OX HOM Framework

Release [0.0.1]

Justin Crapse

Aug 13, 2023

CONTENTS:

The OX HOM Framework is a light-weight Python Object Oriented Framework build on top of Houdini's HOM API. This framework makes it easier to organize and write code for Houdini's hou module (HOM.) A few highlighted features include:

- Easy node connections by input and output labels (instead of index values)
- Easy adding and deleting parm templates from nodes
- Setting menu parm values by label instead of by index (with auto complete)
- Search for child nodes by substring or regex (returns a list of nodes)
- Handling saving and loading of presets
- Simple logging system for helpful debugging (no print statements needed!)
  - Easily change logging level
- Easily extendible framework.
- Extensive auto-complete features:
  - Input and output labels for any given node type
  - parameter name values
  - node types organized by network context
- And many more helpful functions and improved workflows

CHAPTER

ONE

INSTALLATION

Steps:

1. Dowload the framework
    - Github: https://github.com/justincrapse/ox_hom_framework
    - Or by direct link: OX Framework Download Link
2. Install As Plugin and Verify

    Plugin Installation follows the standard installation steps for any Houdini Plugin. Please see video below.

CHAPTER

TWO

GETTING STARTED

2.1 Getting Started Video Tutorial

2.2 Working With Nodes

The OX Framework uses a basic class structure to work with nodes. Each node within houdini has a class associated with it within the framework. These node classes are generated from the nodes themselves. Any node can easily be added to the frameowrk or updated for new versions of Houdini.

Here is an example of working with nodes in the framework:

```
import hou

from ox import nodes, OXNode


obj_ox_node = OXNode(hou.node('/obj'))

geo_ox_node = nodes.obj_nodes.GeoNode(ox_parent=obj_ox_node, node_name='my_geo_node')
box_node = nodes.geo_nodes.BoxNode(ox_parent=geo_ox_node, node_name='my_box_node')
```

Instead of creating a node from a HOM node's .createNode() method and passing in the name of a node, we can leverage autocomplete to find the node we want to create and simply pass the parent OX node in as a parameter. The node classes reside within their node context as seen above. With autocomplete, you can easily see all the contexts (obj_nodes, geo_nodes, redshift_nodes etc.) as well as all the possible child nodes for those contexts.

To get a framework node for an existing node, you can use a number of methods to find the node from the parent and pass it in to the node class:

```
import hou

from ox import nodes, OXNode


obj_ox_node = OXNode(hou.node("/obj"))

my_geo_node = obj_ox_node.get_child_node_by_name(child_name='my_geo_node')
geo_ox_node = nodes.obj_nodes.GeoNode(my_geo_node)
```

Note that in the example above, I did not pass in a keyward argument into the GeoNode class. This is because the first positional argument is a hou node. Because this a common and intiutive way to pass in this node, it is okay to break the regular rules of always passing in keyword arguments when calling methods and instantiating classes.

2.2.1 Best Practice For "ox_node" vs. "node"

Whenever the text "node" is expressed in the framework, it is reffering to hou.Node nodes. In the framework, as well as in all scripts written using the framework, you should prepend "ox_" to the term "node" that occurs in any variable refering to nodes to distiguish it from hou.Node nodes. See script below for an example.

2.3 Parameters

All parameters exist as class members starting with "parm_" and ending with the name of the parameter. You may need to go into houdini and hover over a parm label to see its name (which can be quite different from the label,) or just type my_ox_node.parm_ and let autocorrect give you all the parameter options for that node.

Instead of:

```
node.parm('some_parm_name').set(some_value)
```

You can do:

```
som_ox_node.parm_<some_parm_name> = some_value
```

2.4 Menu Parameters

Menu parameters can be a pain to set using the HOM. Some menus use integer numbers and some use string values that map to the menu option labels. Instead of dealing with any of that, we can simple access the menu parameter label to set it like so:

```
some_ox_node.parm_some_menu_parameter.menu_<some_menu_value_label>
```

The code above will figure out what value to set that menu parameter. Just that one line of code will set it to the right menu value for that label. This is not a conventional way to modify an attribute, but it is incredibly simple and works well. See the code below for a real example of this in action.

