Nuke Commands, Python, TCL

- Top Node AutoWrite
- TCL Commands
- Retime track in nuke
- Python - import all image sequences under folder
- Random Python

Top Node AutoWrite

if you need your output file named exactly like your input file then use a little TCL expression in the file knob of the write node

suggestion your filename convention is like: //path/to/file/filename.pattern.ext

then:

```
[lindex [split [lindex [split [knob [topnode].file] .] 0] /] end]
```

will give you back just the "filename" part of the original read node

if you want to write to same folder as the original file but with a new filename extension you could use:

```
[file dirname [knob [topnode].file]]/[lindex [split [lindex [split [knob [topnode].file] .] 0] /] end]_conv.%04d.exr
```

TCL Commands

```
xxxxxxxxxx
```

1

```
GETTING A KNOB’S VALUE OF A SPECIFIC NODE:
```

2

```

```

3

```
#First frame of current read/write:
```

4

```
[value Read1.first]
```

5

```

```

6

```
#Getting a knob’s value of current node:
```

7

```
[value this.first_frame]
```

8

```

```

9

```
#Return label value of the input node:
```

10

```
[value this.size]
```

11

```

```

12

```
#Name of the input node:
```

13

```
[value this.input0.label]
```

14

```

```

```
15
#Name of the node before the group (Outside):
16
[value this.input.name]
17

18
#Return 1 if the node is on error otherwise 0:
19
[value this.parent.input.name]
20

21
#Get the bounding Box from the input of the node:
22
[value error]
23

24

25
#Here some expression for the Format
26
format.x
27
format.y
28
width
29
height
```

```
30
    bbox.x
31
    bbox.y
32
    bbox.w
33
    bbox.h
34
    
35
    #Get the format from the input of the node:
36
    #left boundary
37
    [value input.bbox.x]
38
    #right boundary
39
    [value input.bbox.r]
40
    
41
    #Get the format from the input of the node:
42
    #width
43
    [value input.format.r]
44
    #height
```

```
[value input.format.t]
```

```

```

```
#Get the x position of the point #3 of the Bezier1 of the Roto1 node:
```

```
[value Roto1.curves.Bezier1.curve_points.3.main.x]
```

```

```

```
#Return sample pixel value of the node Add1 reading in the red at position of knob Center:
```

```
[sample Add1 Red Center.x Center.y]
```

```

```

```
#Get the value of the channel of a node, at a specific pixelcoordinates (e.g.: 10,10):
```

```
[sample [node input] red 10 10]
```

```

```

```
#-------------------------------------------------------------------
```

```
#SET VALUES
```

```

```

```
#Setting a knob’s value of a specific node:
```

```
[knob Read1.first 10]
```

```

```

```
#Setting a variable, without returning that (useful in a textnode):
```

```
[set seq [value Read1.file]; return]
```

```

```

```
#------------------------------------------------------------------
```

```
#STRING
```

```

```

```
#Replace string in current node file knob with regex (string “proj” to “projects” in all oc
```

```
[regsub -all "proj" [value [node this].file] "projects"]
```

```

```

```
#String map (replace multiple stringpairs) (this returns: xxffffxxyy):
```

```
[string map {"aa" "xx" "bb" "yy"} "aaffffaabb" ]
```

```

```

```
#Compare strings:
```

```
[string equal [value Text1.message] "bla"]
```

```

```

```
#Regexp matching:
```

```
[regexp -inline "_v\[0-9]{3}" [value Read2.file]]
```

```

```

```
#Evaluating string
```

```
[python os.getenv('rotate') == 'xavierb']
```

```

```

```
#-----------------------------------------------------------------------
```

```
#IF CONDITION
```

```

```

```
[if {condition} {expr1} else {expr2}]
```

```

```

```
#Example:
```

```
[if {[value blackpoint]==1} {return 2} {return 3}]
```

```
[if {[value blackpoint]==1} {return True} {return False}]
```

```
[if {[value blackpoint]==1} {return blackpoint} {return whitepoint}]
```

```
[if {[value filter]=="gaussian"} {return filter} {return False}]
```

```

