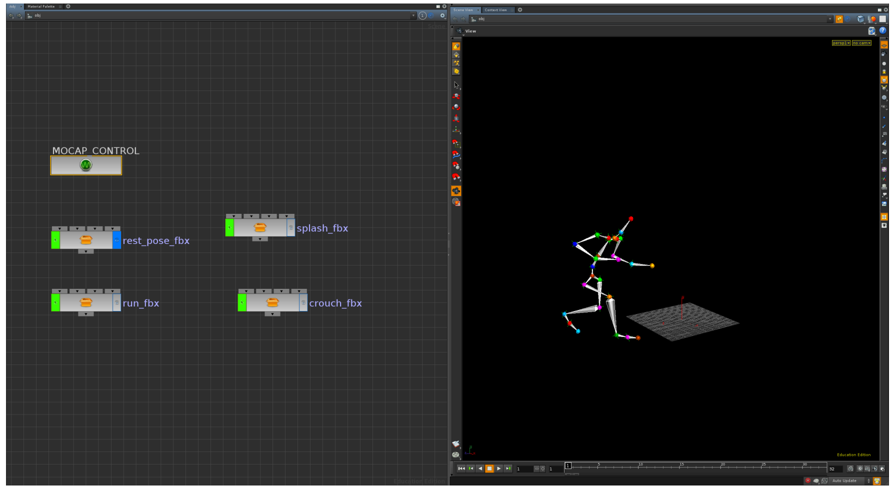
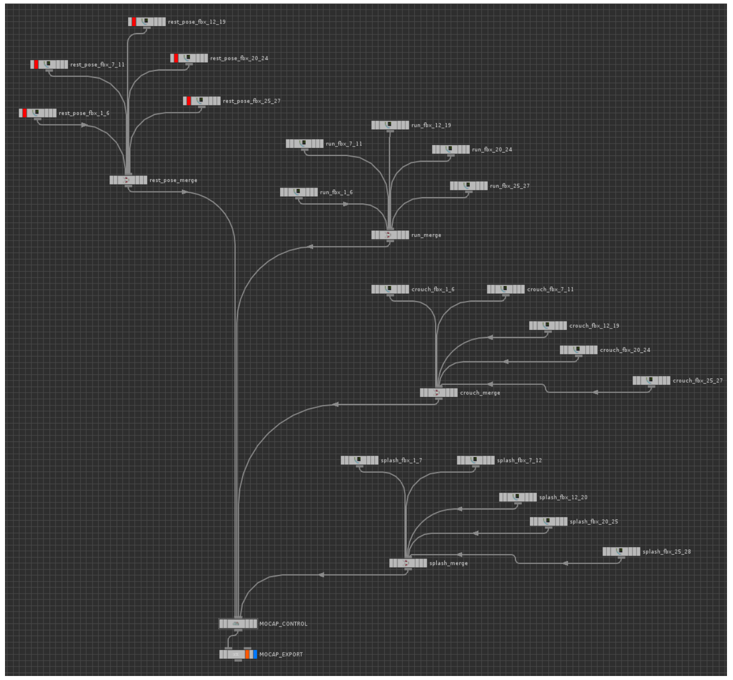
H12.1 – Hscripting #2

This lecture will look at developing a **HScript** to automatically configure a **MOCAP Setup** for Houdini. This is based upon the hand created MOCAP Network given in an earlier lecture.



Open the scene **fbx_mocap_complete.hipnc**. This scene contains a hand configured mocap setup that can be used as reference whilst developing the script. MOCAP clips are imported as fbx data, and a custom CHOP Network has been configured to switch the rest_pose_fbx MOCAP data with the other MOCAP fbx clips.

Each fbx MOCAP clip is read into the custom **CHOP Network** using **Object Chain CHOPs** (one for each identified bone chain of the MOCAP skeleton hierarchy).

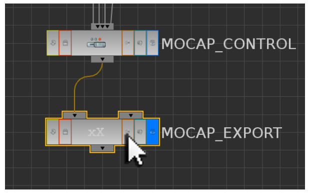


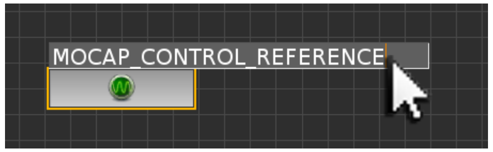
A **Switch CHOP** is then used to switch between the MOCAP clips re-assigning the data back onto the original rest_pose_fbx MOCAP clip through use of a **Rename CHOP**.

NOTE: A full list of HScript commands can be seen in a Textport by typing help. Typing help –k <keyword> can précis this list to only the HScript commands relating to a certain keyword. Typing help <HScript command name> will return the Help Card for a specific command.

CREATING AN AUTOMATED MOCAP SETUP SCRIPT

Inside the **CHOP Network**, **deactivate** the **Orange Export Flag** on the **Rename CHOP**. This will restore the original rest pose for the MOCAP skeleton.