Here is a simple code snippet to illustrate the basic workflow for working with Nodes:

```
import hou
from ox import OXNode
from ox import nodes
from ox.helpers import ox_helperc

obj_node = hou.node("/obj")
obj_ox_node = OXNode(node=obj_node)

geo_ox_node = nodes.obj_nodes.GeoNode(ox_parent=obj_ox_node, node_name="my_geo")
cube_ox_node = nodes.geo_nodes.BoxNode(ox_parent=geo_ox_node, node_name="my_cube")
cube_ox_node.parm_scale = 2

cube_trans_ox_node = nodes.geo_nodes.TransformNode(ox_parent=geo_ox_node, node_name=
↪"cube_trans")
cube_trans_ox_node.connect_from(cube_ox_node)

cube_normal_ox_node = nodes.geo_nodes.NormalNode(ox_parent=geo_ox_node, node_name="cube_
```

(continues on next page)

(continued from previous page)

```
↪norm")
cube_normal_ox_node.connect_from(cube_trans_ox_node)

cube_uv_ox_node = nodes.geo_nodes.UvtextureNode(ox_parent=geo_ox_node, node_name="cube_uv
↪")
cube_uv_ox_node.connect_from(cube_normal_ox_node)
cube_uv_ox_node.parm_type.menu_face
cube_uv_ox_node.parm_sv = cube_uv_ox_node.parm_su.parm  # this will copy by reference
cube_uv_ox_node.parm_sw = cube_uv_ox_node.parm_su.parm  # this will copy by reference
```

2.5 The OXNode Class

The "OXNode" class:

```
from ox import OXNode
```

This OXNode class contains the common methods for most nodes. All node classes inherit from OXNode.

The OXNode class inherits from the "ParmTemplate" base_objects class as a mix-in. Mix-ins are an uncommon Python inheritance pattern best avoided. In this case, it serves as a way to organize the parm template code into its own document as to not convolute the OXNode namespace.

When automating scripts, you won't always know what type of node you are dealing with, but you'll still want the functionality of the framework. In these cases, you can simply use the OXNode class directly:

```
from ox import OXNode

connected_node = some_ox_node.get_connected_output_node_by_index(index=0)
connected_ox_node = OXNode(node=connected_node)

connected_ox_node.run_some_oxnode_function()
```

Note that I passed in "connected_node" as a keyward agrument. While this is the general rule to live by, the "node" keyword can be omitted as it is A common access pattern that will not change as the first parameter arg.

2.6 The OX:Admin toolbar

The administrative toolbar "OX:Admin" contains a couple of important node class generator tools and a sandbox tool See "Adding Node Classes" for more information.

2.7 Logging/Debugging

The framework uses a simple Python logging configuration that greatly helps debugging efforts as the Python framework is only loaded at Houdini Startup (so you cannot add print statements without restarted the software to see the output.)

To change the logging level for your session, type in the following into a Python terminal in houdini:

```
import ox

ox.set_logging_level(level=10)
```

This will let the logging level to "10," which is the debug level.

To set up the logger in your script, write the following code (with use case examples):

```
import logging

ox_logger = logging.getLogger("ox_logger")


# now the loger is ready to use
ox_logger.debug('some debug message')
ox_logger.info('some info message')
```

If you need a Python logging primer, Real Python has a great page here: https://realpython.com/python-logging/

CHAPTER

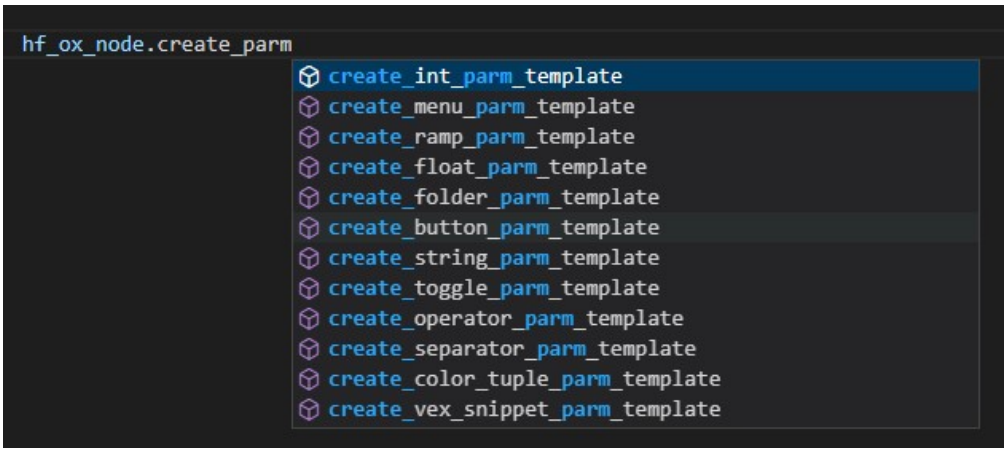
THREE

WORKING WITH PARM TEMPLATES

3.1 Creating Parm Templates

Adding a new parameter to a node can be a complicated matter depending on what you are trying to do. This is mainly a two-step process of 1. creating the parm template and 2. Adding that parm template to the node you want (and in the right place.)

