My Shelf Setup XML

My shelf tools setup, in progress. Save to houdinixx.x/toolbar/daam.shelf

```
<?xml version="1.0" encoding="UTF-8"?>
<shelfDocument>
  <!-- This file contains definitions of shelves, toolbars, and tools.
 It should not be hand-edited when it is being used by the application.
 Note, that two definitions of the same element are not allowed in
 a single file. -->

  <toolshelf name="daam" label="daam">
    <memberTool name="SopToTops"/>
    <memberTool name="rop_to_top"/>
    <memberTool name="TOPS_rangeselect"/>
    <memberTool name="versionup"/>
    <memberTool name="nodeVersionDown"/>
    <memberTool name="sop_makeFile"/>
    <memberTool name="showfile"/>
    <memberTool name="exportToNew"/>
    <memberTool name="sopTimeClamp"/>
    <memberTool name="Flipbook"/>
    <memberTool name="ChangeMulti"/>
  </toolshelf>

  <tool name="SopToTops" label="SOPtoTOP" icon="hicon:/SVGIcons.index?COP2_subnet.svg">
    <script scriptType="python"><![CDATA[import hou
nodes = hou.selectedNodes()
topnet = hou.node("/obj/topnet1")
if not topnet:
    topnet = hou.node("/obj").createNode("topnet")
    topnet.setName("topnet1")
for x in nodes:
    name = x.name()+"_"+str(x.parent())
    fstart = x.parm("f1").eval()
    fend = x.parm("f2").eval()
    if x.type().name() == "rop_geometry":
```

```
        ver = x.parm("vp_version")
        path = x.path()
    elif x.type().name() == "rop_alembic":
        ver = x.parm("vp_version")
        path = x.path()
    elif x.type().name() == "filecache::2.0":
        path= x.path() + "/render"
        ver = x.parm("vp_version")
    else:
        ver = x.parm("version")
        path = x.path()

    topcache = topnet.createNode("ropfetch")
    try:
        if x.parm("cachesim").eval() is 1:
            topcache.parm("batchall").set(1)
            topcache.setColor(hou.Color(.5,0,0))
        else:
            topcache.parm("framesperbatch").set(15)
            topcache.setColor(hou.Color(0,.5,0))
    except:
        if x.parm("initsim").eval() is 1:
            topcache.parm("batchall").set(1)
            topcache.setColor(hou.Color(.5,0,0))
        else:
            topcache.parm("framesperbatch").set(15)
            topcache.setColor(hou.Color(0,.5,0))
    p = hou.IntParmTemplate("version", "Version", 1, min=1, max=40)
    g = topcache.parmTemplateGroup()
    g.insertAfter("pdg_servicename", p)
    topcache.setParmTemplateGroup(g)
    topcache.parm("version").set(ver)
    topcache.parm("roppath").set(path)
    topcache.setName(name)]]></script>
  </tool>

  <tool name="TOPS_rangeselect" label="TOPS RangeSelect"
icon="hicon:/SVGIcons.index?BUTTONS_set_framerange.svg">
    <script scriptType="python"><![CDATA[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")
start.deleteAllKeyframes();
end.deleteAllKeyframes();

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)]]></script>
  </tool>

  <tool name="versionup" label="Node Version UP" icon="hicon:/SVGIcons.index?BUTTONS_up.svg">
    <script scriptType="python"><![CDATA[import hou
nodes = hou.selectedNodes()
for x in nodes:
        try:
            ver = x.parm("vp_version").eval() +1
        except:
            ver = x.parm("version").eval() + 1
        x.parm("version").set(ver)]]></script>
  </tool>

  <tool name="rop_to_top" label="ROPtoTOP" icon="hicon:/SVGIcons.index?CHOP_reorder.svg">
    <script scriptType="python"><![CDATA[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("vp_version")
    path = x.path()
    name = x.name()
    if x.parm("extra"):
        name = name + x.parm("extra").eval()
    else: pass
    topcache = topnet.createNode("ropfetch")
    topcache.setColor(hou.Color(0,0,0))
    topcache.setUserData("nodeshape", "circle")
    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)]]></script>
  </tool>

  <tool name="sop_makeFile" label="SOP makeFile From ROP"
icon="hicon:/SVGIcons.index?BUTTONS_upload.svg">
    <script scriptType="python"><![CDATA[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)]]></script>
  </tool>

  <tool name="showfile" label="TOPS Show File in Nautilus"
icon="hicon:/SVGIcons.index?BUTTONS_addassetdirectory.svg">
    <script scriptType="python"><![CDATA[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))]]></script>
