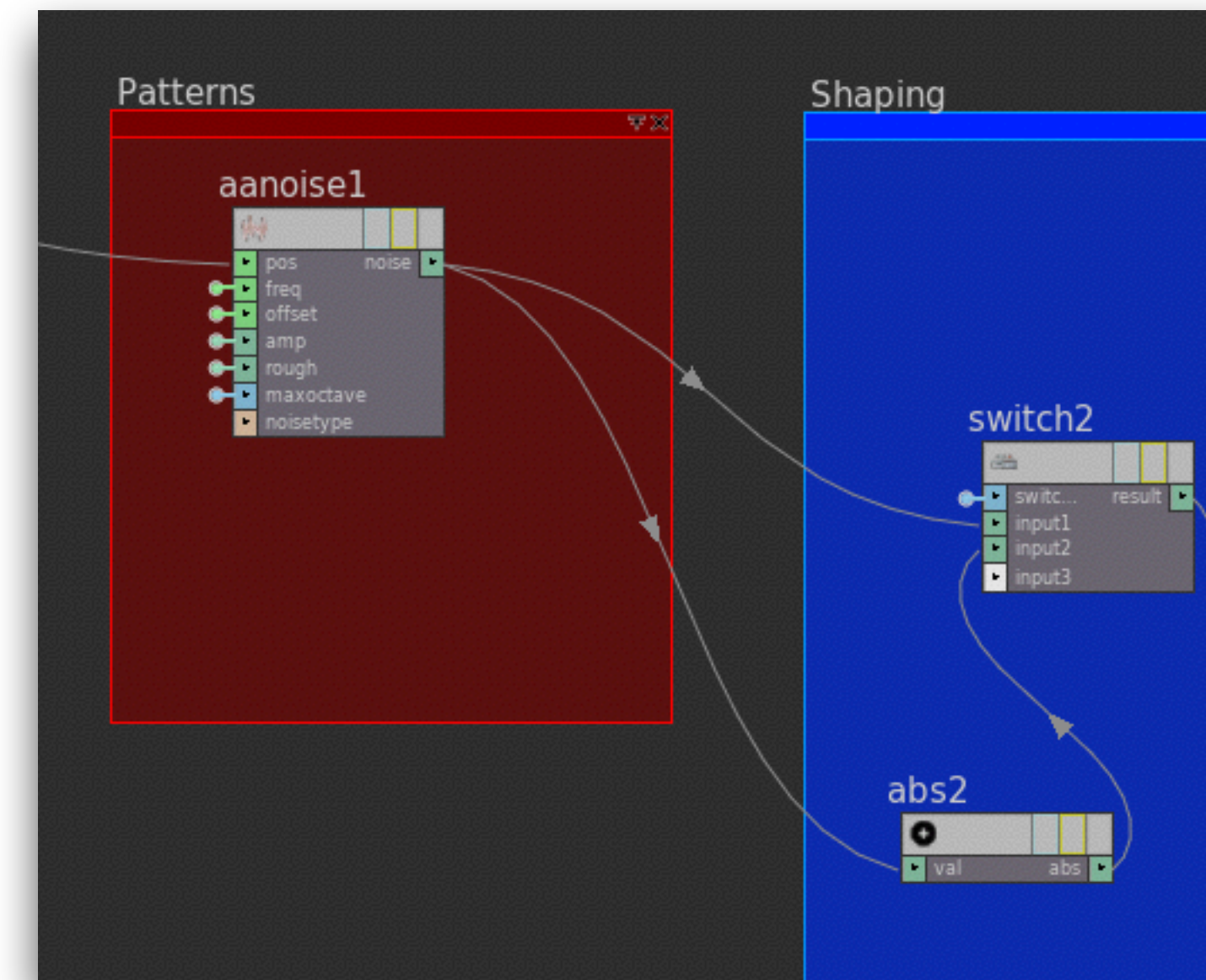
## Step 02 - Add Folding to the New Operator

In the Original Network we had a switch and a ABS operator after the anti-aliased noise.

This allowed the artist to choose between shape folding or no shape folding

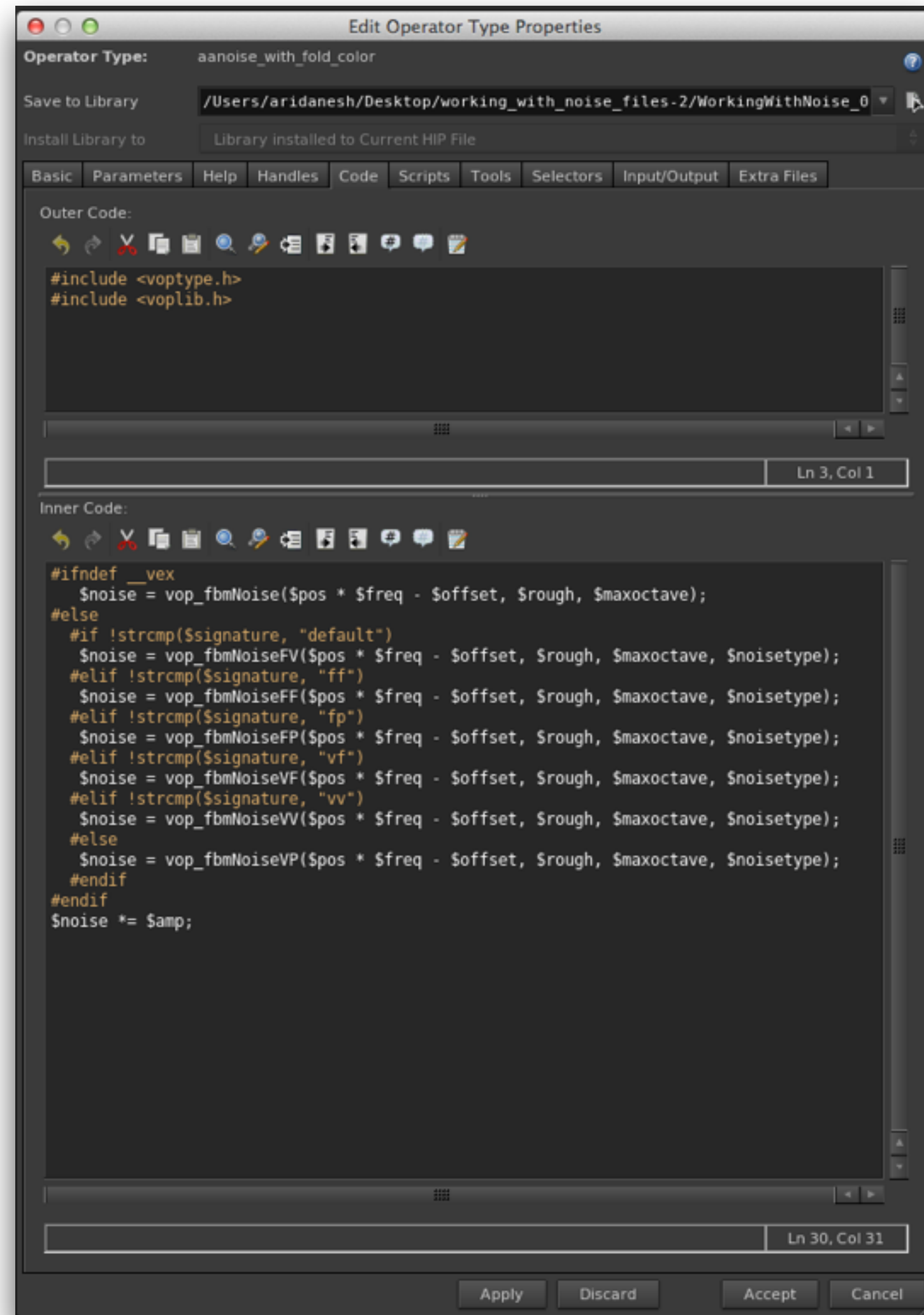
We will now implement this in our new operator

Open the Type Properties for the new operator





# Examine the VEX Code for Noise



In the Type Library select the Code Tab

Notice that there two sections

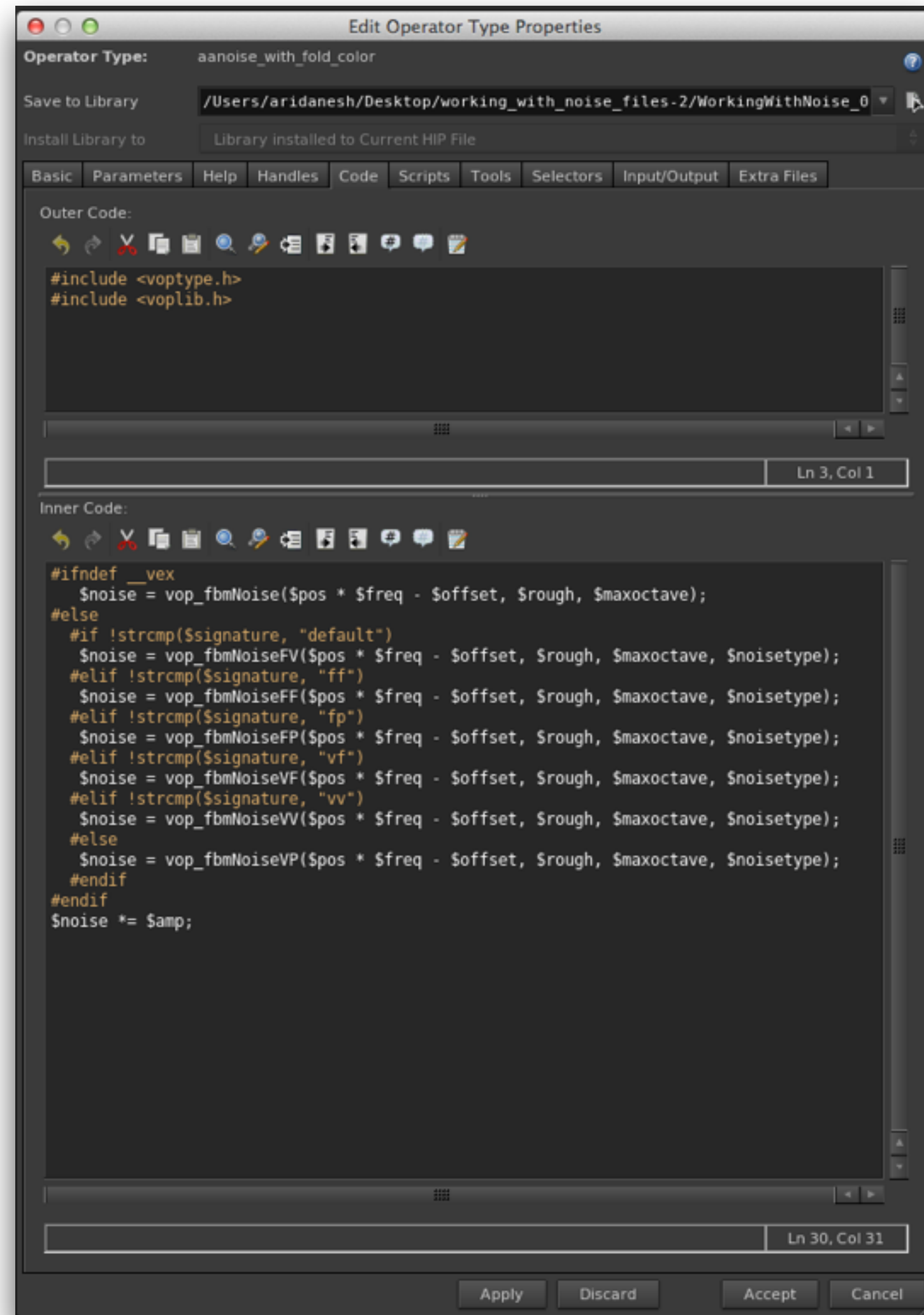
Outer Code - is used to import libraries

Inner Code - is where you write your logic

Notice that the code to execute a noise function is already there



# Why Are There So Many Noise Functions?



**Signatures - Just like you are used to wiring certain VOPs up to different datatypes (e.g., float, vector) and seeing the color tab for the node change depending on the input data type vop\_fbmNoise has many signatures that it can calculate**

# Where can I find the function in documentation?

You Can't!

If you launched Houdini from a shell you can easily navigate to the includes

```
cd $HH/vex/include
```

To see voplib.h

```
cat voplib.h
```

# Step 02 - Adding Folding

Go to the parameters tab and add a toggle

Name - fold

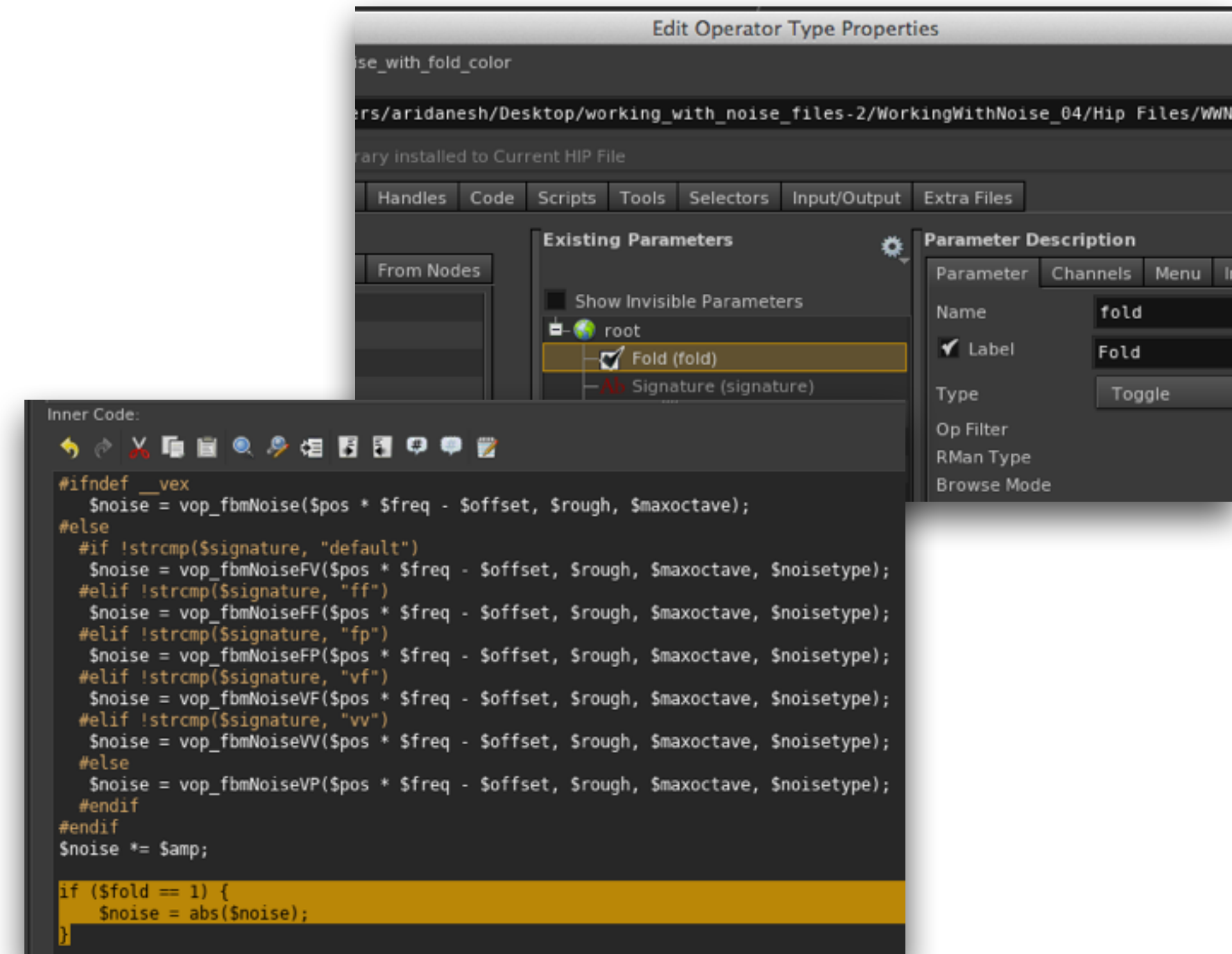
Label - Fold

Go to the Code Tab - in the Inner Code Append

```
if ($fold == 1) {  
    $noise = abs($noise);  
}
```