All OX nodes have direct access to creating (almost) any type of parm template you would need. Adding the remaining parm templates is a matter of priority, but should not take long. The reason we are including these in the framework is for autocompletion and to have a much simpler way to add the parm templates where you want them.



Each of the parm template create methods returns a parm template. The reason we don't add the parm template in the same method is that the logic can be convoluted and would require a multitude of parameters to be repeated per create method.

3.2 Adding Parm Templates

To add a parm template, simply call the add_parm_template method like so:

```
new_parm_template = some_ox_node.create_int_parm_template(name='some_parm')
some_ox_node.add_parm_template(parm_template=new_parm_template, insert_after_parm='<name_
↪of_parm_to_inser_after | parm>')
```

Keep in mind there are more strategies available to inserting parm templates. See the method signature for more methods.

3.3 Removing Parm Templates

Removing parm templates is simple:

```
some_ox_node.remove_parm_template_by_name(name='some_parm_name')
```

Folders are more difficult as their names are not controlled by our code. In this case, use the following:

```
some_ox_node.remove_folder_by_label(label='My Folder Label')
```

Subfolders are a special case. You'll need to provide the parent and child folder labels.

```
some_ox_node.remove_subfolder_by_labels(parent_folder_label='My Parent Folder Label',␣
↪folder_label='My Folder Label')
```

CHAPTER

FOUR

OX NODE CLASS

This page includes the class method documenatation for the OXNode class and ParmTemplate mix-in class.

class ox.base_objects.ox_node.**OXNode**(*node=None*)

apply_child_parms_dict(*children_parms_dict*)

Applies a dictionary of child nodes and their parameter values as returned from get_child_parms_dict method

apply_parms_dict(*parms_dict*)

Applies the values from a parm dictionary returned by get_parms_as_dict

child(*child_name*)

shorthand method for get_child_node_by_name

connect_from(*ox_node=None*, *input_index=0*, *out_index=0*, *input_label=None*)

Connects to this node's input from another ox node's output. use input_label over input_index whenever possible.

copy_node(*new_name_postfix=None*, *destination_node: Optional[hou.Node] = None*, *delete_if_exists=False*, *keep_existing_parm_values=False*, *return_ox=True*, *keep_existing_children_parm_values=False*)

Copies a node to a destination with optional behaviors

create_node(*node_type_name*, *node_name=None*) → hou.Node

Creates a child node. It's better to create nodes using the 'nodes' package.

create_node_if_not_exists(*ox_node_class=None*, *ox_node_type=None*, *node_name=None*) → hou.Node

Creates a child node, but only if it does not already exist. need to pass in either ox_node_class or ox_node_type str

create_ox_node_if_not_exists(*ox_node_class*, *node_name=None*)

shortcut method that will go ahead and create and return an ox node

delete_all_child_nodes()

delete all child nodes. Keep in mind this might not delete everything within a node network. See "delete_all_items" method

delete_all_items()

deletes all items within the node

delete_child_node_by_name(*child_name*, *expect_child_node=True*)

Deletes a child node matching the child_name. will raise an exception if no child is found and excpect_child_node is True

Parameters
expect_child_node – this is another parm

delete_node()

A simple method to delete the current node. Also see destroy_node method

delete_parms_by_name(*parm_name_list: List[str]*)

Deletes all parameters by list of parm names

delete_preset(*node_path*, *preset_name*, *preset_path=None*)

Deletes a preset. uses default preset path when preset_path not specified

destroy_node()

A simple method to delete the current node. Also see delete_node method

get_child_by_node_type(*node_type*, *substring=None*, *expect_match=False*, *only_one_match=True*)

