Fun Shelf Scripts

Version up sops that have a "version" parm - I use this to version up a lot of exports in tops or rops:

```
import hou
nodes = hou.selectedNodes()
for x in nodes:
	ver = x.parm("version").eval() + 1
	x.parm("version").set(ver)
```

Send a ROP fetch to TOPs name "topnet1" for convenient cooking.:

```
import hou
nodes = hou.selectedNodes()
topnet = hou.node("/obj/topnet1")
frb = round((hou.playbar.frameRange()[1] - hou.playbar.frameRange()[0]) / 2)

if not topnet:
    topnet = hou.node("/obj").createNode("topnet")
    topnet.setName("topnet1")
if not hou.node("/obj/topnet1/rangegenerate_start"):
    rg = topnet.createNode("rangegenerate")
    rg.setName("rangegenerate_start")
for x in nodes:
    ver = x.parm("ver")
    path = x.path()
    name = x.name()
    if x.parm("extra"):
        name = name + x.parm("extra").eval()
    else: pass
```

```
    topcache = topnet.createNode("ropfetch")
    p = hou.IntParmTemplate("version", "Version", 1)
    g = topcache.parmTemplateGroup()
    g.append(p)
    topcache.setParmTemplateGroup(g)
    topcache.parm("version").set(ver)
    topcache.parm("roppath").set(path)
    topcache.parm("framesperbatch").set(frb)
    topcache.setName(name)
```

Send SOP to a ROP net named "ropnet1", adds version parm to update the original SOP version cache:

```
import hou
nodes = hou.selectedNodes()
ropnet = hou.node("/obj/ropnet1")
for x in nodes:
    name = x.name()+"_"+str(x.parent())
    path= x.path()
    if x.type().name() == "rop_geometry":
        ver = x.parm("vp_version")
    elif x.type().name() == "rop_alembic":
        ver = x.parm("vp_version")
    else:
        ver = x.parm("version")
    topcache = ropnet.createNode("fetch")
    topcache.parm("source").set(path)
    p = hou.IntParmTemplate("version", "Version", 1)
    g = topcache.parmTemplateGroup()
    g.append(p)
    topcache.setParmTemplateGroup(g)
    topcache.parm("version").set(ver)
    topcache.setName(name)
```