Notice fold is typed with “\$” prepended

When calling a parameter you need to prepend with \$



# The Input/Output Tab

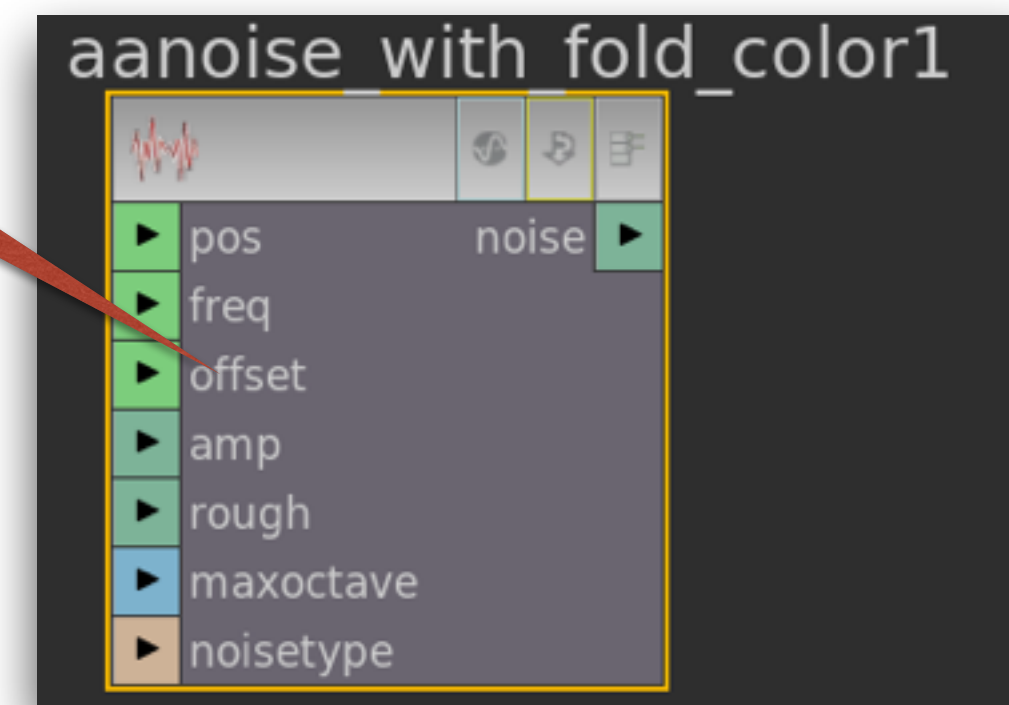
I want to be able to wire in the Fold option

Currently my VOP Operator look like the original aanoise VOP

We want to add a input connection for “fold”

In the Type Library select the Input/Output Tab

Looks like the original





# The Input/Output Tab

I want to be able to wire in the Fold option

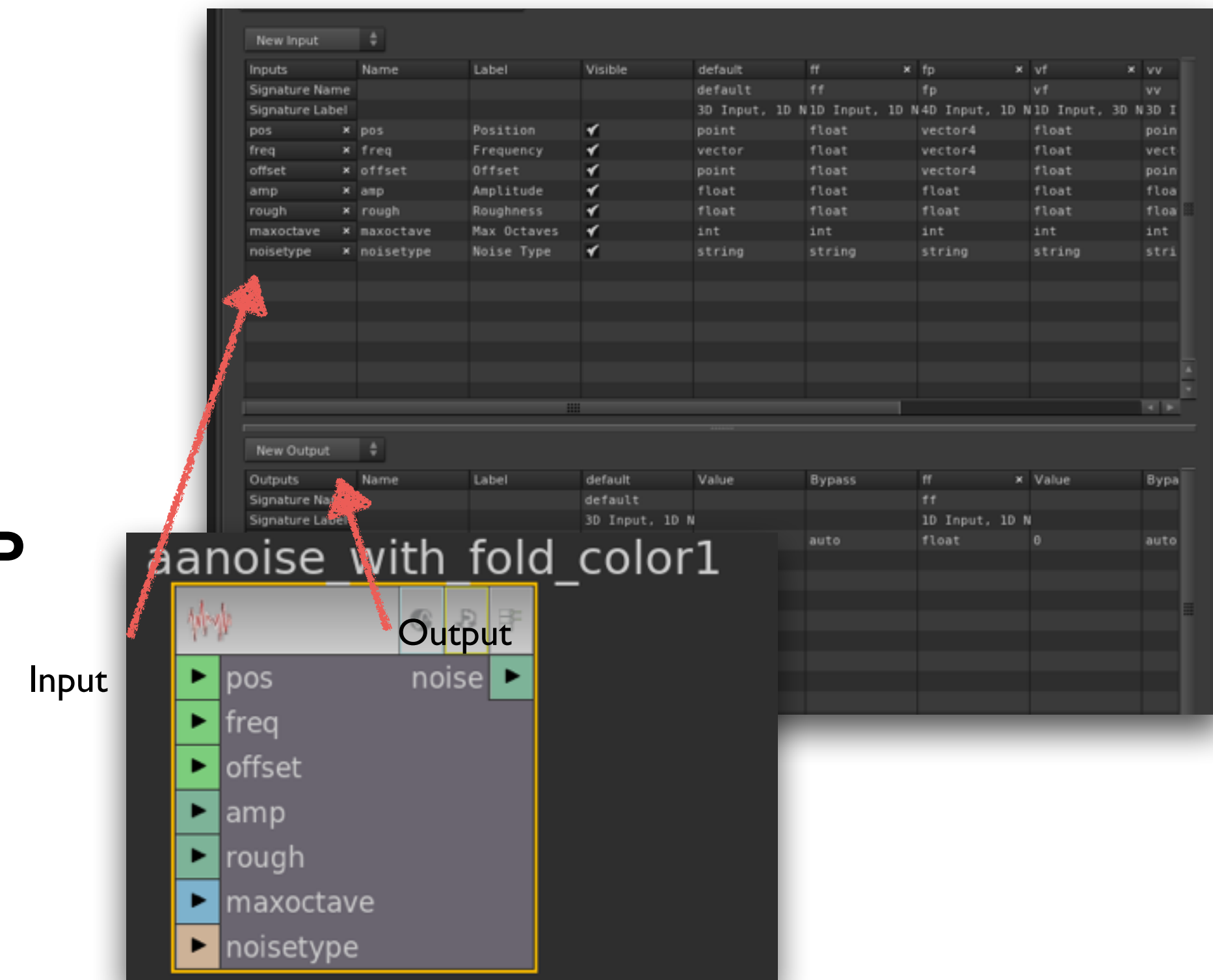
Currently my VOP Operator look like the original aanoise VOP

We want to add a input connection for “fold”

In the Type Library select the Input/Output Tab

The Top section corresponds to the input tabs of the VOP

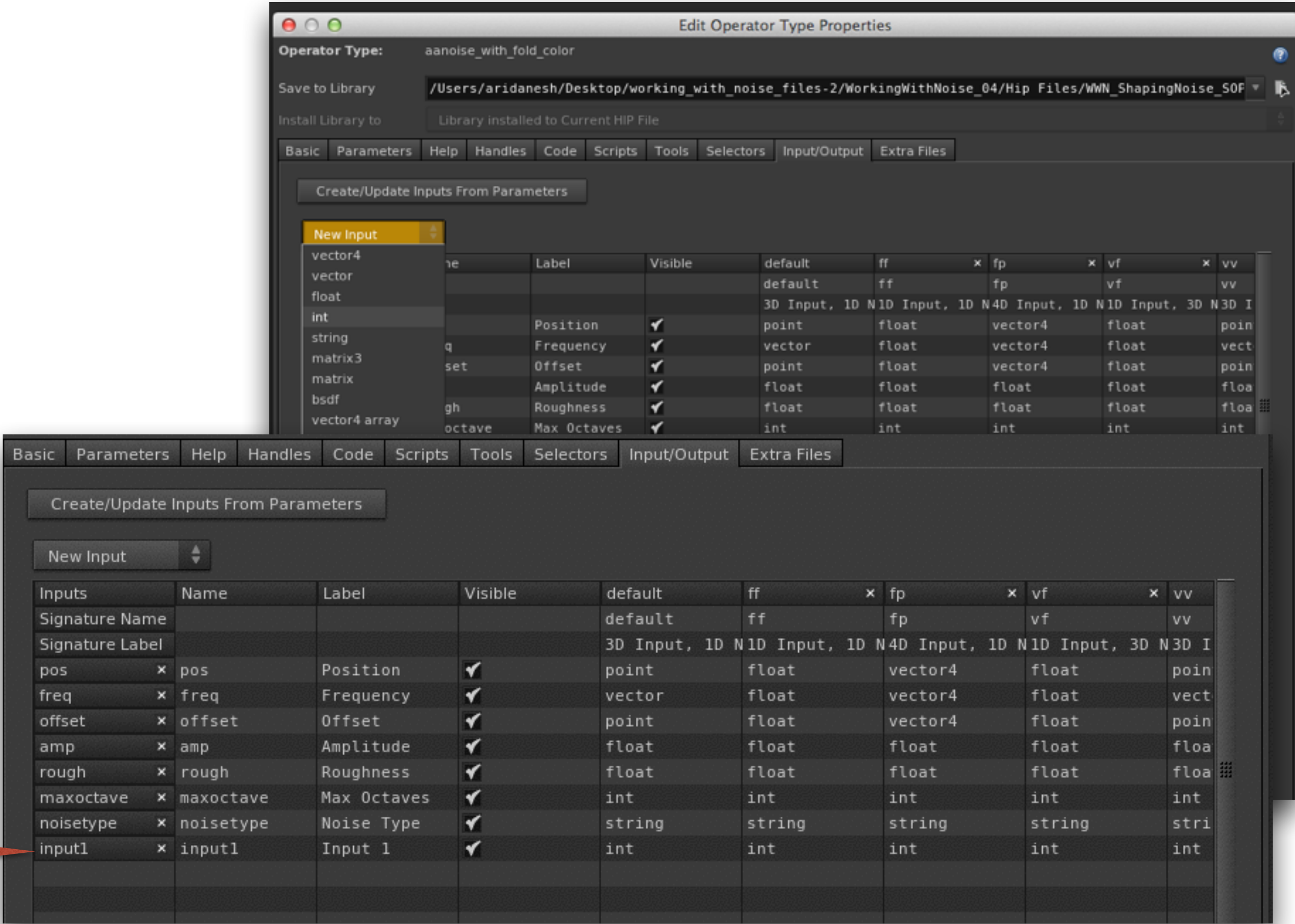
The Bottom Section corresponds to the output tabs of the VOP



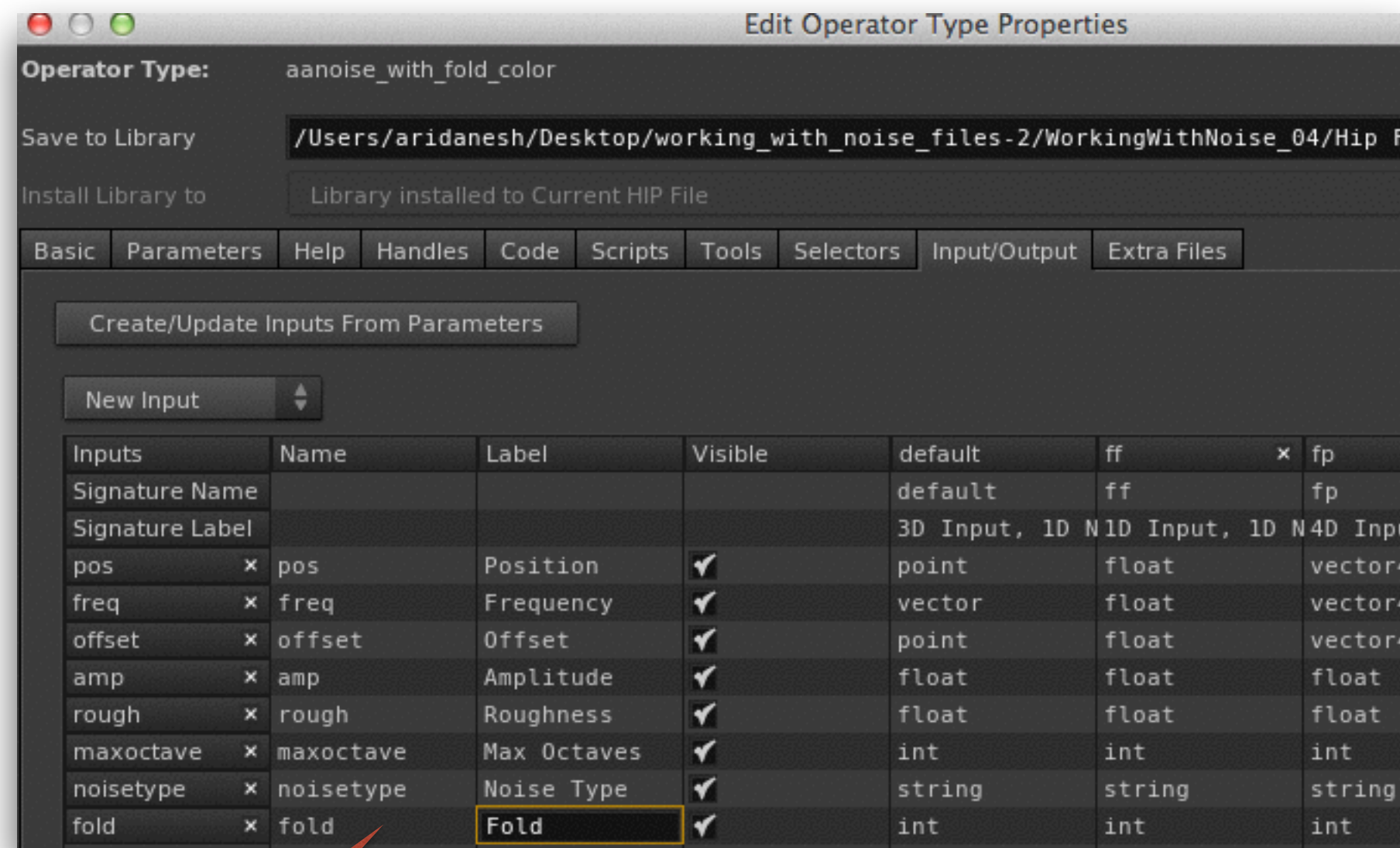
# Adding the Input for the Fold Toggle

- Click on New Input and select int
- In the Name column replace input1 with fold
- In the Label Column replace Input 1 with Fold
- Click Apply