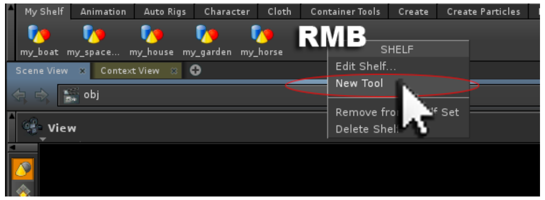




Back at **Object Level**, **rename** the **MOCAP CHOP Network** to **MOCAP_CONTROL_REFERENCE**. This will allow for an automated MOCAP CHOP Network to be created, and compared with the original hand made network as the MOCAP Hscript is being developed.

SCRIPT BASED SHELF TOOLS

As well as being able to store HScripts as external text files, and sourcing them into Houdini, it is also possible to create HScripts as Shelf Tools.



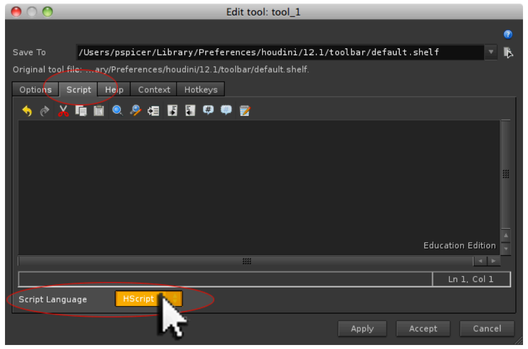
Unstow My Shelf, and **RMB** on an empty part of it. From the resulting menu choose **New Tool**.

Under the **Options** section of the resulting **Edit Tool Window** specify:

Name	**MOCAP_Setup**
Label	**MOCAP_Setup**

Under the **Script** section of the **Edit Tool Window** specify:

Script Language	**HScript**


Edit tool: tool_1

Save To /Users/pspicer/Library/Preferences/houdini/12.1/toolbar/default.shelf

Original tool file: ...ary/Preferences/houdini/12.1/toolbar/default.shelf.