SOP TO TOPS updated:

```
import hou
```

```
nodes = hou.selectedNodes()

if not nodes:
    hou.ui.displayMessage("Please select at least one ROP or SOP node.")

else:
    # ------------------------------------------------
    # Find topnet1 in the current network
    # ------------------------------------------------
    editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
    current_network = editor.pwd() if editor else hou.pwd()

    topnet = current_network.node("topnet1")

    # Fall back to /obj/topnet1
    if not topnet:
        topnet = hou.node("/obj/topnet1")

    if not topnet:
        topnet = hou.node("/obj").createNode("topnet", "topnet1")

    # ------------------------------------------------
    # Frame batching for ROPs
    # ------------------------------------------------
    frame_range = hou.playbar.frameRange()
    frb = max(
        1,
        round((frame_range[1] - frame_range[0]) / 2)
    )

    # ------------------------------------------------
    # Process selected nodes
    # ------------------------------------------------
    for x in nodes:

        node_type = x.type().name()
        node_category = x.type().category().name()

        # Explicitly recognize File Cache HDAs
```

```
is_filecache = node_type.startswith("filecache")

# Treat File Cache as SOP regardless of category
is_sop = node_category == "Sop" or is_filecache

# Name: node name followed by parent name
parent_name = x.parent().name()
name = x.name() + "_" + parent_name

# Create TOP ROP Fetch
topcache = topnet.createNode("ropfetch")

# ---------------------------------------------
# SOP handling
# ---------------------------------------------
if is_sop:

    # File Cache internal render ROP
    if is_filecache:
        path = x.path() + "/render"
    else:
        path = x.path()

    # Version parameter
    ver = x.parm("vp_version") or x.parm("version")

    # Simulation detection
    cachesim = x.parm("cachesim")
    initsim = x.parm("initsim")

    is_sim = (
        (cachesim is not None and cachesim.eval() == 1)
        or
        (initsim is not None and initsim.eval() == 1)
    )

    if is_sim:
        # Simulation SOP: RED
        topcache.parm("batchall").set(1)
        topcache.setColor(hou.Color(0.5, 0, 0))
```

```
    else:
        # Normal SOP: GREEN
        topcache.parm("framesperbatch").set(15)
        topcache.setColor(hou.Color(0, 0.5, 0))

# ------------------------------------------------
# ROP handling
# ------------------------------------------------
else:

    path = x.path()

    # Preserve extra naming suffix for ROPs
    extra = x.parm("extra")
    if extra:
        name = x.name() + extra.evalAsString() + "_" + parent_name

    ver = x.parm("vp_version") or x.parm("version")

    topcache.parm("framesperbatch").set(frb)

    # ROP: BLACK, circular shape
    topcache.setColor(hou.Color(0, 0, 0))
    topcache.setUserData("nodeshape", "circle")

# ------------------------------------------------
# Add version parameter
# ------------------------------------------------
if topcache.parm("version") is None:

    p = hou.IntParmTemplate(
        "version",
        "Version",
        1,
        default_value=(1,),
        min=1,
        max=40
    )

    g = topcache.parmTemplateGroup()
```

```
            if g.find("pdg_servicename"):
                g.insertAfter("pdg_servicename", p)
            else:
                g.append(p)

            topcache.setParmTemplateGroup(g)

        # ---------------------------------------------
        # Link version to source node
        # ---------------------------------------------
        if ver:
            relative_path = topcache.relativePathTo(x)

            topcache.parm("version").setExpression(
                'ch("{}/{}")'.format(
                    relative_path,
                    ver.name()
                ),
                language=hou.exprLanguage.Hscript
            )

        # ---------------------------------------------
        # Set ROP path and name
        # ---------------------------------------------
        topcache.parm("roppath").set(path)
        topcache.setName(name, unique_name=True)

    # No layoutChildren() call.
    # Existing TOP network layout remains untouched.
```

Make a file node from a rop export node:

```
import hou, re
nodes = hou.selectedNodes()
parent = nodes[0].parent()
nodeList = []
for x in nodes:
    name = x.name()+"_load"
```

```
    version = x.parm("vp_version").eval()
    path= x.parm("sopoutput").eval()
    path = str(path)
    newpath1 = re.sub("_v\d+", "_v{0}", path).format("`padzero(3,ch('version'))`")
    newpath1 = re.sub("\.\d+", ".$F4", newpath1)



    file = parent.createNode("file")
    try:
        file.setName(name)
    except:
        continue
    file.setColor(hou.Color(0,0,1))
    file.parm("file").set(newpath1)
    p = hou.IntParmTemplate("version", "Version", 1)
    g = file.parmTemplateGroup()
    g.append(p)
    file.setParmTemplateGroup(g)
    file.parm("version").set(version)
    nodeList.append(file)

parent.layoutChildren(items=nodeList)
```

Open Directory from Tops

```
import hou, re
nodes = hou.selectedNodes()

for x in nodes:
    rpath = x.parm("roppath").eval()

    try:
        tnode = rpath.rpartition("/")[0]
        dirpath = hou.node(tnode).parm("file").eval()
    except:
        tnode = rpath
        dirpath = hou.node(tnode).parm("vp_path").eval()
        print(dirpath)
    dirpath = dirpath.rpartition("/")[0]
    hou.ui.showInFileBrowser(str(dirpath))
```

Get Alembic camera, import it, and link to other nodes:

```
import hou, os

#set parent network
network = hou.node("/obj")

#get shot number and cache directory
shot = os.getenv("SHOT")
shotcam = shot.split("/")[-1] +"_cam"
cache = os.getenv("CACHE") + "/layout/cam"



#get alembic file
getcampath = hou.ui.selectFile(cache, "SelectCamera", pattern="*.abc")
getcampath = hou.expandString(getcampath)

#make alembic node, set name, connect to SCALE node and add to CAMERA NODE
alnode = network.createNode("alembicarchive", shotcam)
alnode.parm("fileName").set(getcampath)
alnode.parm("buildHierarchy").pressButton()
alnode.setInput(0, hou.node("/obj/SCALE"))
hou.node("/obj/CAMERA").parm("CAMERA").set(f"/obj/{alnode}/CAM/CAMShape")
```