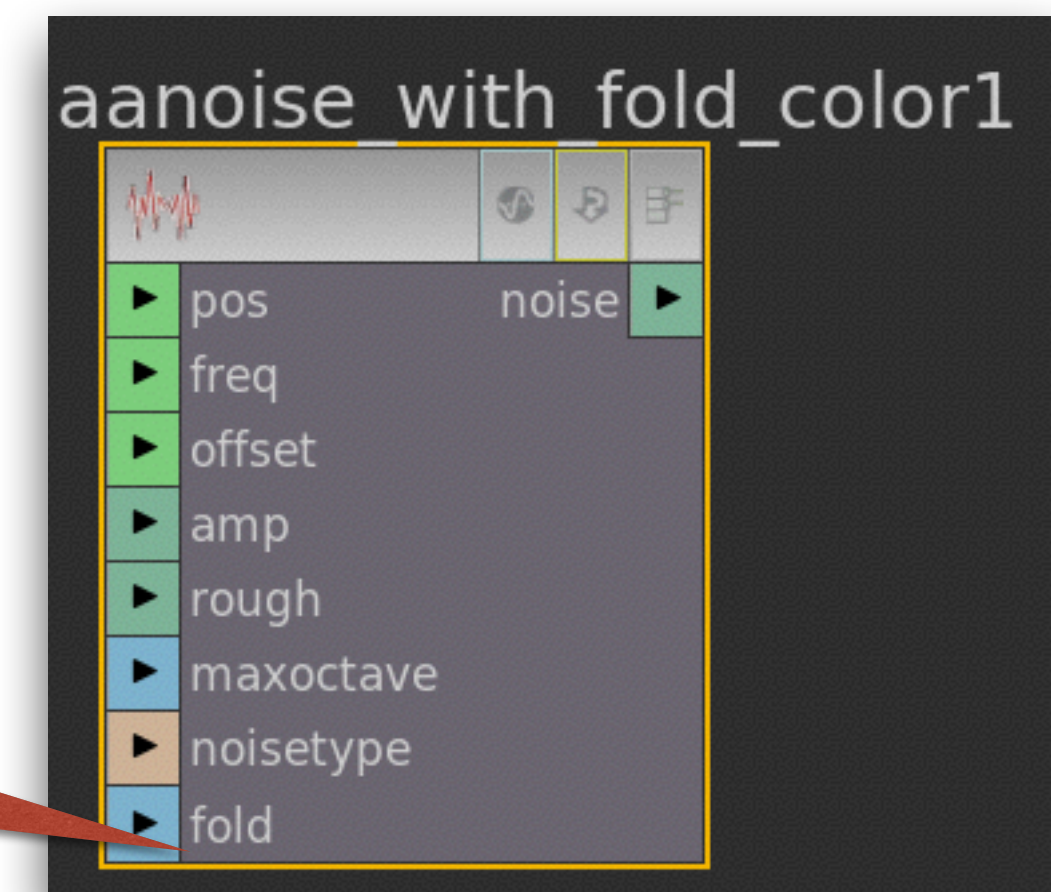
Replace these fields



# The Results



Updated Field Names



New Fold tab color  
coded bue for int

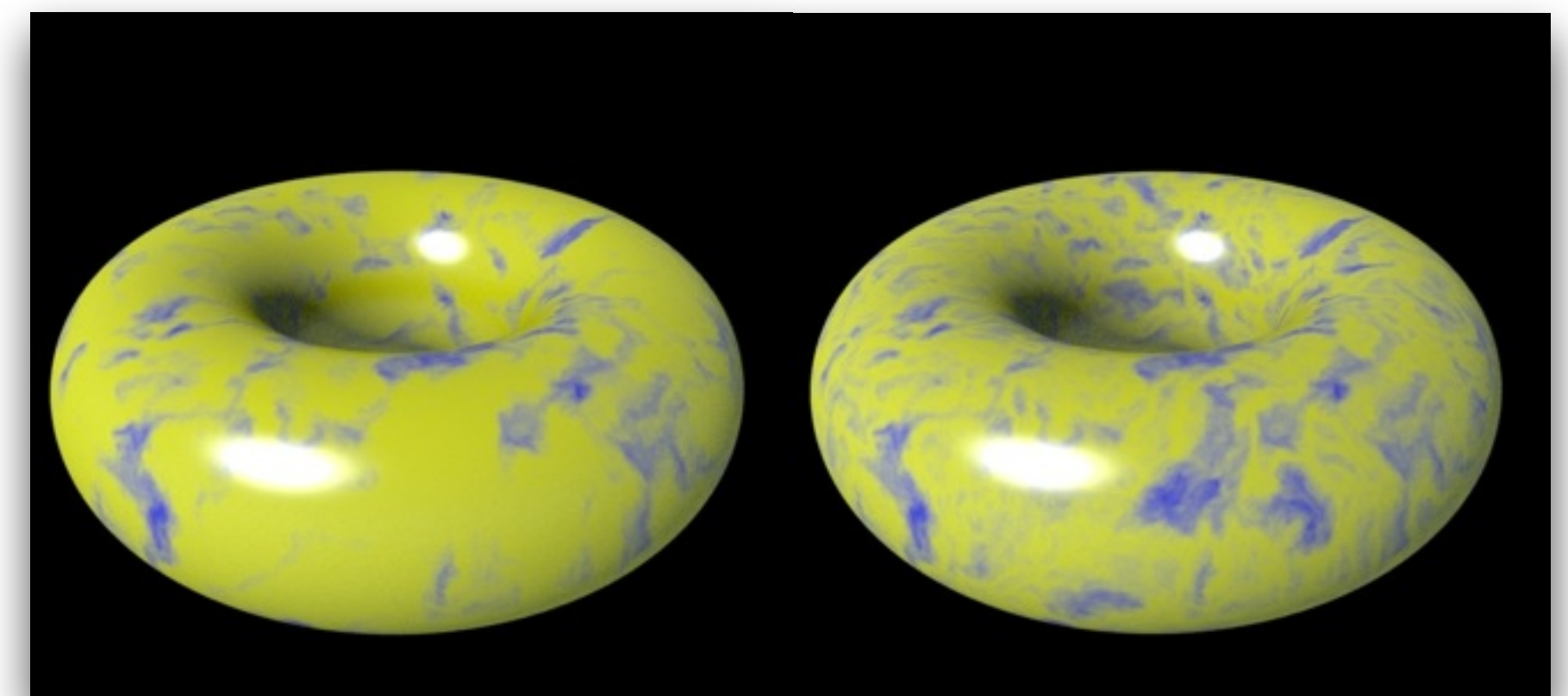
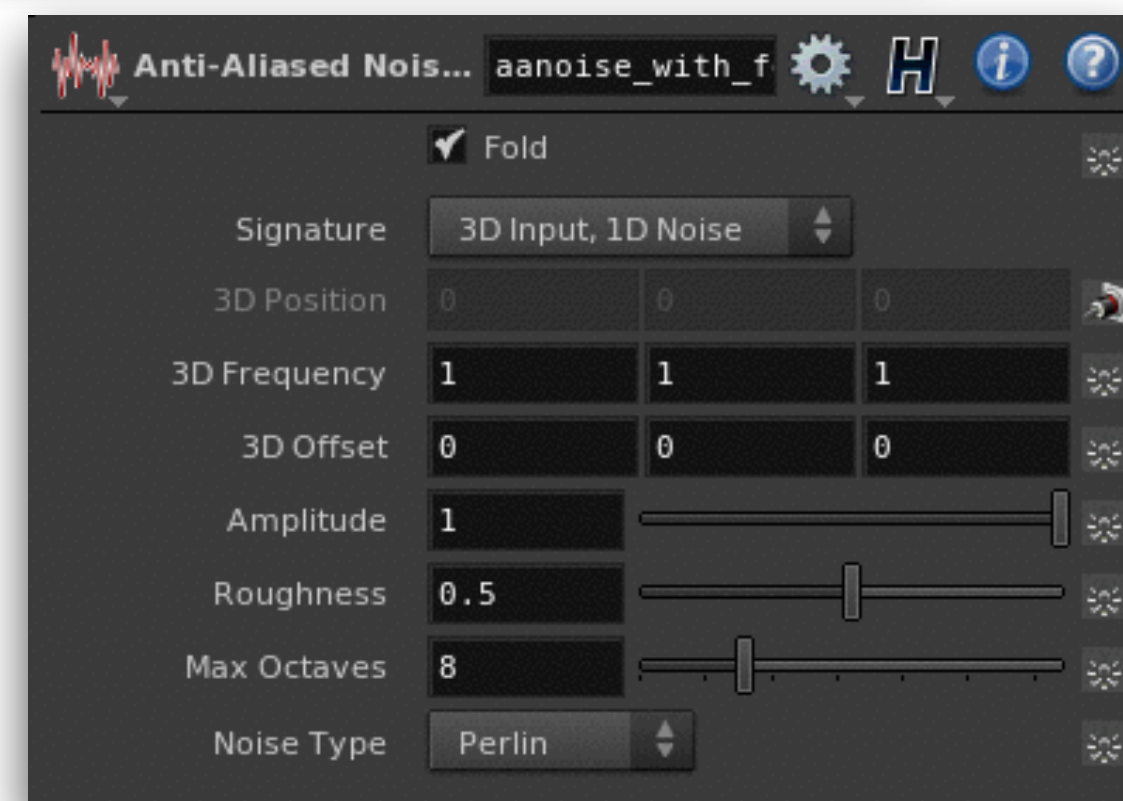


# A Quick Test

Use the new operator instead

Rewire the network to use the new noise operator  
Bypass the switch and abs operators

Bypass these 3 nodes



no fold

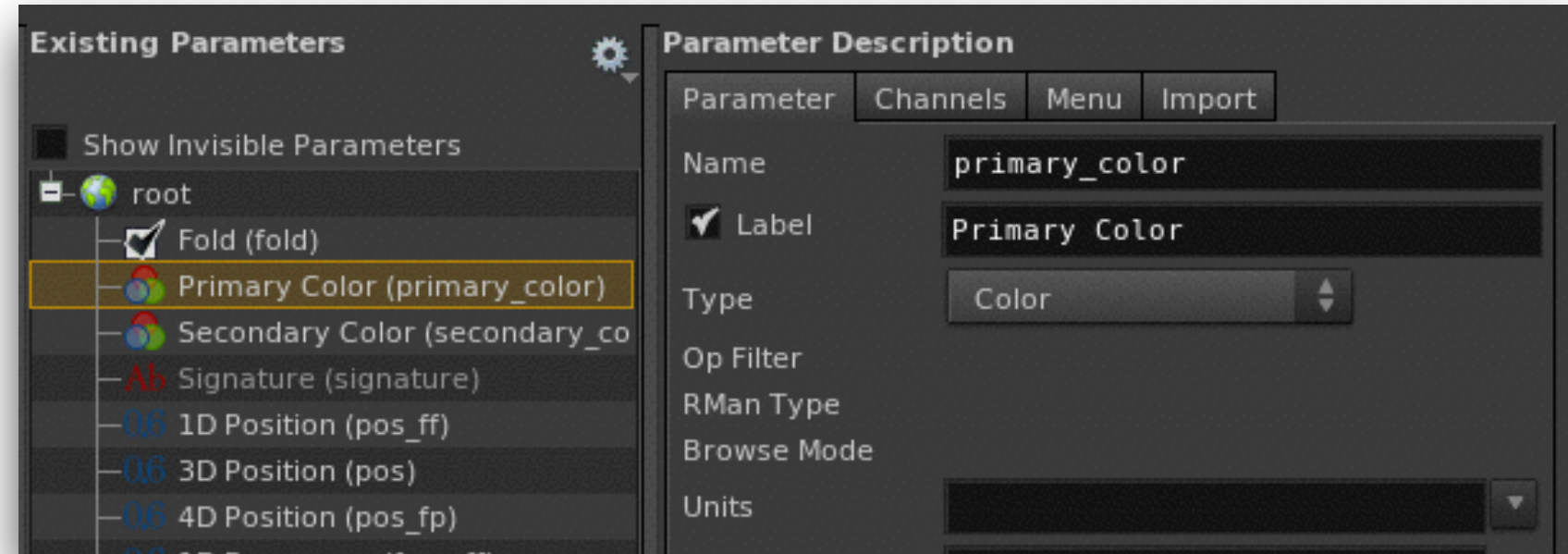
fold



## Step 03 - Adding Color to the Operator

We will want to create to color inputs into the operator and do a color mix that is controlled by the noise we just generated

## Step 03 - Adding Color



Open the Type Library once again

In the parameters add two color parameters

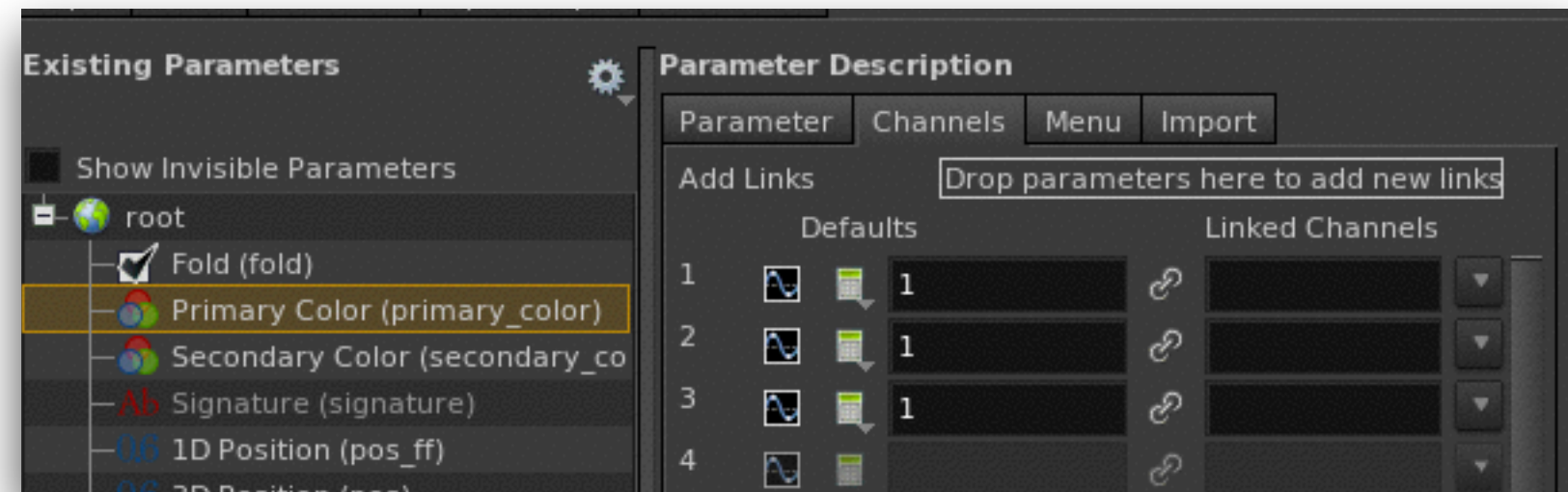
name: primary\_color, secondary\_color

label: Primary Color, Secondary Color

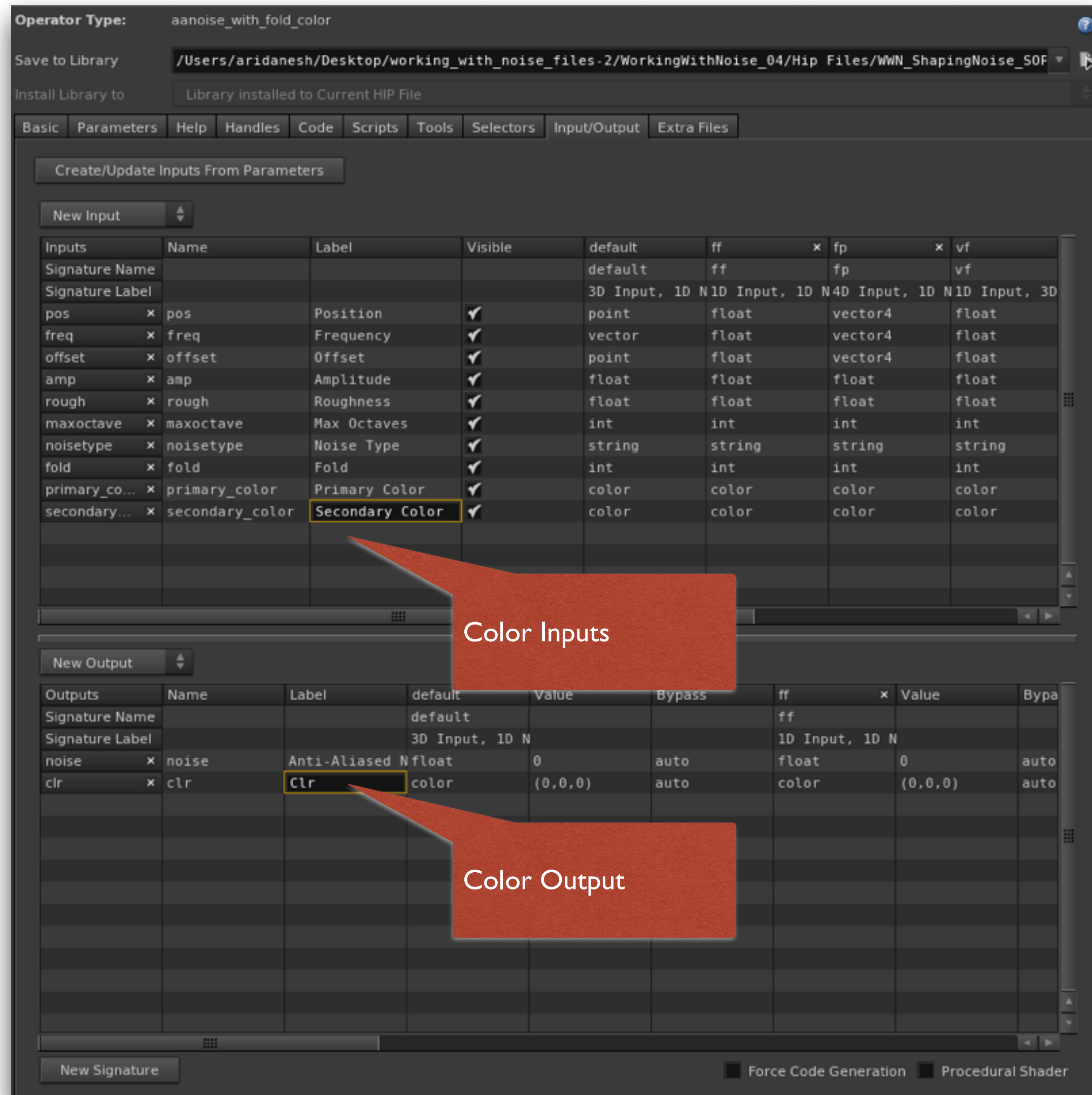
Make the Color defaults:

primary\_color: (1,1,1)

secondary\_color (0,0,0)



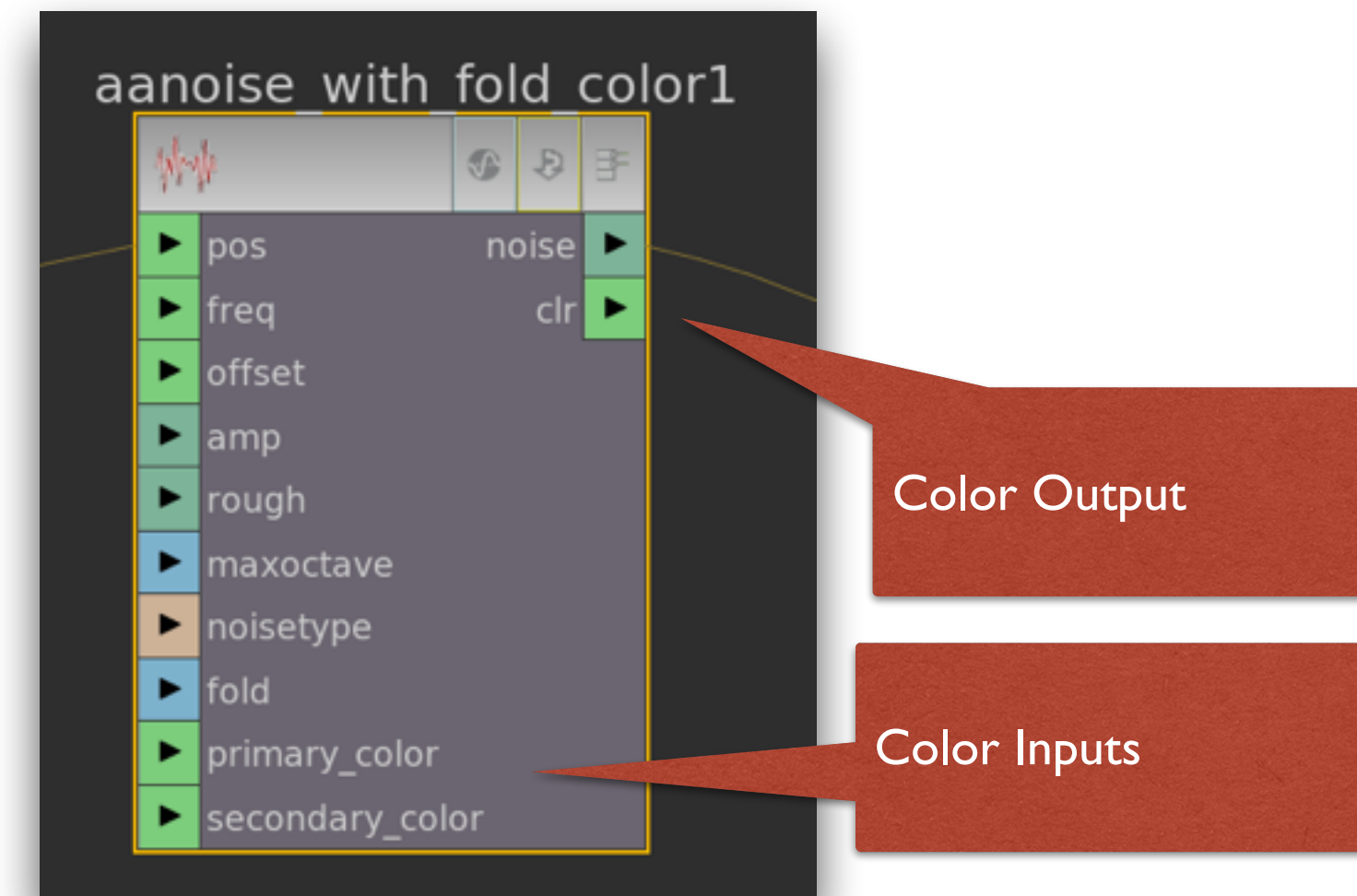
## Step 03 - Adding Color (cont.)