WARNING: TO BE DEPRICATED. this is redundant and poorly named. use get_child_node_by_type instead

get_child_node_by_name(*child_name*) → hou.Node

Returns a matching child node, if it exists, else None is returned

get_child_node_by_partial_name(*substring*, *exclude_substring=None*, *case_sensitive=True*) → hou.Node

Searches for a child node by substring. Returns the first match.

get_child_node_by_type(*node_type*, *substring=None*, *expect_match=False*)

Returns the first matching child node of node_type parameter

get_child_node_paths_by_partial_name(*substring*) → List[str]

Returns child node paths based on child nodes matched to the substring

get_child_nodes() → List[hou.Node]

Returns all child nodes

get_child_nodes_by_partial_name(*substring*, *exclude_substring=None*, *case_sensitive=True*) → List[hou.Node]

Returns a list of child nodes matched by substring

get_child_nodes_by_regex(*regex_str*)

Returns all nodes with names that match the regular expression

get_child_nodes_by_type(*node_type*, *substring=None*, *expect_match=False*)

Returns all child nodes mathcing specified type.

get_child_ox_node_by_name(*child_name*, *node_class=None*)

Returns a child ox node matching the child_hame parameter.

get_child_ox_node_by_type(*node_class*, *expect_match=False*)

Returns the first matching child node as ox node of node_type parameter

get_child_parms_dict(*node_list: Optional[List[hou.Node]] = None*, *exclude_type_list: Optional[List[str]] = None*)

exclude_type_list: a list of types (from node.type() values) to exclude from the parms dict handy method that gets all children nodes and parameters as a dict to reaply later or to another node.

get_connected_input_node_by_index(*index=0*) → hou.Node

Returns the node connected to the specified input index

get_connected_output_node_by_index(*index=0*) → hou.Node

Returns the node connected to the specified output index

get_displayed_child_node() → hou.Node

Returns displayed child node.

get_folder_labels()

Returns top-level folder labels.

get_input_connections_count()

returns number of connected input connections

get_input_connections_node_name_list()

Returns a list of node names of the connected input nodes

get_input_label_index(*label*)

Given an input label, returns the input index value

get_input_labels()

Gets all input labels for the node

get_parent_network_box()

Returns the parent Network box this node is contained in.

get_parm_labels()

Returns all parm labels

get_parm_names()

Returns all parm names

get_parm_names_by_substring(*substring*)

Returns all parm names that match the substring

get_parms() → List[hou.Parm]

Gets all parms

get_parms_as_dict(*substring=None*, *as_raw_value=False*)

returns all parameters as a dictionary with the parm name as key and parm value as the value. If a substring is specified, this will only match parameter names with that substring

get_parms_by_name_substring(*substring*, *ends_with=False*, *starts_with=False*) → List[hou.parm]

Returns all parms that match the specified substring

get_parms_by_regex(*regex_str*)

Returns all parms that match the regular expression

get_planes()

Returns planes. Heightfield Masks are held as planes. Not sure what else this is fore.

get_prim_groups()

Returns the primative group names

get_prim_int_values(*prim_name*)

Returns primative integer values by prim name

get_prim_values_by_field(*field*, *filter_out_blank_values=True*)

Returns primative values by specified field

has_child_with_name(*child_name*) → bool

Checks for child match by exact name

has_child_with_node_type(*node_type*, *expect_match=False*) → bool

Checks for a child match by node type

is_display_flag_set()

Checks to see if display flag is set

layout_children()

Auto-layout children

load_preset(*preset_name=None*)

Loads a preset. if no preset name specified, just use the node name. This is a good default for many use cases

move_node_relative_to(*ox_node*, *x=0*, *y=-1*, *unit_multiplier=1*)

a handy method that will move this node relative to another node. The default moves the node below the relative node

remove_parms_by_name_substring(*substring*, *ends_with=False*, *starts_with=False*)

Removes all parms by name that match a substring

rename_node(*new_name*)

Alternative to 'set_name' with more debugging if needed.

save_preset(*preset_name*, *preset_path*, *node_path=None*)

Saves a preset

select_node(*on=True*)

" Set node as selected

set_bypass_flag(*on=True*)

Sets the bypass flag

set_color(*color*)

Can pass in a tuple for RGB values or a hou.Color object.

set_display_and_render_flags(*on=True*)

Sets the display and render flags

set_display_flag(*on=True*)

Sets the display flag

set_name(*new_name*)

Sets the node name

set_render_flag(*on=True*)

Sets the render flag

unlock_node()

Allows editing of node contents.