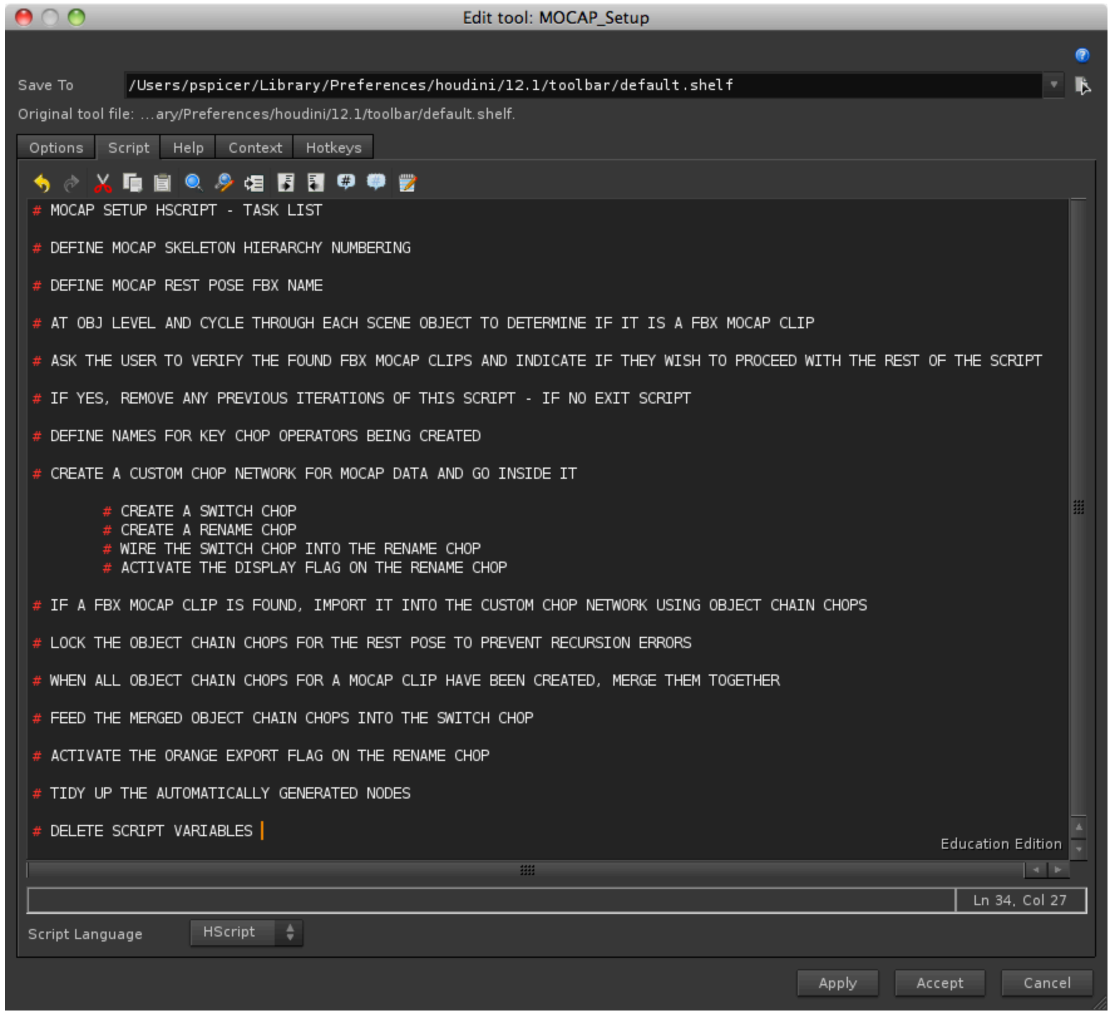
Ln 1, Col 1

Script Language HScript

Press the **Apply** button to confirm the changes.

HSCRIPT TASK LISTS

When developing scripts it is a good idea to develop a **Task List** to map out the functionality of the script before any actual commands are initiated. This Task List outlines all of the key development areas for the script, defining their purpose. As the script develops, this Task List will become comments articulating the script, making it simpler to read.



Save To /Users/pspicer/Library/Preferences/houdini/12.1/toolbar/default.shelf

Original tool file: ...ary/Preferences/houdini/12.1/toolbar/default.shelf

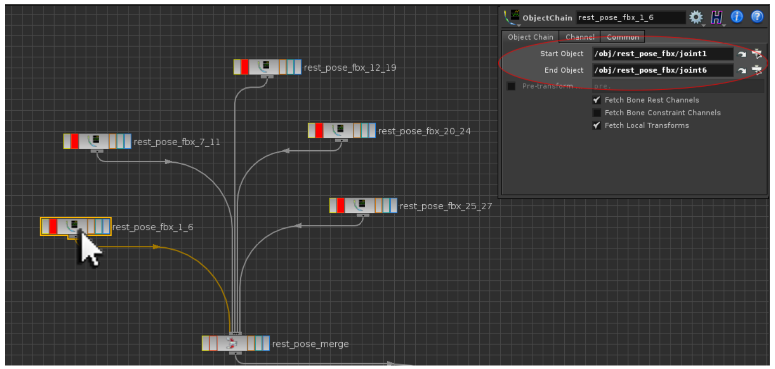
```
# MOCAP SETUP HSCRIPT - TASK LIST

# DEFINE MOCAP SKELETON HIERARCHY NUMBERING

# DEFINE MOCAP REST POSE FBX NAME

# AT OBJ LEVEL AND CYCLE THROUGH EACH SCENE OBJECT TO DETERMINE IF IT IS A FBX MOCAP CLIP

# ASK THE USER TO VERIFY THE FOUND FBX MOCAP CLIPS AND INDICATE IF THEY WISH TO PROCEED WITH THE REST OF THE SCRIPT

# IF YES, REMOVE ANY PREVIOUS ITERATIONS OF THIS SCRIPT - IF NO EXIT SCRIPT

# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED

# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE IT

        # CREATE A SWITCH CHOP
        # CREATE A RENAME CHOP
        # WIRE THE SWITCH CHOP INTO THE RENAME CHOP
        # ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP

# IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM CHOP NETWORK USING OBJECT CHAIN CHOPS

# LOCK THE OBJECT CHAIN CHOPS FOR THE REST POSE TO PREVENT RECURSION ERRORS

# WHEN ALL OBJECT CHAIN CHOPS FOR A MOCAP CLIP HAVE BEEN CREATED, MERGE THEM TOGETHER

# FEED THE MERGED OBJECT CHAIN CHOPS INTO THE SWITCH CHOP

# ACTIVATE THE ORANGE EXPORT FLAG ON THE RENAME CHOP

# TIDY UP THE AUTOMATICALLY GENERATED NODES

# DELETE SCRIPT VARIABLES
```

Script Language HScript

An **initial Task List** for the automated MOCAP Setup Script may look something like the above entry, and is derived from examination of the **fbx_mocap_complete.hipnc** file.

DEFINING VARIABLES

Before objects or components of a HScript are created it is useful to define variable names for them. This can make managing objects or components of the HScript simpler. The **set command** can be used to **define HScript variables**. The set command does not require initial definitions of data types (for example float, int or string), as the command works without such definitions as a prerequisite. For the MOCAP setup script, the **numbering of the Skeleton Hierachy** needs to be set as a variable. This variable needs to be set by the end user, as different MOCAP skeletons may require different number ranges.