Go to the Input/Output Tab

Add the two color inputs - primary\_color, secondary\_color

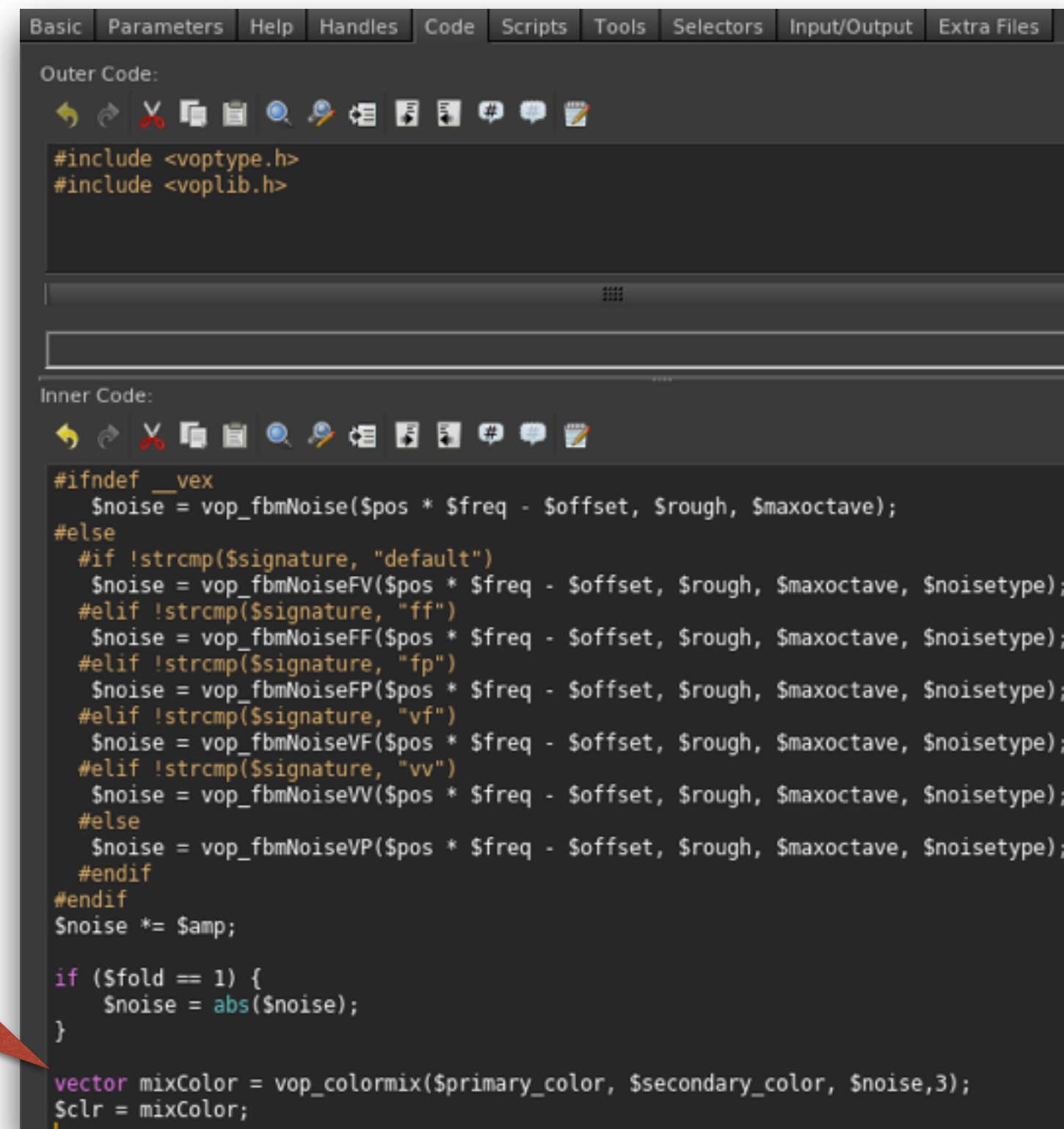
Add an output - clr



## Step 03 - Adding the Code

```
vector mixColor = vop_colormix($primary_color, $secondary_color, $noise,3);  
$clr = mixColor;
```

New code



```
Basic Parameters Help Handles Code Scripts Tools Selectors Input/Output Extra Files  
Outer Code:  
#include <voptype.h>  
#include <voplib.h>  
  
Inner Code:  
#ifndef __vex  
    $noise = vop_fbmNoise($pos * $freq - $offset, $rough, $maxoctave);  
#else  
    #if !strcmp($signature, "default")  
        $noise = vop_fbmNoiseFV($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #elif !strcmp($signature, "ff")  
        $noise = vop_fbmNoiseFF($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #elif !strcmp($signature, "fp")  
        $noise = vop_fbmNoiseFP($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #elif !strcmp($signature, "vf")  
        $noise = vop_fbmNoiseVF($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #elif !strcmp($signature, "vv")  
        $noise = vop_fbmNoiseVV($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #else  
        $noise = vop_fbmNoiseVP($pos * $freq - $offset, $rough, $maxoctave, $noisetype);  
    #endif  
#endif  
$noise *= $amp;  
  
if ($fold == 1) {  
    $noise = abs($noise);  
}  
  
vector mixColor = vop_colormix($primary_color, $secondary_color, $noise,3);  
$clr = mixColor;
```



# What is vop\_colormix

```
vector
vop_colormix(vector c1, c2; float bias; int adjust)
{
    vector    clr;
    if (adjust == 3)
        clr = cspline(bias, c1, c1, c2, c2);
    else if (adjust == 2)
        clr = lerp(c1, c2, float(smooth(0, 1, bias)));
    else if (adjust == 1)
        clr = lerp(c1, c2, float(clamp(bias, 0, 1)));
    else
        clr = lerp(c1, c2, bias);
    return clr;
}
```

This code can be found in voplib.h

As you can see this code is a more intelligent version of a lerp()

It has the additional characteristic that it is compatible with RSL

## lerp VEX function

*Performs bilinear interpolation between the values.*

1. `float lerp(float value1, float value2, float amount)`
2. `vector lerp(vector value1, vector value2, float amount)`
3. `vector4 lerp(vector4 value1, vector4 value2, float amount)`

Performs bilinear interpolation between the values. If the amount is outside the range 0 to 1, the values will be extrapolated linearly.

# Assignment - Wire In Smoothing

Currently in the vop\_colormix code smoothing is hard coded to “3”

It would be better for the artist to be able to choose which smoothing function the vop\_colormix should use

Add a integer parameter and make it a menu. One item for each of the four options in vop\_colormix

Make the parameter a input into the node

Change the vop\_colormix to read the parameter

# Wire In Smoothing

New integer param - smoothing

Menu items for smoothing

Declaring the input

Updated Code

New Input

ts Tools Selectors Input/Output Extra Files

Existing Parameters

- root
  - Fold (fold)
  - Signature (signature)
  - Color
    - Primary Color (primary\_color)
    - Secondary Color (secondary\_color)
    - Smoothing (smoothing)
  - Noise
    - Noise Type (noisetype)
    - Amplitude (amp)
    - Roughness (rough)
    - Max Octaves (maxoctave)

Parameter Description

Parameter Channels Menu Import

Name: smoothing

Label: Smoothing

Type: Integer

Op Fi

RMar

Brow

Units

Size

Callb

S

✓ A

Inter

Invisible

Horizontally Join to Next Parameter

Range: 0 3

Parameter Description

Parameter Channels Menu Import

Use Menu: Normal (Menu Only, Single Selecti...)

Menu Items Menu Script

Token	Label	
0	bias	x
1	clamp	x
2	smooth	x
3	cspline	x

Basic Parameters Help Handles Code Scripts Tools Selectors Input/Output

Create/Update Inputs From Parameters

New Input

Inputs	Name	Label	Visible	default
Signature Name				default
Signature Label				3D Input, 1D M
pos	x pos	3D Position	✓	point
freq	x freq	3D Frequency	✓	vector
offset	x offset	3D Offset	✓	point
amp	x amp	Amplitude	✓	float
rough	x rough	Roughness	✓	float
maxoctave	x maxoctave	Max Octaves	✓	int
noisetype	x noisetype	Noise Type	✓	string
fold	x fold	Fold	✓	int
primary_color	x primary_color	Primary Color	✓	color
secondary_color	x secondary_color	Secondary Color	✓	color
signature	x signature	Signature	✓	string
srcmin	x srcmin	Label	✓	float
srcmax	x srcmax	Source Max	✓	float
destmin	x destmin	Destination Mi	✓	float
destintion_max	x destintion_max	Destination Ma	✓	float
smoothing	x smoothing	Smoothing	✓	int

shape and color\_noise1

```
vector mixColor = vop_colormix($primary_color, $secondary_color, $noise, $smoothing);  
$clr = mixColor;
```

- rough
- maxoctave
- noisetype
- fold
- primary\_color
- secondary\_color
- signature
- srcmin
- srcmax
- destmin
- destintion\_max
- smoothing



## Riddle Me This...

**Why in a SOP otl do you have to “Save Operator Type” and “Match Current Definition” while In a VOP/VEX otl you do not save but just Apply/Accept?**

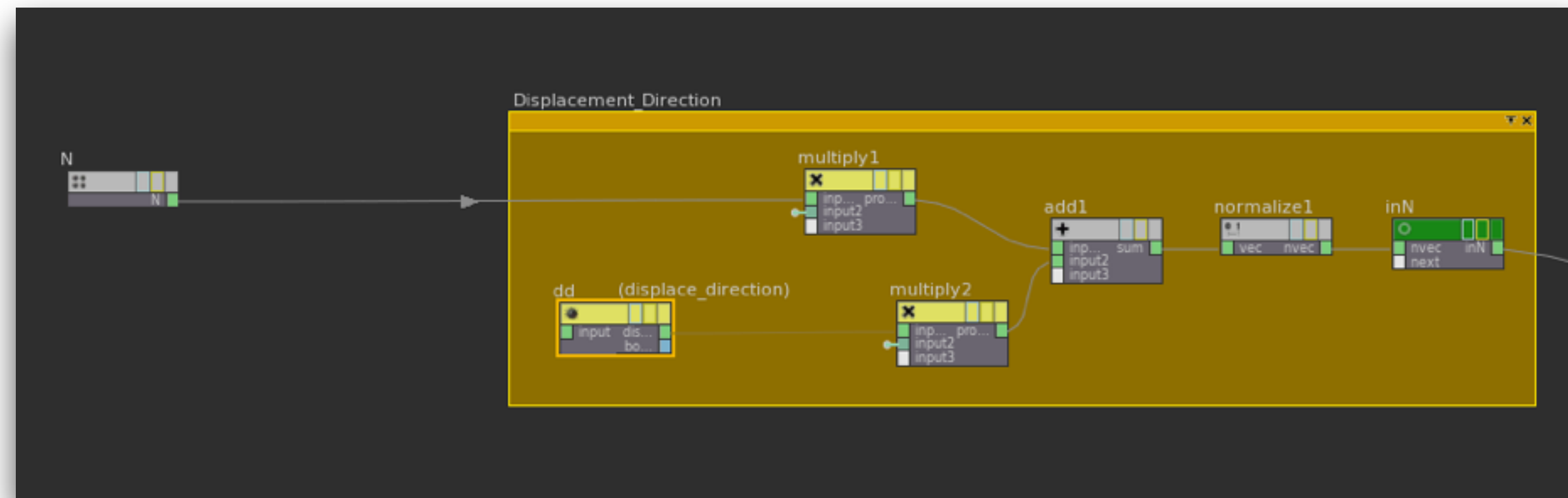
**All assets other then VOP types are simple Node networks. They do not get compiled at any time unless they themselves contain VOP Networks**

**With VOP/VEX, when you hit the Apply/Accept button it saves the code but then compiles the the asset into assembly by the LLVM compiler**

**The gotcha - Because VEX is compiled all parameters must be declared variables in the function arguments.**



# Making a VOP Operator from Scratch



Here is a simple network that changes the normals for displacement direction. Basically this logic is used for shearing and scaling bumps in the geometry

Let us make a custom VOP Operator called Displace\_Direction that contains this logic

# Creating the “Empty Asset”

Go to the Menu Bar at the top of Houdini. Click on:

**File—> New Operator Type...**

**Name:** displacement\_direction

**Label:** Displacement Direction

**What is Operator Style?**

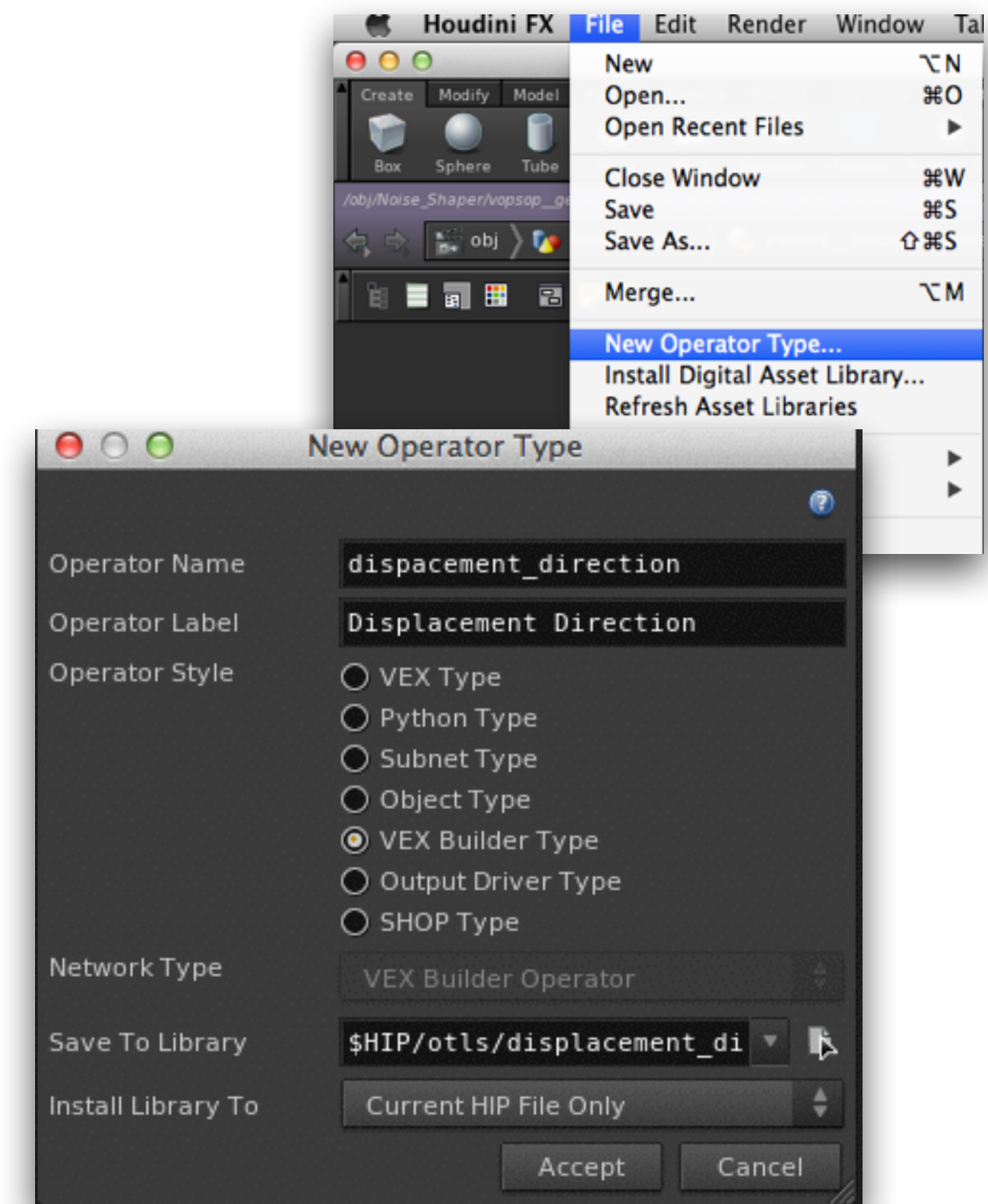
In our case we have two choices:

**VEX Type & VEX Builder Type**

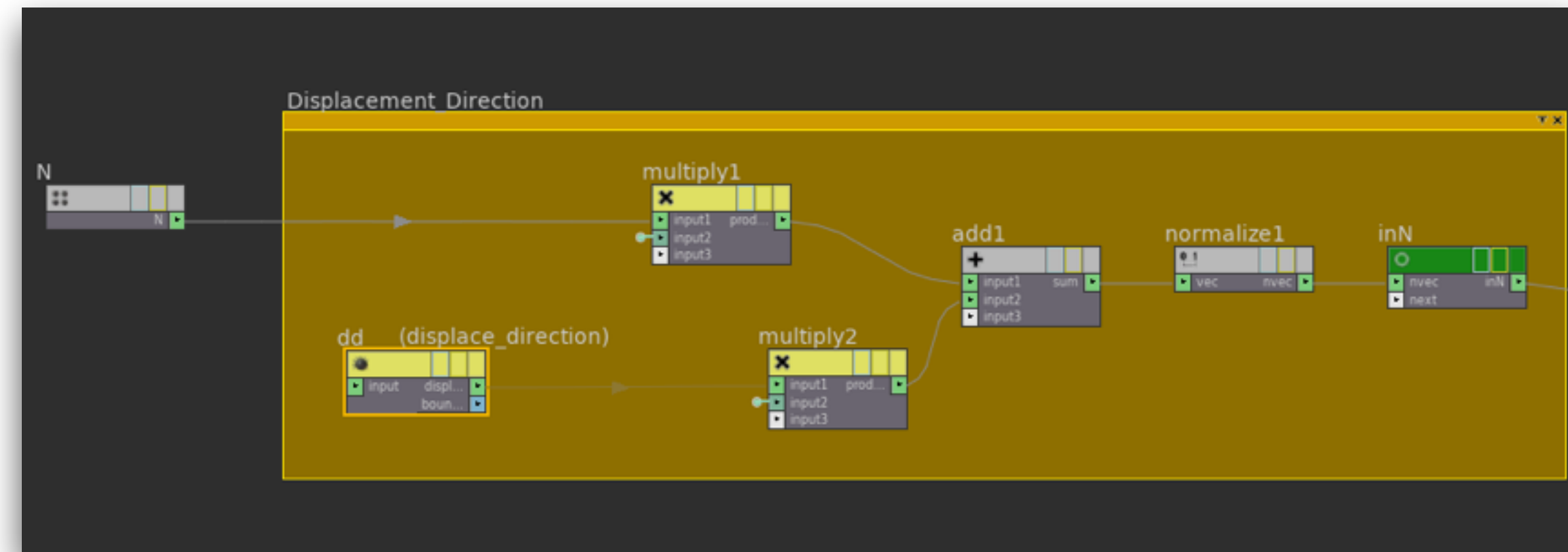
**VEX Type** - Use this option if you want to build an operator for a specific context/sub context. (i.e., surface shaders, displacement shaders)

**VEX Builder Type** - Generic VOP, a VOP that will be used in multiple contexts

**Save to Library** - Save to your project folder's otls folder



# What Inputs Do We Need?



The inputs we will need for the custom operator are:

N - Normal (vector)

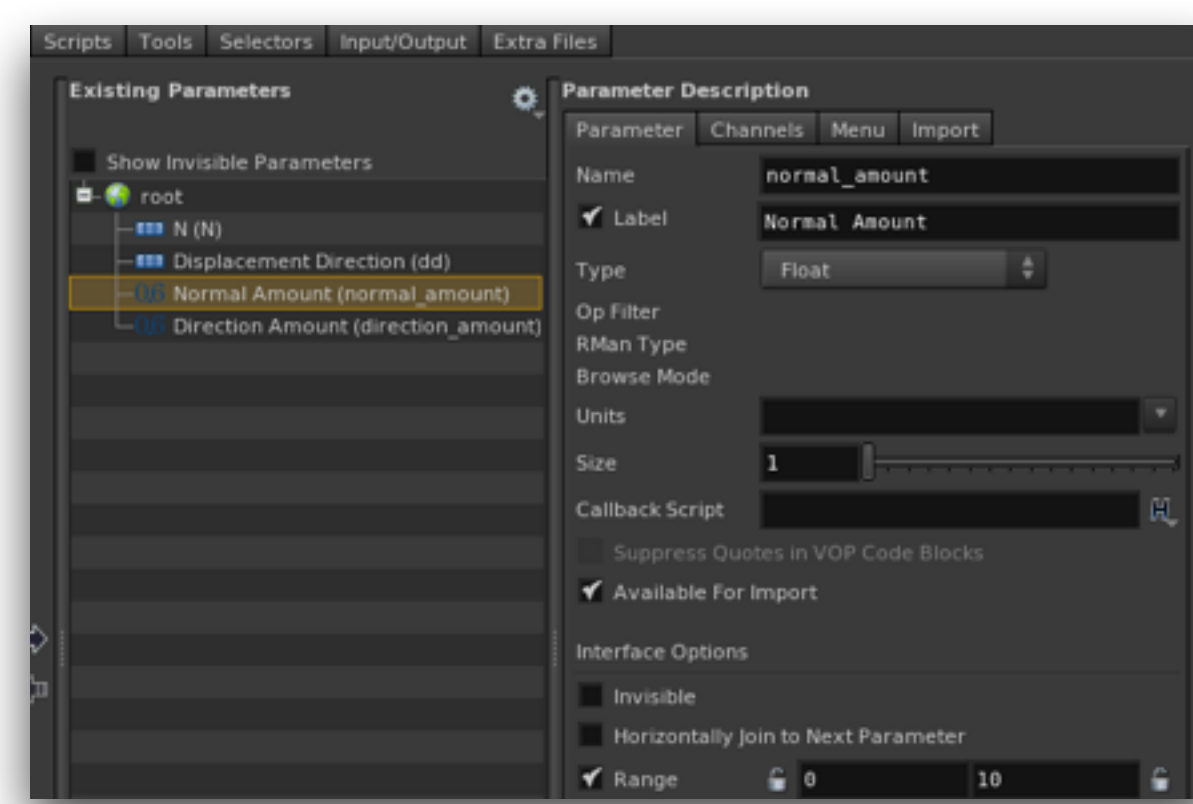
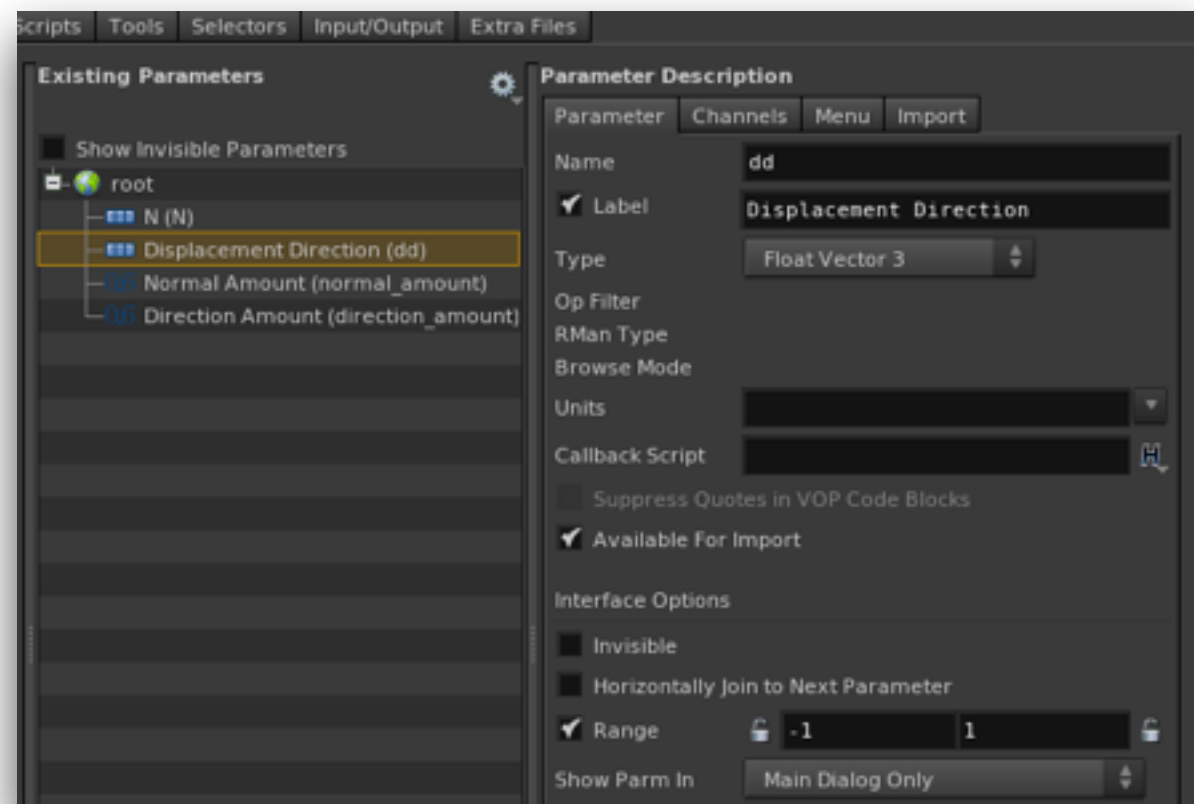
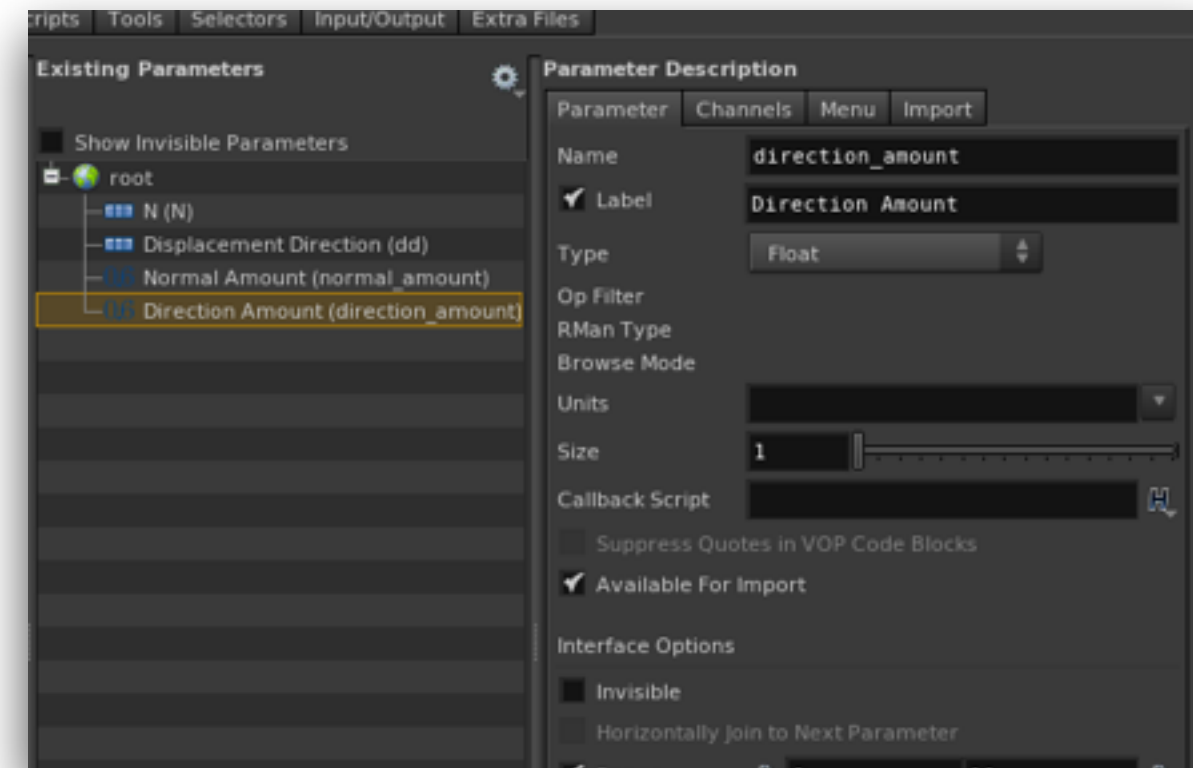
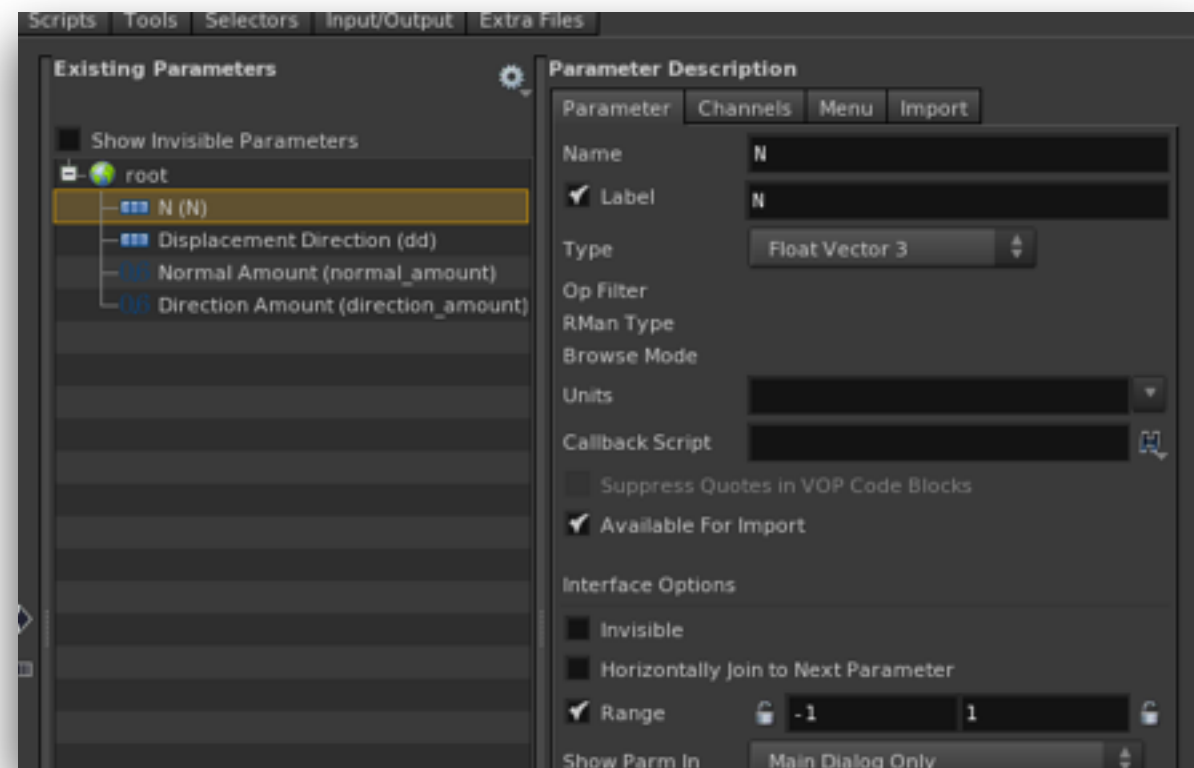
normal\_amount (float) - A scalar multiplier for the normal

displacement\_direction (vector) - which direction do you want to “pull the normals”

direction\_amount - how hard do you want to pull in the displacement direction

# Declaring the Inputs

In the Parameters Tab create the parameters defined on the previous slide

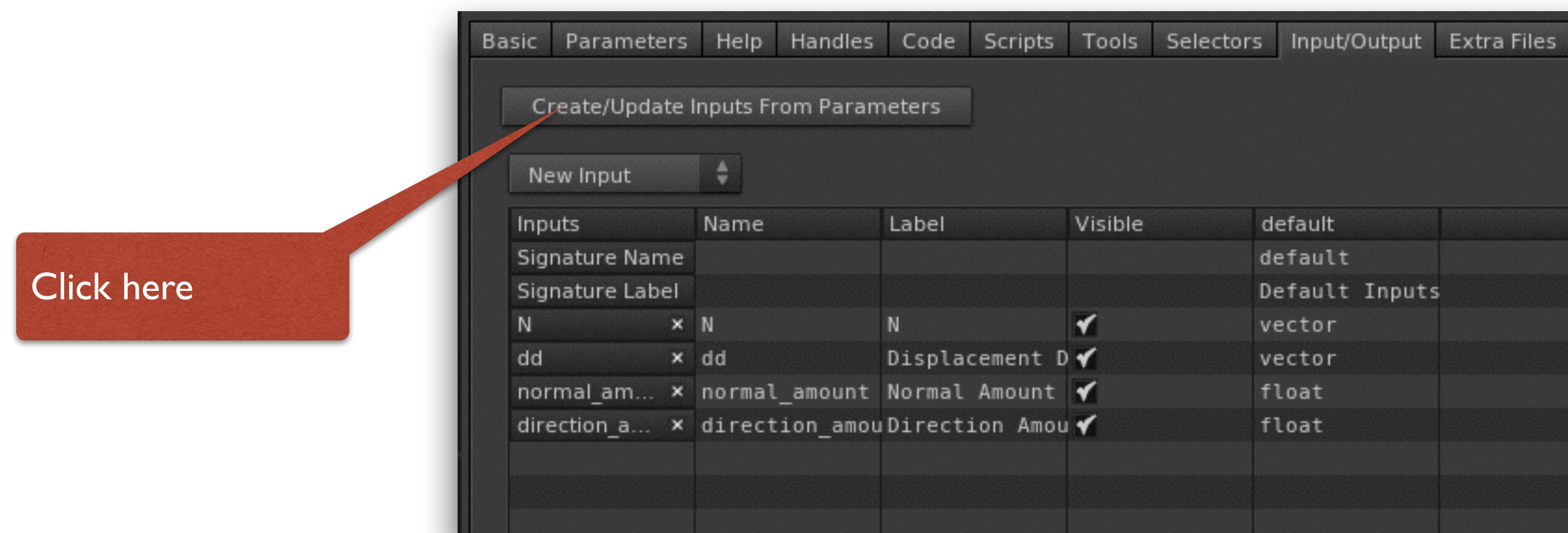




# Declaring the Inputs (cont.)

In the Input/Output Tab click on “Create/Update Inputs From Parameters”