4.1 Parm Template Class

class ox.base_objects.parm_templates.**ParmTemplate**(*node*)

add_folder(*folder_label*, *folder_name*, *as_first=False*)

Adds a folder to a node.

add_parm_template(*parm_template*, *folder_label=None*, *as_first=False*, *insert_after_parm=None*, *insert_before_parm=None*, *return_type=None*, *supress_logger=False*) → hou.Parm

Adds a parm template to a node. if folder_label is specified, as_first, insert_after_parm, and insert_before_parm are not relavant as those will dictate which folder a parm template is added to.

Parameters

- **folder_label** – When specified, the parm_template will be added to the folder by label string.
- **as_first** – Will insert this parameter as the first in the node.
- **insert_after_parm** – This can be a parm object or the name of a parm after which you insert the new parm template
- **insert_before_parm** – same as insert_after_parm but before.
- **return_type** – Return types may vary depending on the type of parm template. Add logic as neccisary. Options: * 'color' # for color parm templates
- **supress_logger** – For specific expected error logging that may intentionally be supressed

add_parm_template_if_not_exist(*parm_template*, ***kwargs*)

Only adds the parm template if it does not already exist

add_parm_template_to_sub_folder()

TODO: Need to implement this.

add_parm_template_with_override(*parm_template*, *folder_label=None*, *as_first=False*, *insert_after_parm=None*, *insert_before_parm=None*, *return_type=None*, *keep_original_value=True*) → hou.Parm

This method will delete the parm method if it already exists and can reapply the previous value to the newly created parm

create_button_parm_template(*name*, *label*, *join_with_next=False*, *script_callback=None*, *script_callback_language=hou.scriptLanguage.Python*, ***kwargs*)

Creates a button parameter template

create_color_tuple_parm_template(*name*, *label=None*, *default_value=[1, 1, 1]*, *is_label_hidden=False*, *join_with_next=False*)

creates a color tuple parameter template

create_float_parm_template(*name*, *label=None*, *num_components=1*, *default_value=()*, *min=0.0*, *max=10.0*, *min_is_strict=False*, *max_is_strict=False*, *is_label_hidden=False*, *join_with_next=False*, ***kwargs*)

Creates a float parameter template

create_folder_parm_template(*name*, *label=None*) → hou.FolderParmTemplate

Creates a folder parameter template

create_int_parm_template(*name*, *label=None*, *num_components=1*, *default_value=()*, *min=0*, *max=10*, *help=None*, *is_label_hidden=False*, *join_with_next=False*, ***kwargs*)

Creates an integer parameter template

create_menu_parm_template(*name*, *menu_items*, *label=None*, *menu_labels=()*, *is_label_hidden=False*, *join_with_next=False*, ***kwargs*)

Creates a menu parameter template

create_operator_parm_template(*name*, *label=None*, *num_components=1*, *is_label_hidden=False*, *join_with_next=False*, ***kwargs*)

Creates an operator parameter template

create_ramp_parm_template(*name*, *label*, *ramp_parm_type=hou.rampParmType.Float*, ***kwargs*)

Creates a ramp parameter template

create_separator_parm_template(*name*, ***kwargs*)

Creates a separator parameter template

create_string_parm_template(*name*, *label=None*, *num_components=1*, *multiline=False*, *join_with_next=False*, *script_callback=None*, *script_callback_language=hou.scriptLanguage.Python*, *tags=None*, *is_label_hidden=False*, *default_value=()*, ***kwargs*)

Creates a string parameter template

create_toggle_parm_template(*name*, *label=None*, *default_value=False*, *is_label_hidden=False*, *join_with_next=False*, ***kwargs*)

Creates a toggle parameter template

create_vex_snippet_parm_template(*name*, *label=None*, ***kwargs*) → hou.StringParmTemplate

This is pretty much just the attribute wrangle vex parameter .asCode() I have not been able to get this to work properly. TODO: figure out how to implement this correctly

delete_folder(*folder_label*)

Deletes a top-level folder

get_entry_labels()

Retruns a list of entry labels. TODO: explain what an entry is

get_folder_name_by_label(*label*)

Returns the folder name by label. Note that folder names are auto-generated and may be unexpected values that don't line up the way you think they would

get_folder_parm_labels(*folder_label*)