```

```
#OTHER METHOD
```

```
condition ? then : else
```

```

```

```
#Example:
```

```
#if (r==1)? return 0: else (return r*2)
```

```
r ==1 ? 0 : r*2
```

```

```

```
#-----------------------------------------------------------------------
```

```
#PATH MANIPULATIONS:
```

```
□
```

10
4

```
#Filepath without extension:
```

10
5

```
[file rootname [value [topnode].file]]
```

10
6

```
□
```

10
7

```
#Filename only:
```

10
8

```
[basename [value [topnode].file ]]
```

10
9

```
□
```

11
0

```
#Filename only without extension:
```

11
1

```
[basename[file rootname [value [topnode].file]]]
```

11
2

```
□
```

11
3

```
#-------------------------------------------------------------------
```

11
4

```
#RELATIVE PATH
```

11
5

```
#In the Read node you can use the relative path for your footage/Obj
```

11
6

```

```

11
7

```
#In your Read Node in the knob "file", use this:
```

11
8

```

```

11
9

```
#Read file "render.exe" in the same folder of your file nuke
```

12
0

```
[python {nuke.script_directory()}]/render.exr
```

12
1

```

```

12
2

```
#Read file "render.exe" in the subfolder where your file nuke is
```

12
3

```
[python {nuke.script_directory()}]/folder/render.exr
```

12
4

```

```

12
5

```
#Read file "render.exe" in the subfolder of your .nuke folder
```

12
6

```
[python {"/Users/gere/.nuke} ]/folder/render.exr
```

Retime track in nuke

Retiming a curve

There are many retiming options in Nuke but how do you retime tracking data or animation?
The answer is actually very simple and can be used wherever you have an animation curve in your script. If you are using an OFlow node, called ‘OFlow1’, and are using ‘Source Frame’ timing, rather than ‘Speed’ to retime a plate, you can access the retime curve by referring to ‘***OFlow1.timingFrame***‘.

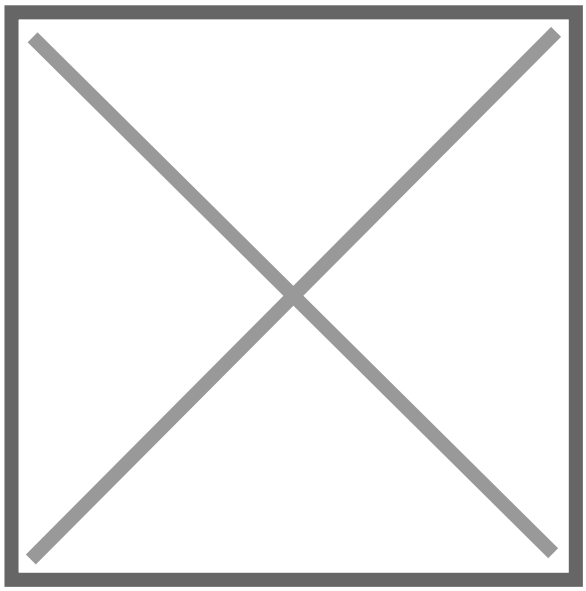
So if you apply the following expression to any other curve in your script, that animation curve will be retimed to match OFlow1:
curve(OFlow1.timingFrame)



Translated: “*For the frame I am currently on, instead of using the current value, jump to the frame the OFlow is currently looking at and use the value from this curve at that frame*“.