This will bring in to the input/output tab all the parameters you just created



# Adding the Output

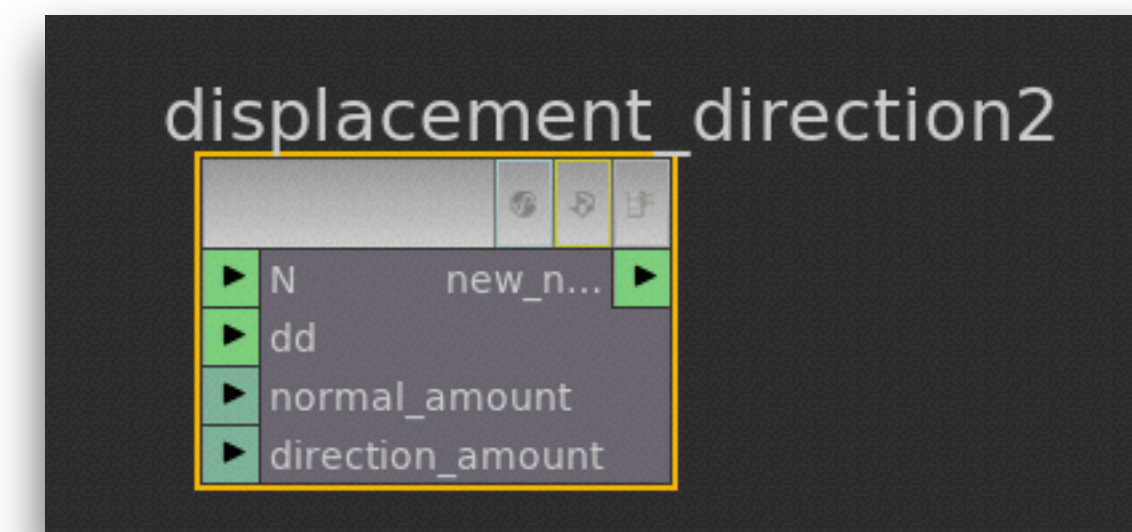
In the Output Section add a vector

Name: new\_normal

Label: New Normal

Click apply and you will be able to drop down your new operator

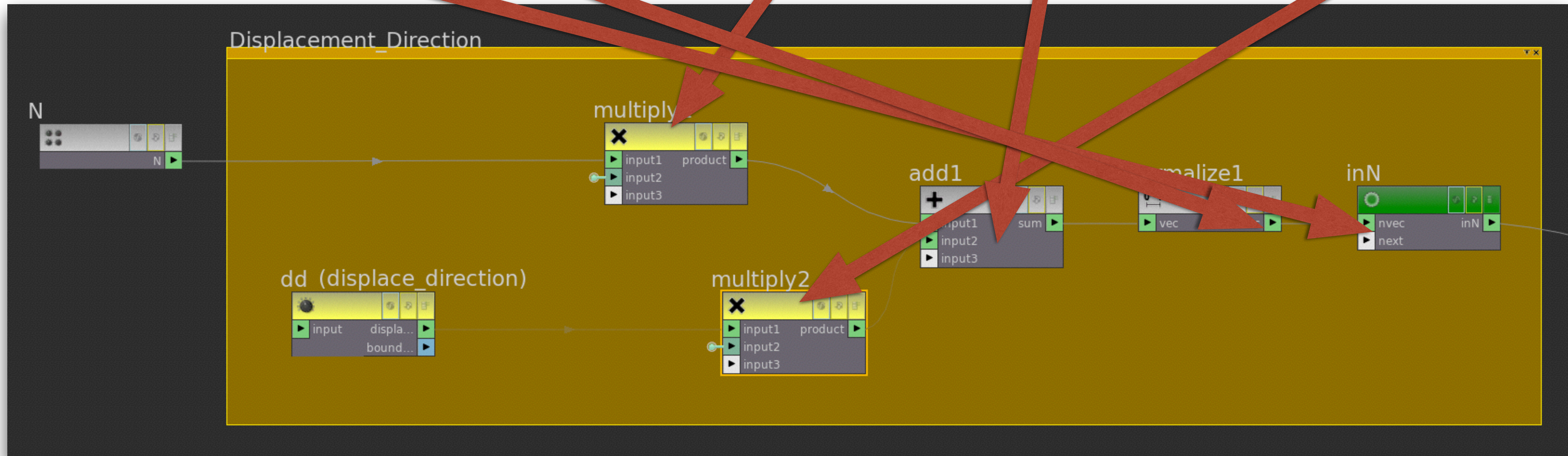
New Output					
Outputs	Name	Label	default	Value	Bypass
Signature Name			default		
Signature Label			Default Inputs		
new_normal x	new_normal	New Normal	vector	(0,0,0)	auto



# Adding the Code

In the Code Tab type the following line of code:

```
$new_normal = normalize(($N * $normal_amount) + ($dd * $direction_amount));
```



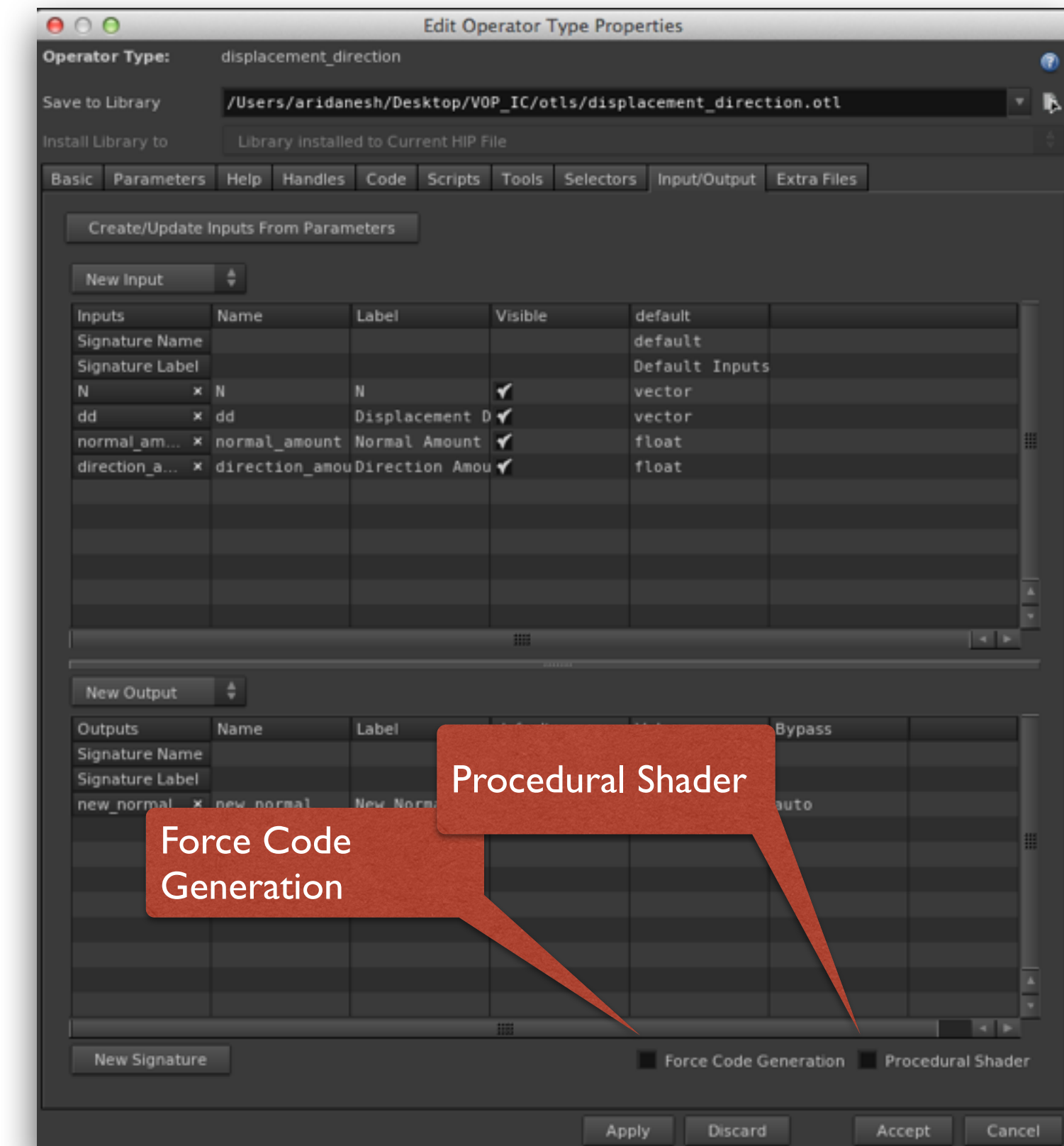


# More on the Input/Output Tab

**Force Code Generation** - When a VOP operator appears in a VOP network, the VEX Builder will only include the code generated by that operator if it determines that its code is required. Generally, this is true for subnet type VOPs, the Output VOP, and any VOP that is connected, directly or indirectly, to the input of a VOP that has required code. However, you can force the VEX Builder to generate the code for your VOP by turning on the Force Code Generation toggle.

**Procedural Shader** - Indicates whether the VOP is implemented by a DSO or DLL file instead of code generated by the VEX Builder.

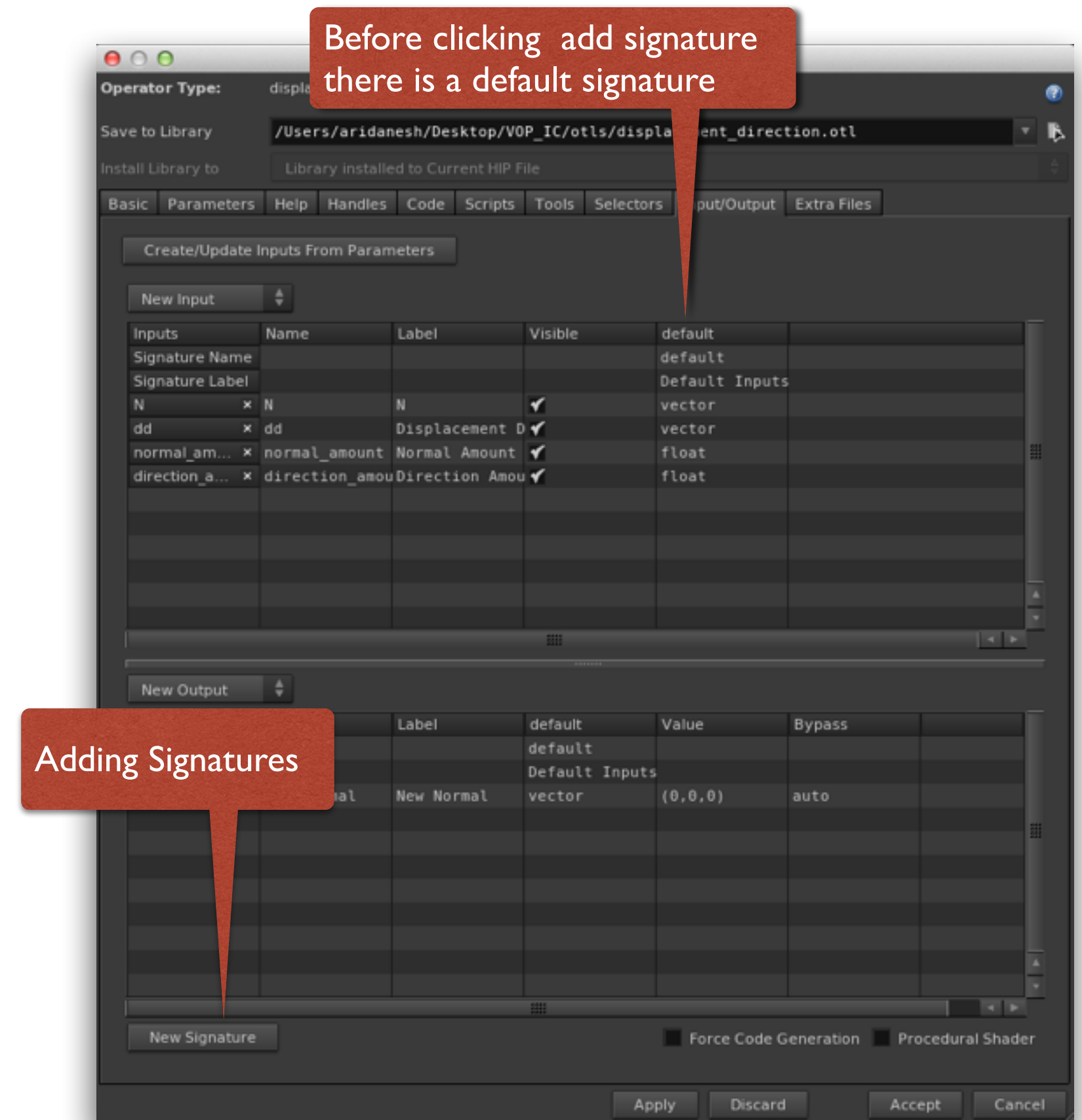
**Visible Column** - This column is present when editing a VOP Asset. It is the default value for whether inputs are visible or not, which is customized on a per-node basis by clicking on the icons next to the corresponding VOP parameters.





# Adding Signatures

A signature provides an alternate set of data types for each input and output. By default, a new VOP operator has only one signature. To create a new signature, press the New Signature button. The new signature will be created with a default name and description, and with the same data types for all inputs and outputs as the previous signature.