Returns the parm labels of a folder

get_folder_parm_templates_by_label(*folder_label*)

Returns the folder parm templates given the folder label string

get_folder_parms_by_label(*folder_label*) → List[hou.Parm]

Returns parms of a folder

get_folder_template(*folder_label*)

Returns as folder's template by folder label

get_parm_template_by_label(*label*)

Returns a parm template by label

get_parm_template_names_by_substring(*substring*)

Returns parm template names matched by specified substring

get_parm_templates_by_type(*template_type*)

Returns list of parm templates by type. template types can be found in ox.constants.parm_template_types

get_sub_folder_parm_value_dict_by_folder_labels(*parent_folder_label*, *folder_label*) → dict

Returns a dictionary of parm names and values for the subfolder

remove_folder_by_label(*label*)

Removes a folder given the label string

remove_parm_template_by_name(*parm_name*, *save_template_group=True*)

removes a parm template by name

Parameters

save_template_group – This parameter lets you hold off on saving the template group, which may behave better when removing many parms

remove_subfolder_by_labels(*parent_folder_label*, *folder_label*)

Removes a subfolder by parent and subfolder labels

CHAPTER

FIVE

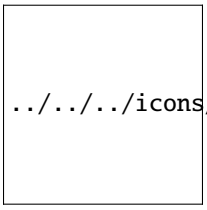
ADDING NODES TO THE FRAMEWORK

The node classes in the framework are generated using admin tools that gather information from a live Houdini session. Not all nodes have been added, but it is very easy and a mostly painless and automated process for 99% of the nodes. Because houdini's node parameters and options change over time with updated versions of Houdini, the framework can also re-run all existing node classes at once (See admin toolbar.)

5.1 Adding A New Node Class To The Framework

Before considering adding a new node to the framework, you'll want to make sure it has not already been added. Node class names use the default node name when the node is created, minus and trailing digits. The node type, however, is taken from hom_node.type().name() and is a class-level variable as "node_type" for every generated node class.

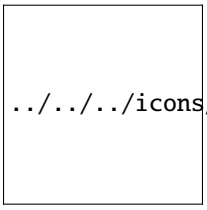
Once you have determined that the node class does not already exist, create a new, clean node of the type you want to add, select it, then click on the "Node Class Generator" tool in the "OX:Admin" toolbar. Shoutout to our cat Zero who possed for the icon photo:



If there are dynamic parameters that only show up after adjusting parameters or imputs, we will want to add these to the "Regenerate Node Classes" tool. Please contact the repository owner if planning to create a pull request for this type of scenario.

5.2 Updating Existing Node Classes

The "Regenerate Node Classes" tool in the "OX:Admin" toolbar:



This should not be ran in general. This will update all node classes currently in the framework. This should idealy only be ran after a major release and merged into the main branch shortly after.

5.3 Handling Dynamic Parameters

Some parameters will not show up in the auto-generated node class if that parameters is not preset by default. To work around this, we can add a node type in the appropriate section of the "Regenerate Node Classes" tool in the "OX:Admin" toolbar (explained above.) For example, in the geo_nodes section, the script is checking for a node of type "labs::hf_combine_masks". Before the node class is generated, the parameter "folder0" is given a value of 10, which dynamically generates parameters that we will likely need to access via the node class. Otherwise, we would not be able to see that parm from our node class.

Of course you can always call some_ox_node.node.parm('layername_1') to access the parameter, but the whole point of the framework is to keep hardcoding to a minimum. Anytime we can code something once, such as in this workaround, it is the preferred standard when adding to this framework.

CHAPTER

SIX

APPENDIX A: CODING ENVIRONMENT

6.1 VSCode IDE Setup

6.1.1 Black Installation and Configuration:

1. Install and configure Black Formatter: https://dev.to/adamlombard/how-to-use-the-black-python-code-formatter-in-vscode-3lo0
2. Set the line-length (This framework uses 150): https://dev.to/adamlombard/vscode-setting-line-lengths-in-the-black-python-code-formatter-1g62
3. Install Black for hython. You'll need to do this for auto-formatting to work when editing HDA scripts. See this video for how to install 3rd party libraries to hython: https://www.youtube.com/watch?v=cIEN50WuPoc