Offsetting a curve

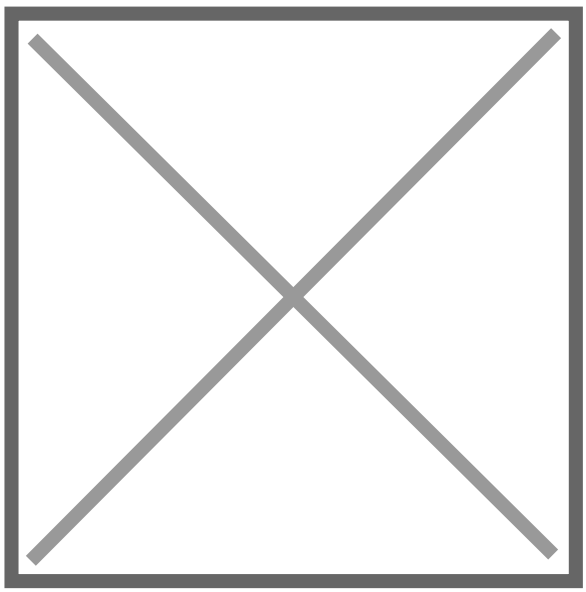
Any control that has keyframe animation, right click the curve icon and choose ‘***Edit Expressions…***‘ and you should see it has one expression in there already: ‘***curve***‘. This is just the curve created from your keyframes. You can treat that curve like a variable, for example ‘***curve*2***‘ to double the amplitude of the curve. Putting a value in brackets after it allows you to slip the curve back and forth in time. ‘***curve(frame-5)***‘ would offset the curve by 5 frames. “*At this frame, don’t use this value, use the one 5 frames earlier*“. ‘***frame***‘ is a variable that always contains the current frame number. The resulting output would look something like this:



An example script

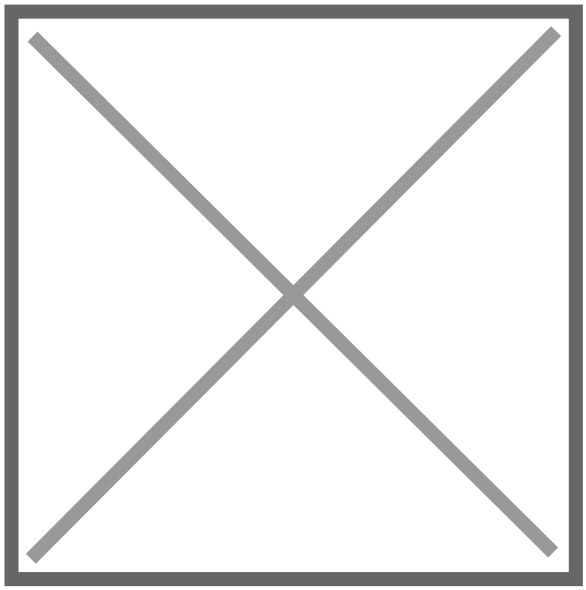
Let’s say you have a shot of a building and you want to patch in some extra windows. You don’t want to use a still because there are shadows of people moving around inside the building so you want to copy another window, translate it to the new position and then retime the input so the people moving round look less repetitive. If it was a locked off shot, you could just retime it, copy and translate it. But if there is a camera move and camera wobble it will need to be stabilized, translated then match-moved. But where do you put the retime node? It makes sense to put it after the stabilize, but then you will break concatentation with the translate and match-move nodes, potentially introducing softening. Using the above method of retiming the tracking data, you can keep the transform nodes together and keep concatentation.

In this example I’m using a CornerPin node as my stablize/match-move, exported from Mocha Pro.