  </tool>

  <tool name="exportToNew" label="SOP send to new"
icon="hicon:/SVGIcons.index?BUTTONS_cut.svg">
    <script scriptType="python"><![CDATA[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())]]></script>
  </tool>

  <tool name="sopTimeClamp" label="SOP time clamp"
icon="hicon:/SVGIcons.index?SOP_timeblend.svg">
    <script scriptType="python"><![CDATA[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()]]></script>
  </tool>

  <tool name="nodeVersionDown" label="Node Version DOWN"
icon="hicon:/SVGIcons.index?BUTTONS_down.svg">
    <script scriptType="python"><![CDATA[import hou
nodes = hou.selectedNodes()
for x in nodes:
        ver = x.parm("version").eval() -1
        x.parm("version").set(ver)]]></script>
  </tool>

  <tool name="Flipbook" label="Flipbook" icon="hicon:/SVGIcons.index?BUTTONS_capture.svg">
    <script scriptType="python"><![CDATA[import hou
import re
```

```
def flipbook_from_node():
    try:
        # Ask user to pick a ROP node to steal the path from
        stealfile = hou.ui.selectNode(
            title="Select OpenGL Export Node to steal path from",
            multiple_select=False,
            node_type_filter=hou.nodeTypeFilter.Rop
        )
        if not stealfile:
            return

        # Get the output path and convert to $F4 format
        filepath = hou.node(stealfile).parm("vp_path").eval()
        renderpath = re.sub(r'\.\d+\.', '.$F4.', filepath)

        # Confirm the output path
        confirm = hou.ui.displayMessage(
            f"Use this path?\n\n{renderpath}",
            buttons=('Yes', 'No'),
            default_choice=0
        )
        if confirm != 0:
            return

        # Resolution preset options
        resolution_presets = [
            "1920x1080",  # Default
            "1280x720",
            "2048x858",
            "3840x2160",
            "Custom"
        ]

        # Ask user to select resolution preset
        choice = hou.ui.selectFromList(
            resolution_presets,
            message="Select Flipbook Resolution",
            title="Resolution Presets",
            exclusive=True,
            default_choices=[0]
```

```
        )

        if not choice:
            return

        selection = resolution_presets[choice[0]]

        if selection == "Custom":
            # Ask for custom resolution
            resx_str = hou.ui.readInput("Enter custom resolution X:", buttons=("OK",),
close_choice=0)[1]
            resy_str = hou.ui.readInput("Enter custom resolution Y:", buttons=("OK",),
close_choice=0)[1]
            resx = int(resx_str)
            resy = int(resy_str)
        else:
            resx, resy = map(int, selection.split("x"))

        # Get current frame range
        frames = hou.playbar.frameRange()

        # Setup the flipbook
        cur_desktop = hou.ui.curDesktop()
        scene = cur_desktop.paneTabOfType(hou.paneTabType.SceneViewer)
        if not scene:
            hou.ui.displayMessage("No Scene Viewer found.")
            return

        scene.flipbookSettings().stash()
        flipbook_options = scene.flipbookSettings()

        flipbook_options.output(renderpath)
        flipbook_options.frameRange((frames[0], frames[1]))
        flipbook_options.useResolution(True)
        flipbook_options.resolution((resx, resy))

        # Run the flipbook
        scene.flipbook(scene.curViewport(), flipbook_options)

    except Exception as e:
```

```
        hou.ui.displayMessage(f"Error: {str(e)}")

# Run it
flipbook_from_node()]]></script>
  </tool>

  <tool name="ChangeMulti" label="ChangeMultiParm"
icon="hicon:/SVGIcons.index?BUTTONS_pdg_task_table.svg">
    <script scriptType="python"><![CDATA[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}")
]]></script>
  </tool>

  <shelfSetEdit name="shelf_set_1" fileLocation="/mnt/jobs/software/houdini/hfs20.5.370-
py310/houdini/toolbar/ShelfDefinitions.shelf">
    <addMemberToolshelf name="NOISE" inPosition="14"/>
    <addMemberToolshelf name="daam" inPosition="15"/>
  </shelfSetEdit>
</shelfDocument>
```

Revision #3
Created 2025-05-13 22:52:37 UTC by Anthony
Updated 2025-05-30 15:52:36 UTC by Anthony