6.1.2 Configure the hou and ox Module for Auto-Complete

Add the following code to your settings.json file (I recommend doing this at the user level.) Don't know how to do that? Check this stack overflow: https://stackoverflow.com/questions/65908987/how-can-i-open-visual-studio-codes-settings-json-file

```
{
    "python.autoComplete.extraPaths": [
        "C:/Program Files/Side Effects Software/Houdini 19.5.303/houdini/python3.9libs
↪",
        "C:/Users/justi/source/repos/ox_framework/python3.9libs",
    ],
    "python.analysis.extraPaths": [
        "C:/Program Files/Side Effects Software/Houdini 19.5.303/houdini/python3.9libs
↪",
        "C:/Users/justi/source/repos/ox_framework/python3.9libs"
    ],
    "python.autoComplete.preloadModules": [
        "hou",
        "ox"
    ]
}
```

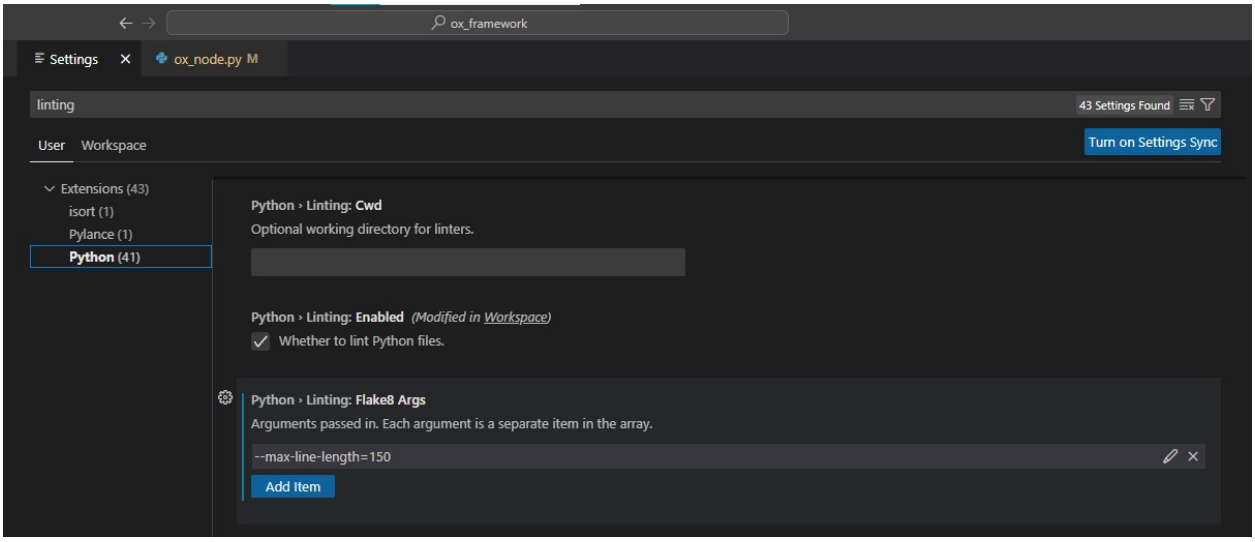
Make sure you adjust the paths above to wherever your Houdini installation lives and also the python3.xlibs path for the ox framework. Note: It was difficult to figure out how to get this to work and I couldn't find how from a single source. This may or may not work for you, but if it does not, search the internet for "houdini hou autocomplete VSCode" or equivalent.

6.1.3 Flake8 Linter

Python linting in VSCode: https://code.visualstudio.com/docs/python/linting

You may use any linter you'd like, but if you are going to contribute to the framework, I will be enforcing a clean result from Flake8. The Setup can be found in the root of the directory as "setup.cfg"

When modifying toolbar scripts, the setup.cfg file may be ignored as it was for me. In this case, you can add command line arguments for flake8 in the settings. Go to File>Preferences>Settings and type in linting. Then click on "Python" on the left under "Extensions" as seen in the image below. Add as many options as you need. For now this example just has –max-line-length=150.



The nodes folders are excluded for now, but most of that is auto-generated and will not be modified directly.

6.2 PyCharm IDE Setup

PyCharm is a great IDE. I never figured out how to get the hou module to work with it, unfortunately. Please consider contributing if you have been able to get this to work with Pycharm.