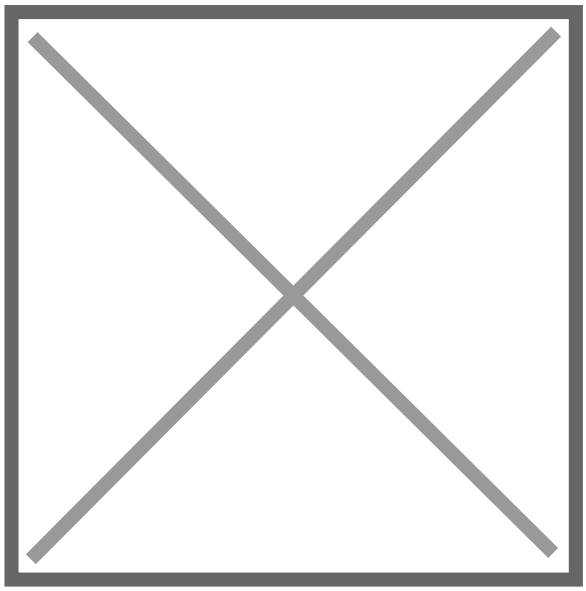


One of the patches in this example doesn’t use any retiming, and the other does. They use an identical setup but one has expressions linked to the OFlow. I have put a dot node in there so you can see the expression line more clearly. The CornerPin nodes labelled ‘stabilize’ and ‘match’ are identical but the stabilize ones have the ‘invert’ check box ticked. I could have cloned several of them but the tracking data is unlikely to change so a straight copy/paste is better here.

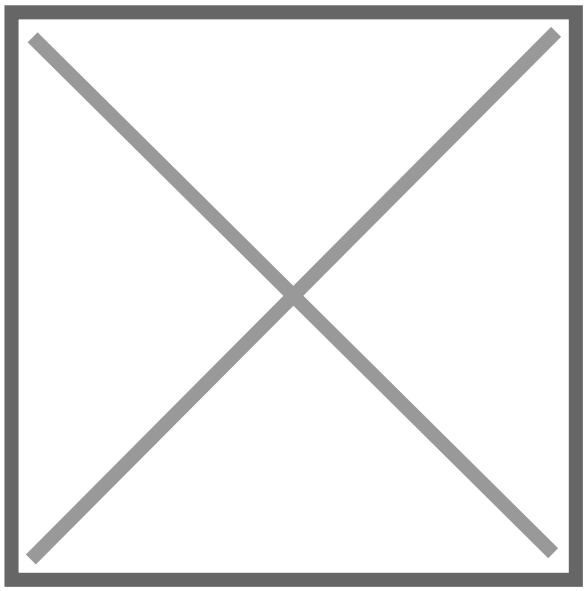
When I right click on any parameter in cornerPinTrack4 and choose ‘***Edit Expressions…***‘ I enter the above expression into each corner:



No matter how I animate my OFlow1 node, the cornerPinTrack4 node will adjust it’s tracking data to match the newly retimed plate, keeping it stabilized. Of course, if you retime it to extremes, the plate and/or tracking data will run out. In this example I have just slowed it down and sped it up again. The retime curve looks like this:



When I look at the curve for one of the values in my corner pin (the x value of corner 1) it shows the original keyframes with the newly calculated curve on top:



I drew the roto shape on my reference frame around the window I wanted to copy, whilst looking at the result of the cornerPinTrack4. I then cloned the offset transform node and put it beneath the roto shape. I could have just viewed the result of the offset transform node when I drew my rotoshape to achieve the same thing, but this way I can change my mind about the offset later if I want, moving it to a different position and the rotoshape will always be in the correct place.

To avoid grain issues you will probably want to degrain the plate and regrain the patches afterwards. Even if it doesn't get softened, you don't want grain running at different speeds.

Using a TimeWarp node instead

If you are retiming with a timewarp node, you can use the 'lookup' value instead to achieve the same thing:
curve(TimeWarp1.lookup)

Summary

This trick can be used anywhere in your nuke script; anywhere there's a keyframed knob, you can adjust the curve by adding an expression to it. In most cases retiming is done to the plate beforehand or at the end of a comp, so your matchmove will have been done on the retimed plate already, or if at the end, then it doesn't matter. But occasionally a decision will be made later on, after you have your matchmove, to retime something and it means throwing off all your animation.

But using this expression you can keep the existing animation, whether that be tracking, a 3D camera, switches, fades or transforms, and avoid precomps and other methods that might introduce softness by breaking concatenation.

And yes, I did say 3D camera… if you have a tracked camera but the plate has to be retimed, you can put this expression on every animated value in that camera (once you untick '***read from file***') and the camera will now match the retimed plate. No need to send it back to the matchmove department.