Selected node in obj level to an object merge:

```
node = hou.selectedNodes()
net = hou.node("/obj")
for x in node:
    name = x.name()
    makeout = net.createNode("geo")
    makeout.setInput(0,x)
    makeout.setName(name+"_INOUT")
    makeout.moveToGoodPosition()
    makeout.setColor(hou.Color(.5, .25, .6))
    merge = makeout.createNode("object_merge")
    merge.parm("objpath1").set(x.path())
```

Add timeshift with clamp after a cache node of a single frame (pipeline annoyance):

```
node = hou.selectedNodes()
for x in node:
    outnode = x.outputs()[0]
    name = x.name() + "_TIMESHIFT_clamp"
    start = x.parm("f1")
    end = x.parm("f2")
    net = x.parent()
    timeshift = net.createNode("timeshift")
    timeshift.parm("rangeclamp").set(3)
    timeshift.parm("frange1").deleteAllKeyframes()
    timeshift.parm("frange2").deleteAllKeyframes()
    timeshift.parm("frange1").set(start)
    timeshift.parm("frange2").set(end)
    timeshift.setName(name)
    timeshift.setInput(0, x)
    outnode.setInput(0, timeshift)
    timeshift.moveToGoodPosition()
```

Tops - range extend, sets the frame range by grabbing from selected cache nodes:

```
node = hou.selectedNodes()[0]
popup = hou.ui.selectNode(title="Select Node to grab
range",node_type_filter=hou.nodeTypeFilter.Sop)

popup = hou.node(popup)
start = popup.parm("f1")
end = popup.parm("f2")

if node.type().name() == "rangeextend":
    node.parm("newrangex").set(start)
    node.parm("newrangey").set(end)
if node.type().name() == "rangegenerate":
```

```
    node.parm("rangex").set(start)
    node.parm("rangey").set(end)
```

Update Multiple Node Params at Once:

```
import hou

# Prompt for parameter name
parm_name = hou.ui.readInput("Parameter to change (e.g. 'uniformscale')", buttons=("OK",
"Cancel"))[1]
if not parm_name:
    raise hou.Error("Parameter name not specified.")

# Prompt for value (literal or expression)
value_input = hou.ui.readInput(f"Value to set for '{parm_name}'\n(Prefix expressions with
expr:)", buttons=("OK", "Cancel"))[1]
if not value_input:
    raise hou.Error("Value not specified.")

# Get selected nodes
selected_nodes = hou.selectedNodes()
if not selected_nodes:
    raise hou.Error("No nodes selected.")

# Check for expression prefix
is_expr = value_input.strip().startswith("expr:")
expr_str = value_input.strip()[5:].strip() if is_expr else value_input.strip()

# Apply to all selected nodes
for node in selected_nodes:
    parm = node.parm(parm_name)
    if not parm:
        print(f"Node '{node.name()}' does not have a parameter named '{parm_name}'")
        continue

    try:
        if is_expr:
            parm.setExpression(expr_str, language=hou.exprLanguage.Hscript)
        else:
            # Try float conversion first
```

```
            try:
                val = float(expr_str)
            except ValueError:
                val = expr_str
            parm.set(val)
    except Exception as e:
        print(f"Failed to set value on '{node.name()}': {e}")
```

Make a new take and enter it:

```
import hou

# Prompt for name
button, text = hou.ui.readInput(
    "Name the new take:",
    buttons=("OK", "Cancel")
)

if button == 1:
    raise hou.OperationFailed("Cancelled")

text = text.strip()
if not text:
    raise hou.OperationFailed("Empty name not allowed")

# Create the take via HScript
hou.hscript(f"takeadd {text}")

# Switch to it
hou.hscript(f"takeset {text}")

hou.ui.displayMessage(f"Created and set take: {text}")
```

Revision #13
Created 2024-01-30 11:33:47 UTC by Anthony
Updated 2026-11-16 15:56:01 UTC by Anthony