# Adding Signatures - A Quick Example

Let us add a noise offset to our Displacement Direction

We will want the artist to be able to wire in 1D or 3D Noise therefore we can have either two separate inputs, with two separate name and two equations or we can use signatures

In the Parameters Tab add a Vector

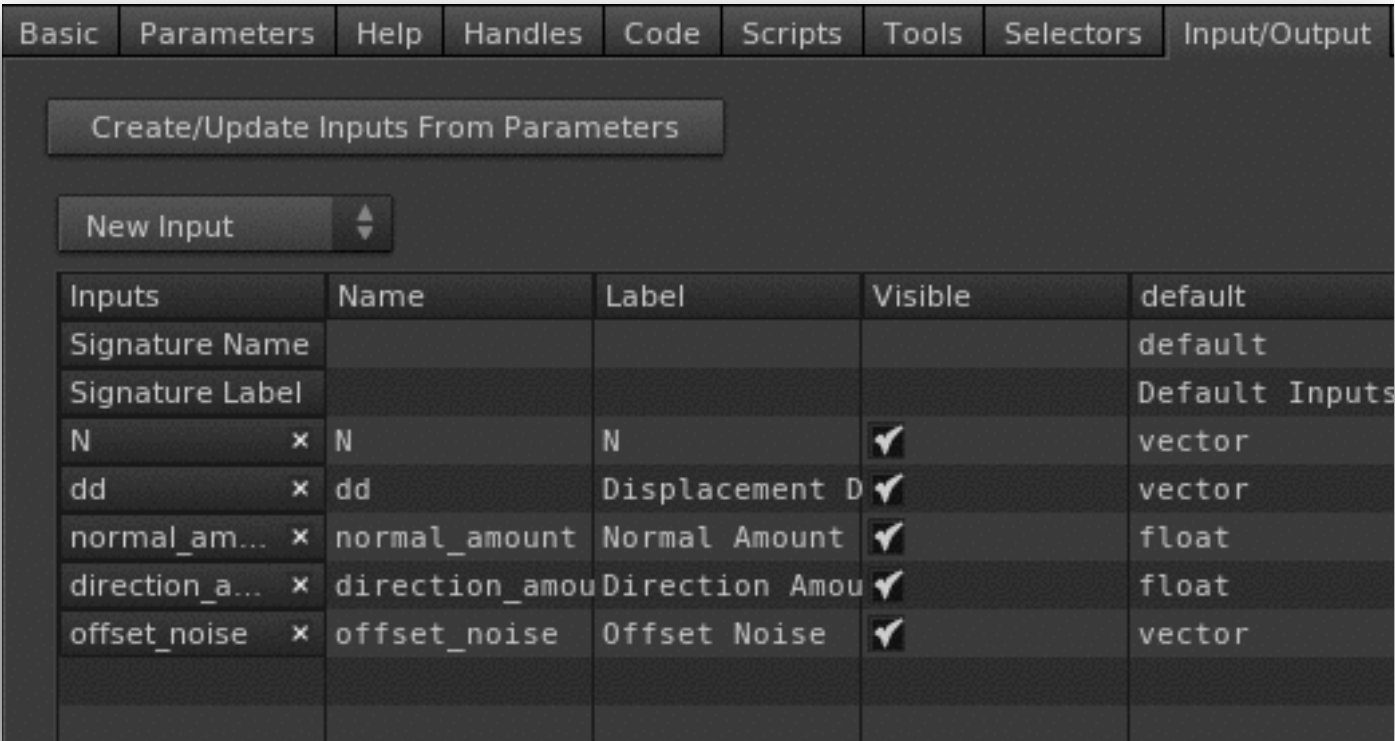
Name - offset\_noise

Label - Offset Noise

In the Input/Output tab create a vector

Name - offset\_noise

Label - Offset Noise



# Adding Signatures - A Quick Example (cont.)

Click on Add Signature

A new column appears “S1” - Signature 1

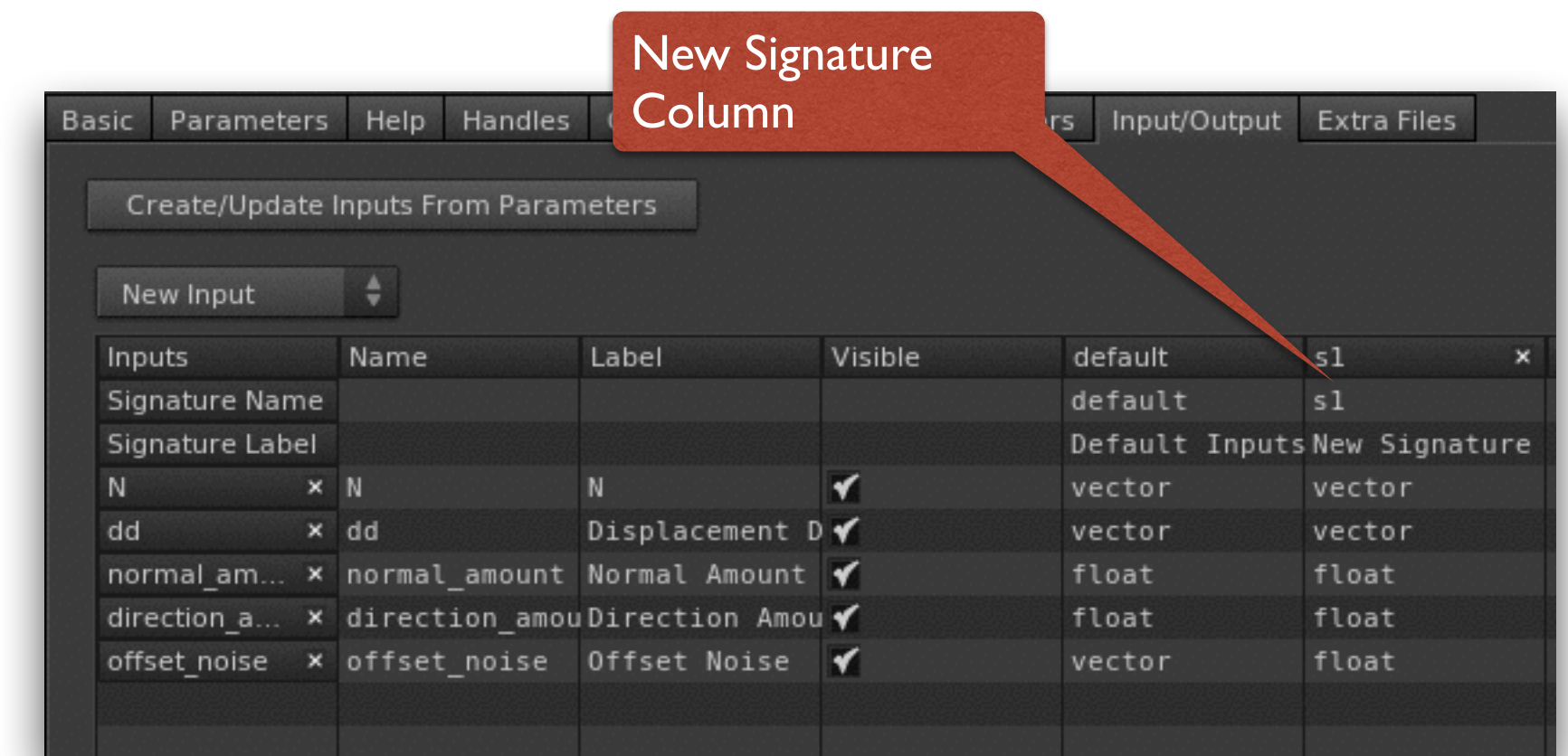
Change the value to a float

Add the additional line of code

```
$new_normal = normalize(($N * $normal_amount) + ($dd * $direction_amount));
```

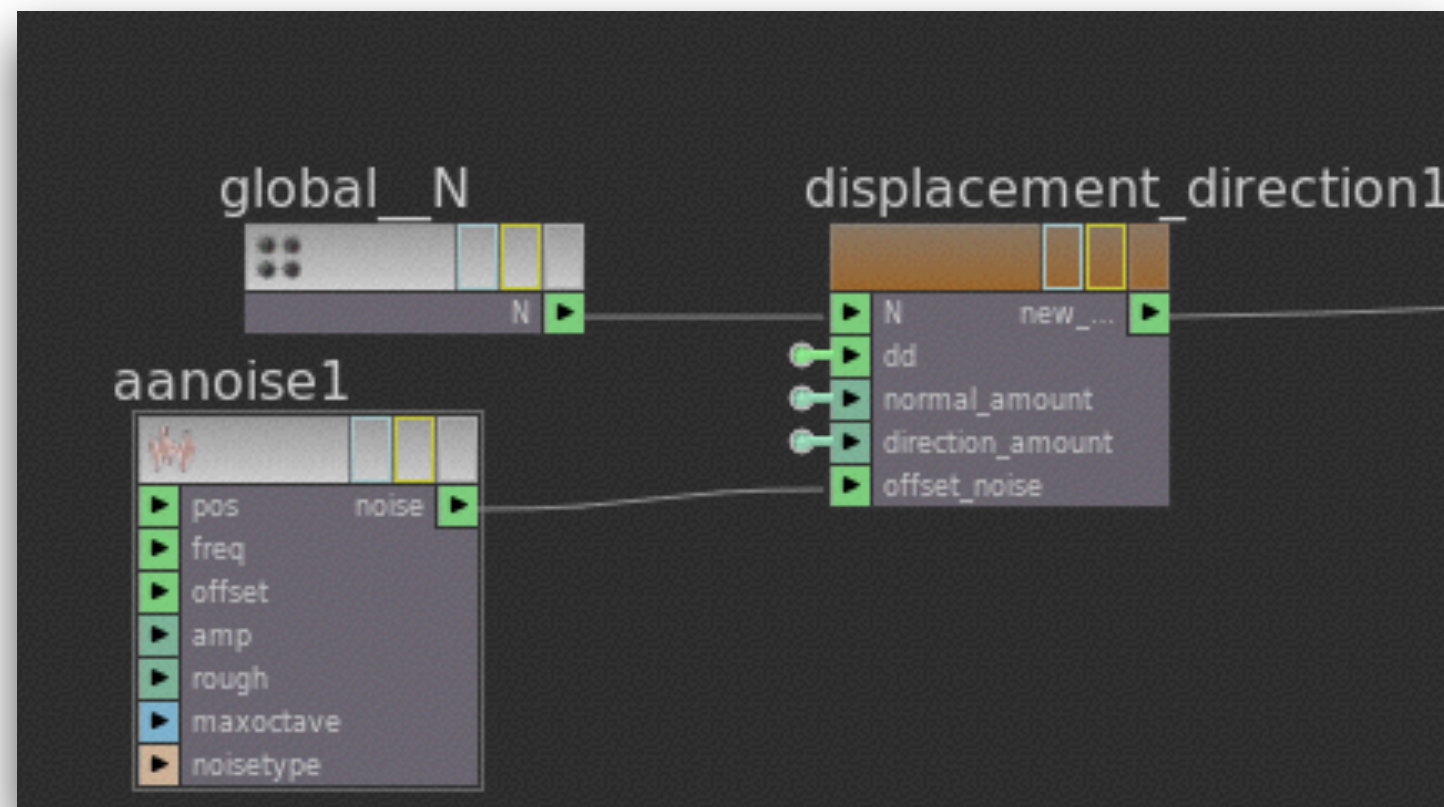
```
$new_normal += $offset_noise;
```

Click Apply/Accept





# Testing the Code



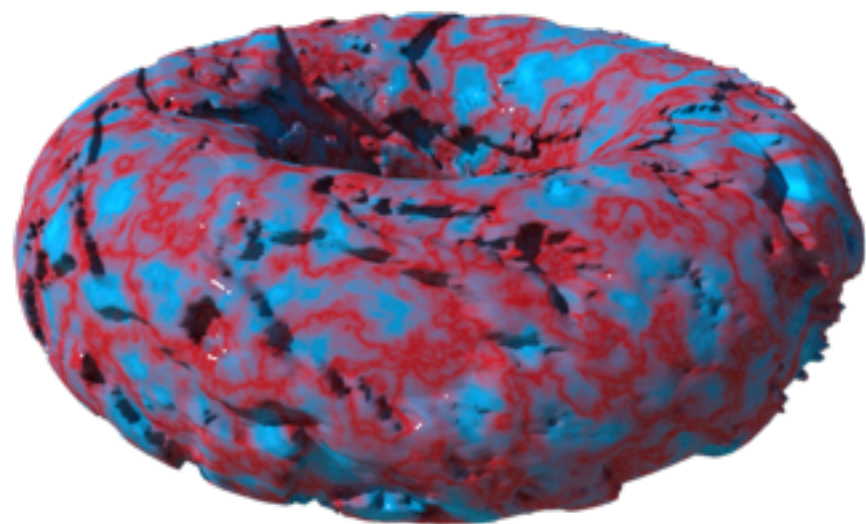
Wire in an antialias noise VOP to the offset\_noise input

Change Noise type to Simplex

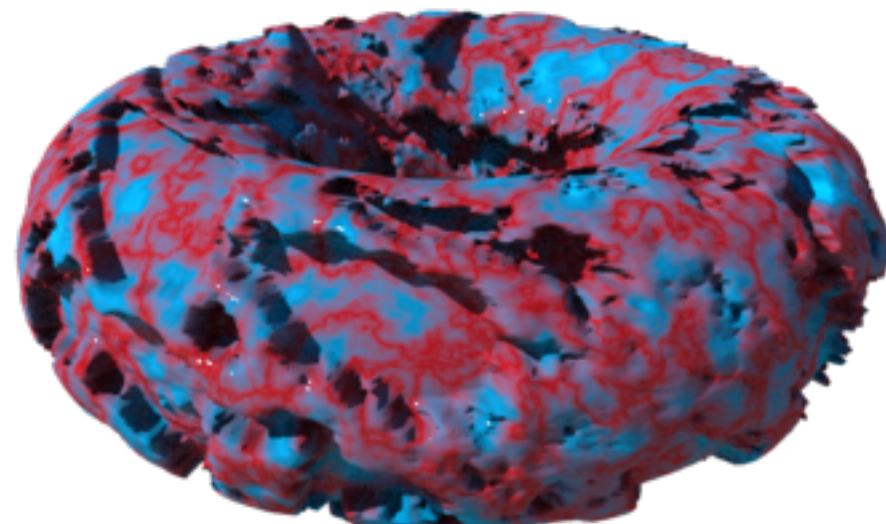
Test 3D in 1D out - See the color of the flag change

Test 3D in 3D out - See the color of the flag change

See the results of the test geometry change



1D Noise



3D Noise





# Creating a Desktop Tool in Python

# Creating a New Shelf

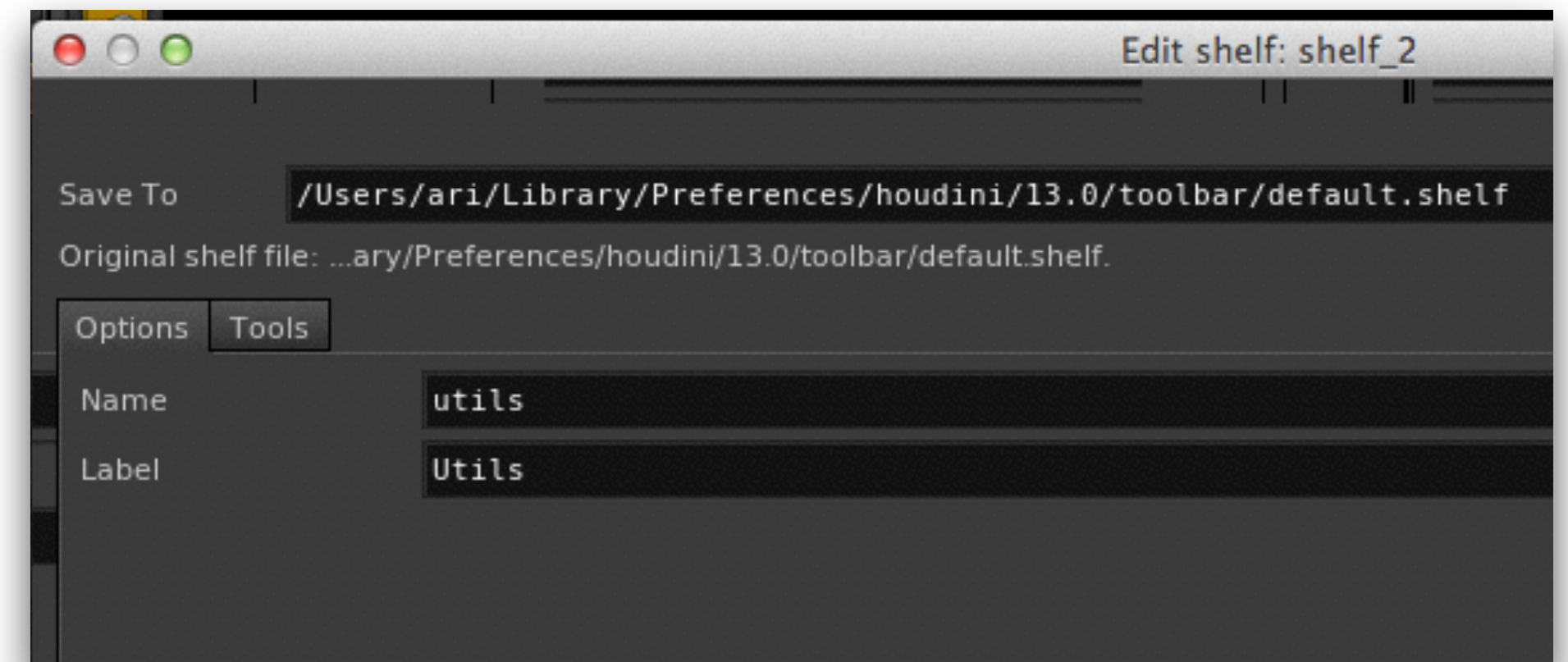
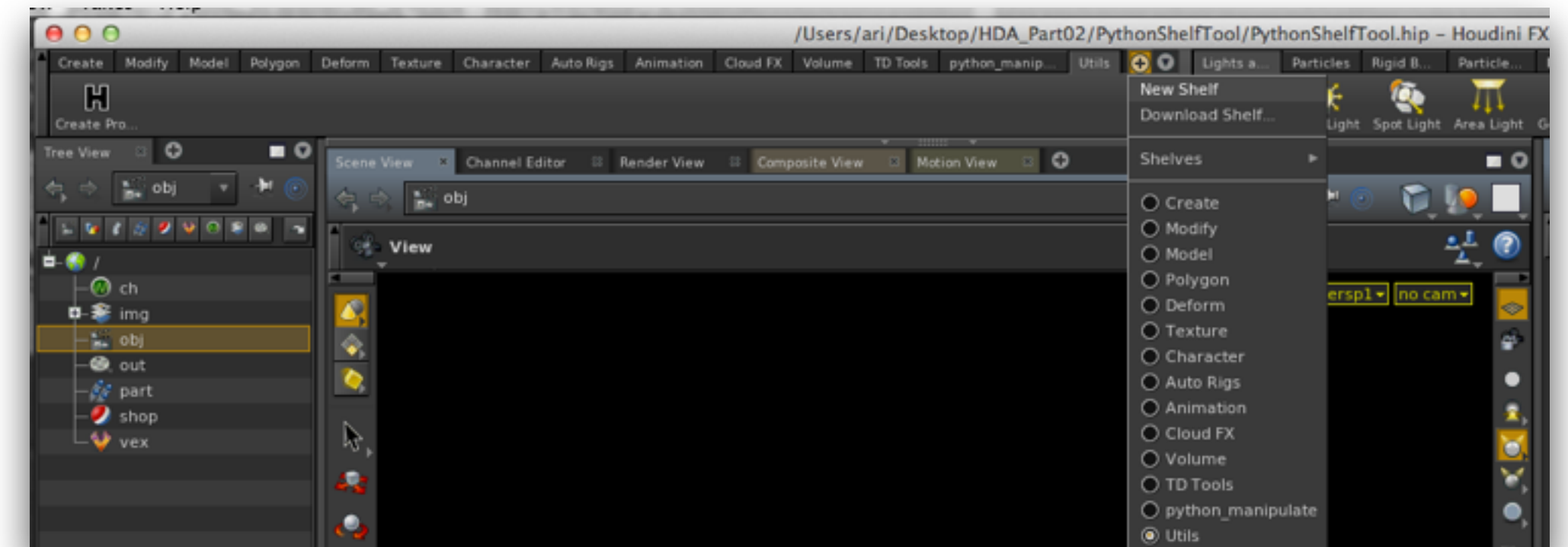
Click on the “+” button and select the New Shelf