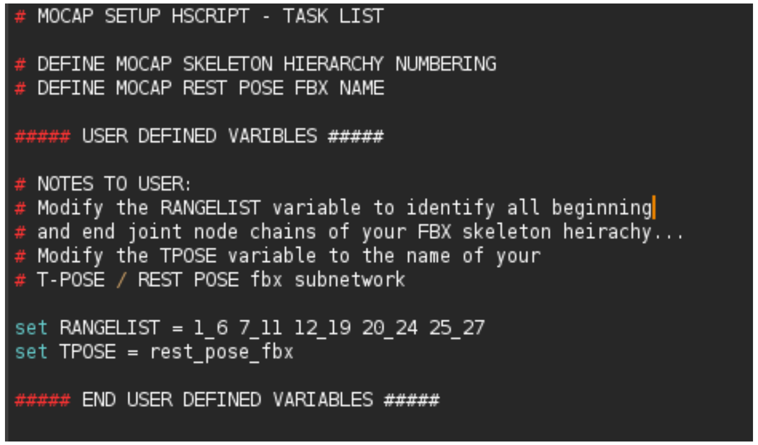
ObjectChain rest_pose_fbx_1_6

Start Object /obj/rest_pose_fbx/joint1
End Object /obj/rest_pose_fbx/joint6
Pre-transform ... pre.
[✓] Fetch Bone Rest Channels
[ ] Fetch Bone Constraint Channels
[✓] Fetch Local Transforms

Examination of the **MOCAP_CONTROL_REFERENCE CHOP Network** reveals that an **Object Chain CHOP** needs to be created for **each branch** of the MOCAP skeleton, with each Object Chain CHOP referencing the **beginning** and **end bones** of the branch in its parameters. From this setup, the following number ranges can be identified:

Joints: 1 to 6; 7 to 11; 12 to19; 20 to 24; and 25 to 27

Declaring this number range as a variable at the start of the script will allow for end user changes to be done simply, rather than having this information buried in the main body of the script. Similarly, having an **end user declare the rest pose fbx object name** can help with the porting back of the MOCAP data onto the rest pose fbx object correctly. Again this is a name that might vary from setup to setup, so having variable control over how this name is used in the main body of the script is better than hardwiring this name into the script.

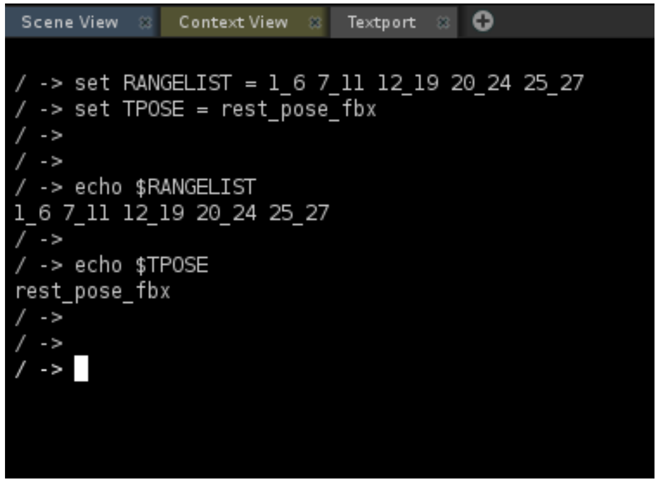


```
# MOCAP SETUP HSCRIPT - TASK LIST

# DEFINE MOCAP SKELETON HIERARCHY NUMBERING
# DEFINE MOCAP REST POSE FBX NAME

##### USER DEFINED VARIBLES #####

# NOTES TO USER:
# Modify the RANGELIST variable to identify all beginning
# and end joint node chains of your FBX skeleton heirachy...
# Modify the TPOSE variable to the name of your
# T-POSE / REST POSE fbx subnetwork

set RANGELIST = 1_6 7_11 12_19 20_24 25_27
set TPOSE = rest_pose_fbx

##### END USER DEFINED VARIABLES #####
```

Use the **set command** to generate variables called **RANGELIST** and **TPOSE** as end user variables. Instructions for the end user can also be added as comments. Key variables should always have concise but informative names, to help save time when developing a script. **Variables** are also **CASE Sensitive**, and for legibility should be **upper case** in accordance with standard variables such as **$HOME**.

TESTING VARIABLES

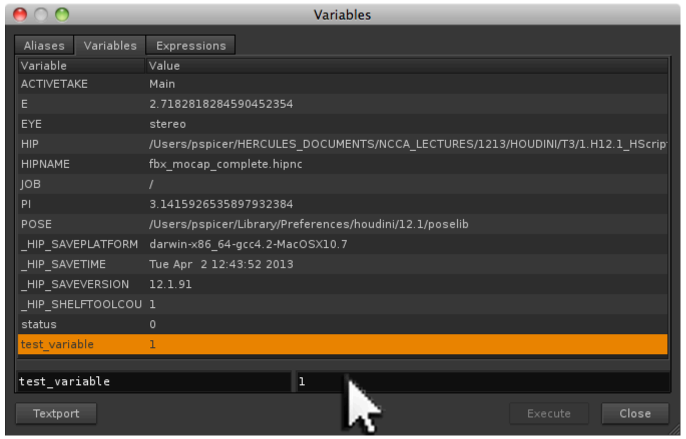
When HScripts are being developed, it can be useful to test variables to ensure they are returning what is expected. This can be done in a **Textport** by **MMB pasting selected text** of the HScript, and then using the **echo command** to verify that the variables have been set correctly. For example, the **set RANGELIST** and **set TPOSE** commands can be entered into a **Textport** and verified accordingly.



```
/ -> set RANGELIST = 1_6 7_11 12_19 20_24 25_27
/ -> set TPOSE = rest_pose_fbx
/ ->
/ ->
/ -> echo $RANGELIST
1_6 7_11 12_19 20_24 25_27
/ ->
/ -> echo $TPOSE
rest_pose_fbx
/ ->
/ ->
/ ->
```