6.3 Learning Resources

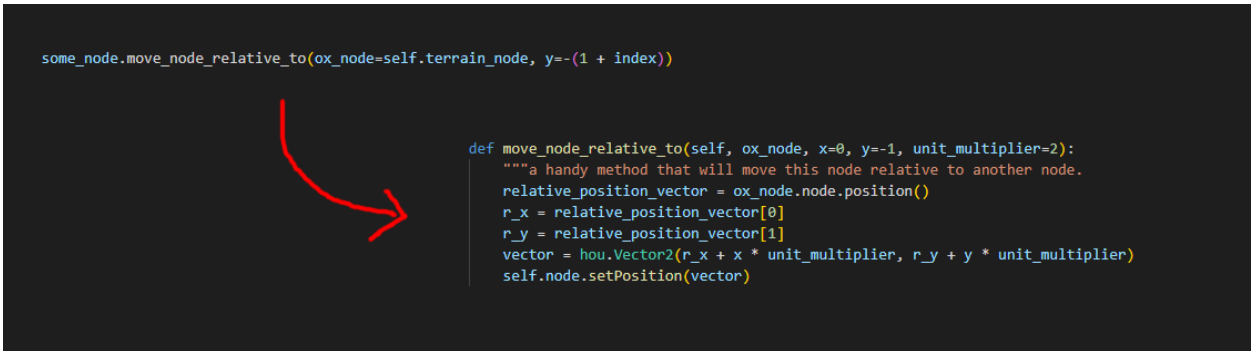
- Python Logging: https://realpython.com/python-logging/

6.3.1 Houdini HOM

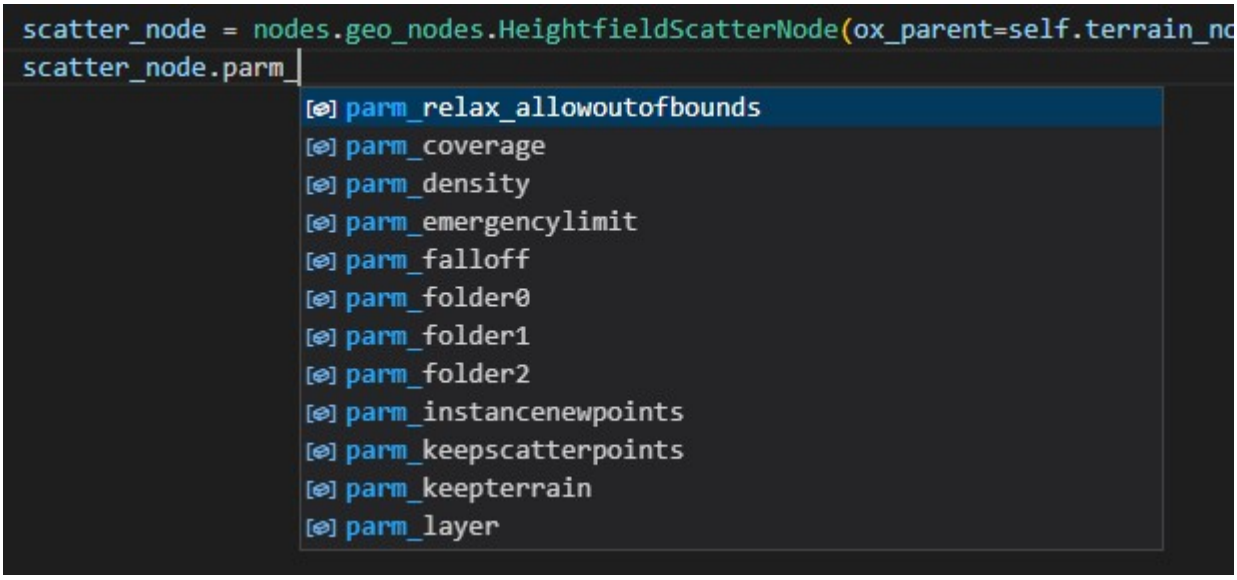
SideFX HOM Documentation: https://www.sidefx.com/docs/houdini/hom/intro.html

Recommended beginner Houdini Automation Tutorial: Intro to Python in Houdini by Socratica FX:

The following is an example of how we can call a method on any node that can leverage the HOM under the hood.



Another main feature of the framework is providing quality-of-life coding environament capabilities such as auto-complete for all nodes and parameters



This OOP class structure is posible through automatic class generation and import registration. This requires very little additional setup, if any, per node.

INDEX

L

M

O

P

R

S

U