In the Pop Up Dialog Box

Name - utils

Label - Utils

Click Accept



# Create a New Shelf Tool

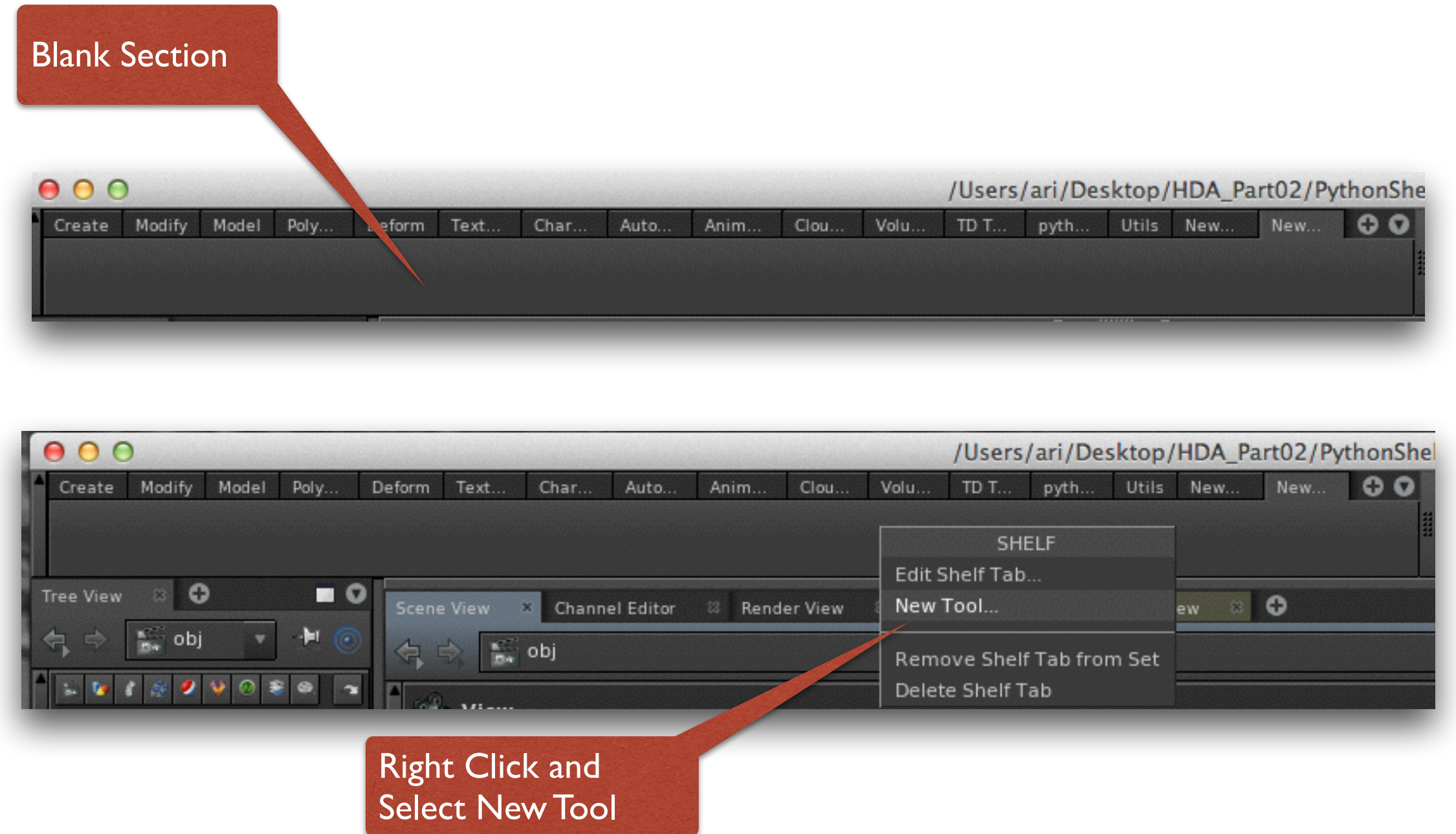
Right Click on the Blank Shelf

In the Drop Down Menu Select “New Menu”

Name - utils

Label - Utils

Click Accept

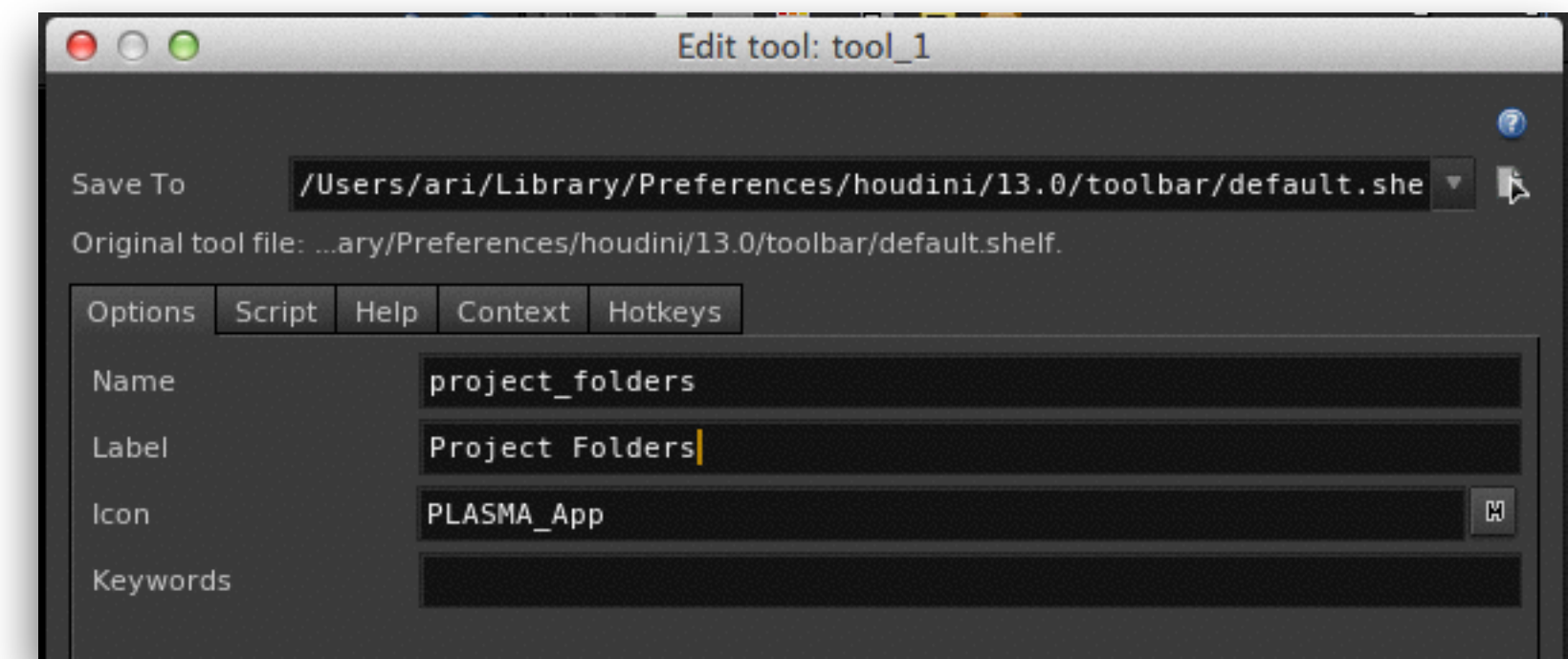


# Create a New Shelf Tool (cont.)

In the Pop Up Dialog

Name - project\_folders

Label - Project Folders





```
import hou, os
```

```
import re
```

```
import objecttoolutils
```

```
folder = hou.ui.selectFile(title = "Create Project", file_type =  
hou.fileType.Hip, chooser_mode = hou.fileChooserMode.Write)
```

```
print os.path.basename(folder)
```

```
theFile = str(folder)
```

```
oFile = hou.expandString(theFile)
```

```
hou.hipFile.save(file_name = oFile)
```

```
path = os.path.dirname( oFile )
```

```
print path
```

```
projectFolders = ("geo", "pic", "icons", "refs", "documents",  
"chops", "lsystems", "img", "audio", "desktop", "gallery",  
"presets", "scripts", "toolbar", "otls", "videos", "utilities",  
"scrap", "shell")
```

```
scriptFolderItems = ("chop", "cop2", "obj", "out", "sop")
```

```
imageFolderItems = ("hdr", "ldr", "panoramas")
```

```
for pf in projectFolders:
```

```
    fullFolderName = path + "/" + pf
```

```
    os.makedirs(fullFolderName)
```

```
scriptFolderPath = path + "/scripts"
```

```
for sf in scriptFolderItems:
```

```
    fullFolderName = scriptFolderPath + "/" + sf
```

```
    os.makedirs(fullFolderName)
```

```
imageFolderPath = path + "/img"
```

```
for imf in imageFolderItems:
```

```
    fullFolderName = imageFolderPath + "/" + imf
```

```
    os.makedirs(fullFolderName)
```

```
theBox = hou.node('/obj').createNode('geo')
```

```
theBox.setName("My_Model")
```

```
theBox.node('file1').destroy()
```

```
b = theBox.createNode('box')
```

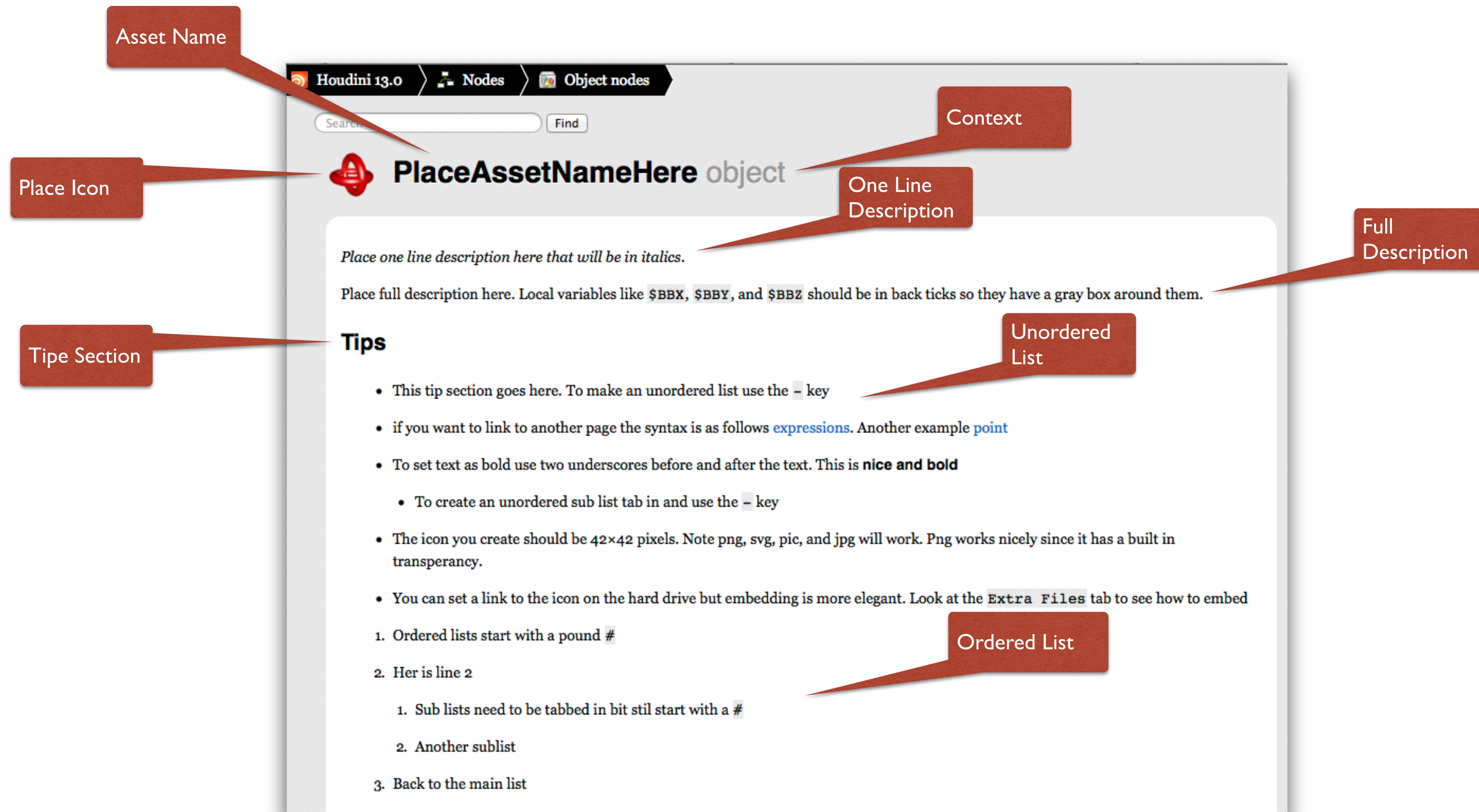
# The Code



# Help Cards Part 2

Or... A redo of the first attempt to explain

# Help Card Template (Part 01)



# Help Card Template (Part 01)

```
#type: node
#context: obj
#largeicon: opdef:Object/aridanesh_testdoc?testDocIcon.png
#tags: points, attrs, core, tech
= PlaceAssetNameHere =
"""Place one line description here that will be in italics."""
Place full description here. Local variables like ` $BBX `, ` $BBY `, and ` $BBZ ` should be in back ticks so they have a gray box around them.
== Tips ==
- This tip section goes here. To make an unordered list use the ` - ` key

- if you want to link to another page the syntax is as follows  [expressions|/expressions]. Another example
  [point|Exp:point]

- To set text as bold use two underscores before and after the text. This is  __nice and bold__
  - To create an unordered sub list tab in and use the ` - ` key

- The icon you create should be 42x42 pixels. Note png, svg, pic, and jpg will work. Png works nicely since it has a built in transperancy.

- You can set a link to the icon on the hard drive but embedding is more elegant. Look at the `Extra Files` tab to see how to embed

# Ordered lists start with a pound `#`
# Her is line 2
  # Sub lists need to be tabbed in bit stil start with a `#`
  # Another sublist
# Back to the main list
```



# Help Card Template (Part 02)

Parameter  
Section

## Parameters

<b>Parameter 1</b>	Place description here
<b>Parameter 2</b>	Place description here

Stylized  
Notes

### Note

This is how you make a note.

### Note

When creating parameters note that the number of tabs determines the sub levels.

More  
Parameters

## Another Tab of Parameters

<b>Position</b>	1. To make a comment use the # sign at the beginning of the line XYZ position.
<b>Weight</b>	Spline weight of the point.
<b>Color</b>	Diffuse color (RGB).
<b>Alpha</b>	Transparency value.
<b>Normal</b>	Normal vector.
<b>Texture</b>	Texture coordinates.

# Help Card Template (Part 02)

@parameters

Parameter 1:

Place description here

When there are two inputs

Parameter 2:

Place description here

Which attribute

Parameter 2B:

Notice tabbing controls sub levels.

NOTE:

This is how you make a note.

NOTE:

When creating parameters note that the number of tabs determines the sub levels.

== Another Tab of Parameters ==

Position:

# To make a comment use the # sign at the begining of the line

XYZ position.

Weight:

#channels: /weight

Spline weight of the point.

Color:

#channels: /diffr /diffg /diffb

Diffuse color (RGB).

# Help Card Template (Part 03)

Declaring Local Variables

## Local variables

<b>PT</b>	Point number.
<b>NPT</b>	Total number of points.
<b>CEX, CEY, CEZ</b>	Centroid of the input geometry.
<b>TX, TY, TZ</b>	Point position.
<b>PSCALE</b>	Particle Scale
<b>PT<sub><i>n</i></sub>, NPT<sub><i>n</i></sub></b>	Append <i>n</i> for the second source.

# Help Card Template (Part 03)

@locals

PT:

Point number.

NPT:

Total number of points.

CEX, CEY, CEZ:

Centroid of the input geometry.

TX, TY, TZ:

Point position.

PSCALE:

Particle Scale

PT<<n>>, NPT<<n>>:

Append <<n>> for the second source.

@related

- [Node:sop/vertex]
- [Node:sop/primitive]
- [Node:sop/xform]





# End Part 04

Digital Assets

**SIDE EFFECTS  
SOFTWARE**