Any custom variable will remain active in a Houdini scene until it is overridden or deleted (**using the set –u option**), or the scene is closed.

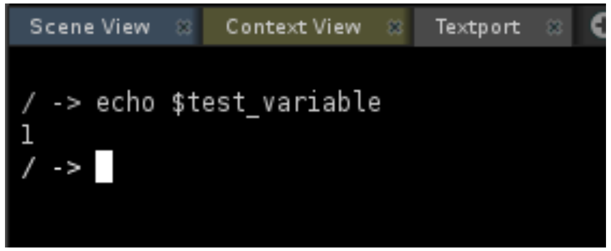
Key Variables can also be declared in the main **Edit menu > Aliases and Variables window**. These will treated as global variables accessible anywhere in Houdini and will be saved with a scene.



Variable	Value
ACTIVETAKE	Main
E	2.7182818284590452354
EYE	stereo
HIP	/Users/pspicer/HERCULES_DOCUMENTS/NCCA_LECTURES/1213/HOUDINI/T3/1.H12.1_HScrip
HIPNAME	fbx_mocap_complete.hipnc
JOB	/
PI	3.1415926535897932384
POSE	/Users/pspicer/Library/Preferences/houdini/12.1/poselib
_HIP_SAVEPLATFORM	darwin-x86_64-gcc4.2-MacOSX10.7
_HIP_SAVETIME	Tue Apr 2 12:43:52 2013
_HIP_SAVEVERSION	12.1.91
_HIP_SHELFTOOLCOU	1
status	0
test_variable	1

```
test_variable    1
```

Variables declared in this way can also be called in both HScripts and the textport.



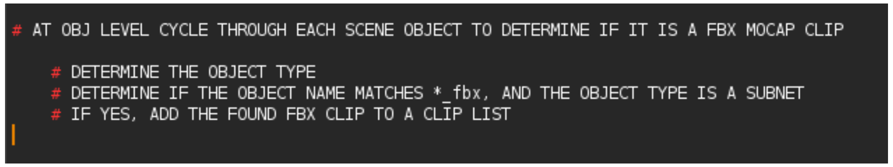
```
/ -> echo $test_variable
1
/ ->
```

Similarly, **global variables** created in a Textport or HScript **using the set –g option** will automatically appear in the **Aliases and Variables window** after they have been declared.

NOTE: When testing a Key Variable in a HScript, the message command should be used instead of the echo command.

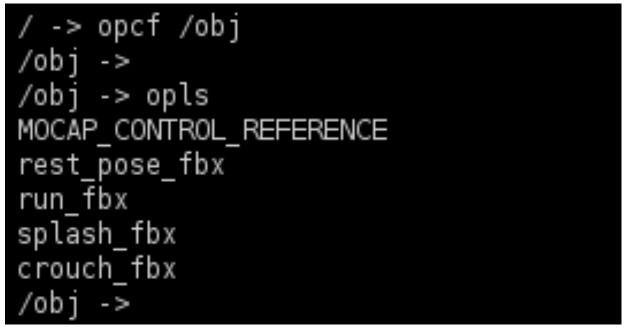
EXAMINING OBJECTS IN TURN

The next task of the MOCAP Setup Script is at Object Level to cycle through each scene object in turn to determine if it is a MOCAP FBX clip or not. Additional tasks can be added to the script to help decipher the steps required to automatically detect MOCAP FBX clips. As MOCAP clips are imported as **subnetworks** with a **name suffix** of **_fbx**, these specific scene objects can be identified through comparisons with other object types and their names.



```
# AT OBJ LEVEL CYCLE THROUGH EACH SCENE OBJECT TO DETERMINE IF IT IS A FBX MOCAP CLIP

    # DETERMINE THE OBJECT TYPE
    # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT TYPE IS A SUBNET
    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
```

Activate a **Textport** and use the **opcf** and **opls** commands to navigate to **Object Level** and list its contents.



```
/ -> opcf /obj
/obj ->
/obj -> opls
MOCAP_CONTROL_REFERENCE
rest_pose_fbx
run_fbx
splash_fbx
crouch_fbx
/obj ->
```

This method of listing can also be used to create a variable list inside a HScript.