Python - import all image sequences under folder

```
"""Finds and creates nodes for file sequences in an entire directory tree"""
# This has the user select a folder, then creates Read nodes
# for all files and sequences under that folder
# lewis@lewissaunders.com 03/2011

import os, nuke, re, time

from curses.ascii import isprint

# Returns a printable version of the input
def printable(input):
    return ''.join(char for char in input if isprint(char))

# Returns a list of all folder paths beneath the given head folder
def folders(head):
    head = head.rstrip('/')

    ret = []
    ret.append(head)
    for (root, dirs, files) in os.walk(head, followlinks=True):
        for d in dirs:
            ret.append(root + '/' + d)
    return ret

# Returns a Nuke-style list of image sequences under the given folder
def seqs(path):

    prog = nuke.ProgressTask("Looking...")
    prog.setProgress(0)

    all_folders = folders(path)
```

```
chunk_list = []

i = 0
for f in all_folders:
    prog.setMessage("Looking under " + os.path.basename(f))
    prog.setProgress((100 * i)/len(all_folders))

    time.sleep(0.01)

    raw = nuke.tcl('filename_list -compress "' + f + '"')

    if raw == None:
        continue

    # Little state machine which munches through the string,
    # working out how to separate it
    inside = False        # True when current char is inside {..} pair
    sep = False        # True when the current char is { or } or space
    prev_inside = False
    prev_sep = True
    chunk = f + '/'

    for c in raw:
        if c == '{':
            sep = True
            inside = True
        elif c == '}':
            sep = True
            inside = False
        elif c == ' ' and not inside:
            sep = True
        elif c == ' ' and inside:
            sep = False
        else:
            sep = False

        if sep:
            if not prev_sep:
                # End of word
                chunk_list.append(chunk)
```

```
                    chunk = f + '/'
            else:
                # Start or middle of chunk
                chunk = chunk + c

            prev_sep = sep
            prev_inside = inside

        # Get final chunk
        chunk_list.append(chunk)
        i = i + 1

    # Lose folders and .ifds, keep files
    chunk_list_no_folders = []
    for f in chunk_list:
        if os.path.isdir(f):
            continue
        if re.search('\.ifd ', f) != None:
            continue
        if re.search('\.ifd.old', f) != None:
            continue
        if printable(f) != f:
            # Non-ASCII characters - skip it
            continue
        chunk_list_no_folders.append(f)
    chunk_list_no_folders.sort()

    del prog

    return chunk_list_no_folders

# Prompt for a folder then create a pile of Read nodes
def doit():
    # Get the selected node, and the path on it's 'file' knob if it
    # has one, or failing that, it's 'proxy' knob, if it has that.
    sel_node = None
    default_dir = None
    try:
        sel_node = nuke.selectedNode()
    except:
```

```
        pass
    if ( sel_node is not None ) and ( sel_node != '' ):
        if 'file' in sel_node.knobs():
            default_dir = sel_node['file'].value()
        if (default_dir == None or default_dir == '') and 'proxy' in sel_node.knobs():
            default_dir = sel_node['proxy'].value()

    # Revert default_dir to None if it's empty so that the file browser
    # will do it's default unset behaviour rather than open on an empty path.
    if default_dir == '':
        default_dir = None



    #
    # (Above lines robbed from nukescripts.create)
    #



    p = nuke.getClipname("Read folder recursively...", default=default_dir, multiple=False)

    if p == None:
        return

    sequences = seqs(p)

    # Pops up a panel with an editable textbox with all the paths
    lines = ''
    for s in sequences:
        lines = lines + s + '\n'
    p = nuke.Panel('Found these files...')
    p.addMultilineTextInput('Found:', lines)
    p.setWidth(1720)
    if not p.show():
        return

    for f in p.value('Found:').splitlines():
        if f != '':
            nuke.createNode('Read', 'file {%s}' % f)

doit()
```

Random Python

> Get the file path from selected nodes:

```
sel=nuke.selectedNodes()
for x in sel:
    print(x.knob("file").getValue())
```