THE FOR EACH LOOP AND IF STATEMENT

As each scene object needs to be assessed individually, a For Each Loop can be used in conjunction with the opls list command to facilitate this. Pseudo code for a For Each Loop can explain best how its works.

```
for each vehicle found in the list (car, boat, plane, train, bus)
        {
                echo $vehicle
        }
end
```

With each iteration of the for each loop through the list, the next item gets automatically assigned to the $vehicle variable, and echoed back to the end user. This pseudo code can also have an if statement embedded into it, to perform an action on specific objects found:

```
for each vehicle found in the list (car, boat, plane, train, bus)
        {
                if $vehicle == “plane”
                echo “Found a plane”
                endif
        }
end
```

These principles can be adapted to MOCAP Setup HScript accordingly. Under the task comment citing the cycling through of each scene object, add the following commands:

opcf /obj

<this will navigate to Object Level>

set FBXOBJECTLIST = " "

<this will create an empty list variable to eventually store found fbx clips

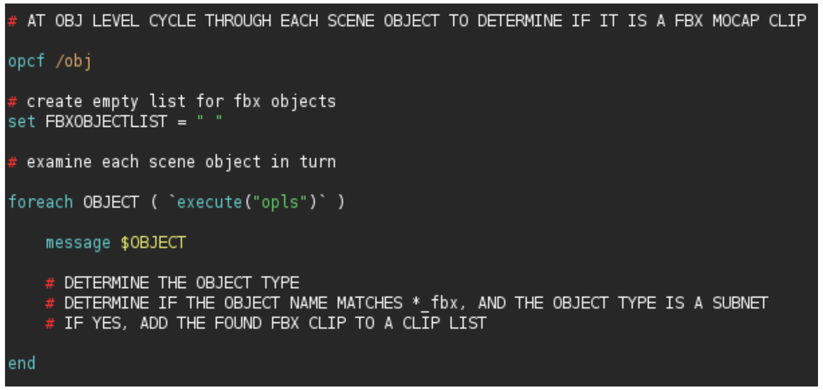
foreach OBJECT (`execute("opls")`)

message $OBJECT

end

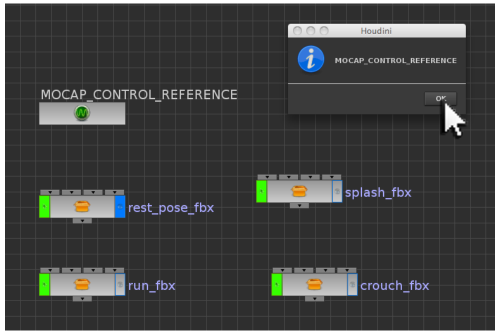
<this will cycle through each scene object in turn and message the end user>

NOTE: the `execute()` command can be used to run Hscript commands inside other HScript commands. In this context, Houdini will generate a list of all scene objects first; cycle through them in turn (using the for each loop) and automatically assigning them to a custom variable called $OBJECT.



```
# AT OBJ LEVEL CYCLE THROUGH EACH SCENE OBJECT TO DETERMINE IF IT IS A FBX MOCAP CLIP

opcf /obj

# create empty list for fbx objects
set FBXOBJECTLIST = " "

# examine each scene object in turn

foreach OBJECT ( `execute("opls")` )

    message $OBJECT

    # DETERMINE THE OBJECT TYPE
    # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT TYPE IS A SUBNET
    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST

end
```

NOTE: As the tasks for determining the object type, and if it matches a FBX clip are also required inside the foreach loop, these comments can be incorporated into the foreach loop before the final end command.



When the HScript is run, a message window appears for each scene object.

DETERMINING THE OBJECT TYPE

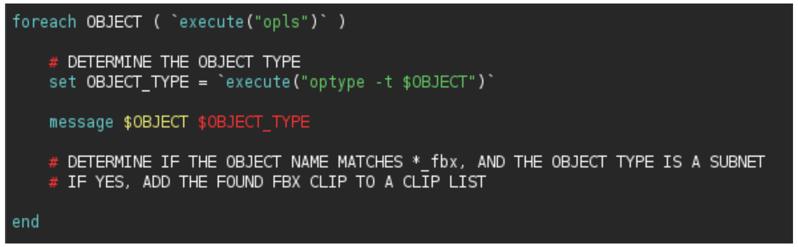
The Object Type can also be determined within this For Each Loop in a similar way by using the HScript command **optype** rather than **opls**.

NOTE: When developing HScripts, unknown commands can be found by typing help –k <keyword> into a Textport.

Under the task **DETERMINE THE OBJECT TYPE**, add the following commands:

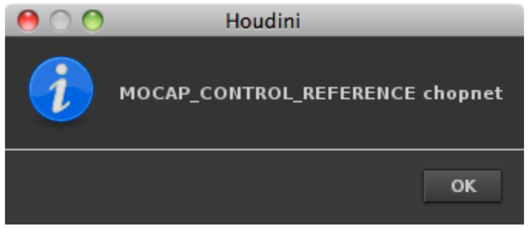
set OBJECT_TYPE = `execute("optype –t $OBJECT")`

message $OBJECT $OBJECT_TYPE



```
foreach OBJECT ( `execute("opls")` )

    # DETERMINE THE OBJECT TYPE
    set OBJECT_TYPE = `execute("optype -t $OBJECT")`

    message $OBJECT $OBJECT_TYPE

    # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT TYPE IS A SUBNET
    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
end
```

When the HScript is run again, a **message window** appears **declaring** both the **object name** currently being evaluated, plus its **object type**. **NOTE: Older message commands from earlier steps can be deleted**



```
MOCAP_CONTROL_REFERENCE chopnet
```

See HScript file mocap_fbx_stage1.hsc

PRÉCISING THE OBJECT NAME AND TYPE LIST

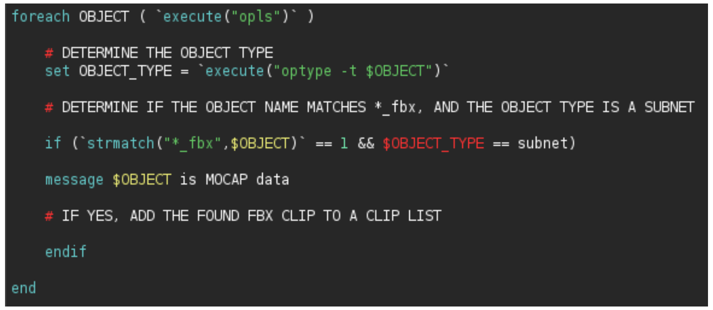
With the object name and object type being successfully returned as a temporary message; this information can now be used as a comparison. An **If Statement** can be embedded into the **For Each Loop** to compare if the **object name matches *_fbx** and its **type** is a **subnetwork**.

Under the task **DETERMINE IF OBJECT MATCHES…** add the following commands:

if (`strmatch("*_fbx",$OBJECT)` == 1 && $OBJECT_TYPE == subnet)

message $OBJECT is MOCAP data

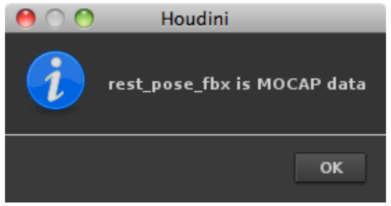
endif



```
foreach OBJECT ( `execute("opls")` )

    # DETERMINE THE OBJECT TYPE
    set OBJECT_TYPE = `execute("optype -t $OBJECT")`

    # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT TYPE IS A SUBNET

    if (`strmatch("*_fbx",$OBJECT)` == 1 && $OBJECT_TYPE == subnet)

    message $OBJECT is MOCAP data

    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST

    endif

end
```

NOTE: As before, the final task citing the additon of found FBX clips to a clip list can be left inside the If Statement.

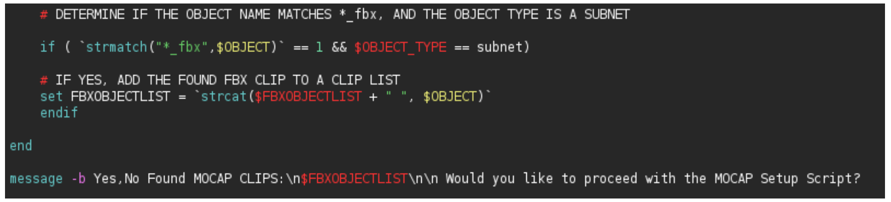
When the script is run again, a message window for each fbx subnetwork appears.



```
rest_pose_fbx is MOCAP data
```

INFORMING THE END USER ABOUT FOUND CLIPS

Currently a message window is return for each individual fbx clip. Rather than doing this, a compiled list of found fbx clips can be returned as a single message, with an option based instruction to the end user to proceed with the MOCAP Setup script or not. This will allow an end user to exit the script if any non fbx subnetworks inadvertently find their way into the compiled list; and also giving chance to rename such objects so that they don't appear in the list, before the script is re-run.



```
    # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT TYPE IS A SUBNET

    if ( `strmatch("*_fbx",$OBJECT)` == 1 && $OBJECT_TYPE == subnet)

    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
    set FBXOBJECTLIST = `strcat($FBXOBJECTLIST + " ", $OBJECT)`
    endif

end

message -b Yes,No Found MOCAP CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP Setup Script?
```

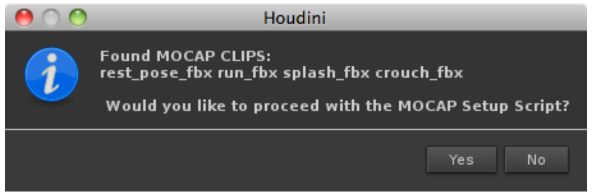
Under the IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST task, add the following commands:

set FBXOBJECTLIST = `strcat($FBXOBJECTLIST + " ", $OBJECT)`

This will append the names of any found fbx clips to the $FBXOBJECTLIST variable, leaving a space after each object addition. As a final step, this $FBXOBJECTLIST variable can be used to inform the end user about the found clips as a single message after the completion of the For Each Loop:

message -b Yes,No Found MOCAP CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP Setup Script?

When the script is run again, a single message window appears listing the found fbx clips, with Yes / No buttons to determine what the next step should be.



```
Found MOCAP CLIPS:
rest_pose_fbx run_fbx splash_fbx crouch_fbx

 Would you like to proceed with the MOCAP Setup Script?
```

NOTE: The use of \n in the message command line denotes a new line.
NOTE: Older message commands from earlier steps can be deleted.

MAKING YES / NO BUTTONS WORK

Currently either Yes or No will close the message window without affecting the script. As each button represents an Index Number (0 for Yes; 1 for No); these numbers can be used to continue the script if Yes is pressed, or exit the script if No is pressed. To do this, the entire message command needs to be wrapped in a run command, which can also facilitate its outcome as a variable:

set ANSWER = `run("message -b Yes,No Found MOCAP CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP Setup Script?")`

message $ANSWER

A test message command can also be added to verify the result of the $ANSWER return.

When the script is run again, the Found MOCAP CLIPS: message appears as before; however a new message window appears returning 0 or 1 dependant upon which option is chosen.

A new If Statement can be created to exit out of the script if No is pressed:

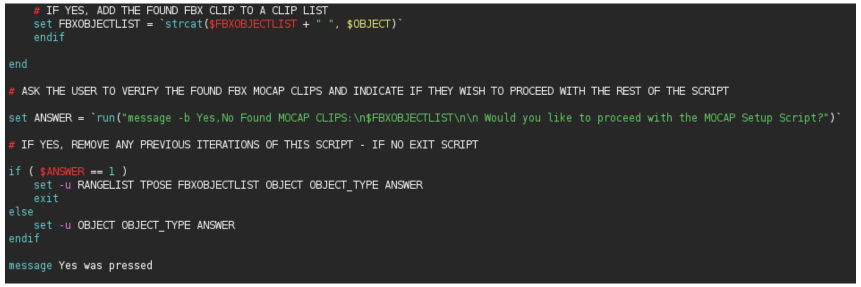
```
message $ANSWER

if ( $ANSWER == 1 )
        exit
endif

message Yes was pressed
```

A final message indicating that the script is still running if Yes is pressed can also be added to help verify this step.

UN-DECLARING VARIABLES
When variables are created they get held into memory. Currently the following variables have been created:

$RANGELIST
$TPOSE
$FBXOBJECTLIST
$OBJECT
$OBJECT_TYPE
$ANSWER

If the user decides to exit the script at this point, these variables will remain in memory. As part of script management, variables can be unset by using the set –u option. This can be done both for exiting the script, and for continuing the script (where only the key variables are kept).



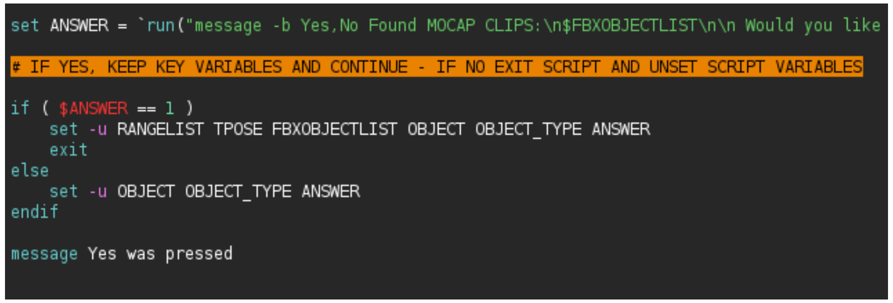
```
    # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
    set FBXOBJECTLIST = `strcat($FBXOBJECTLIST + " ", $OBJECT)`
    endif

end

# ASK THE USER TO VERIFY THE FOUND FBX MOCAP CLIPS AND INDICATE IF THEY WISH TO PROCEED WITH THE REST OF THE SCRIPT

set ANSWER = `run("message -b Yes,No Found MOCAP CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP Setup Script?")`

# IF YES, REMOVE ANY PREVIOUS ITERATIONS OF THIS SCRIPT - IF NO EXIT SCRIPT

if ( $ANSWER == 1 )
    set -u RANGELIST TPOSE FBXOBJECTLIST OBJECT OBJECT_TYPE ANSWER
    exit
else
    set -u OBJECT OBJECT_TYPE ANSWER
endif

message Yes was pressed
```

Modify this new If Statement to include set –u commands either deleting all the current variables created (if No is pressed) or keeping the key variables (if Yes is pressed):

```
if ( $ANSWER == 1 )
   set -u RANGELIST TPOSE FBXOBJECTLIST
   OBJECT OBJECT_TYPE ANSWER <on one line>
   exit
else
   set -u OBJECT OBJECT_TYPE ANSWER
endif

message Yes was pressed
```

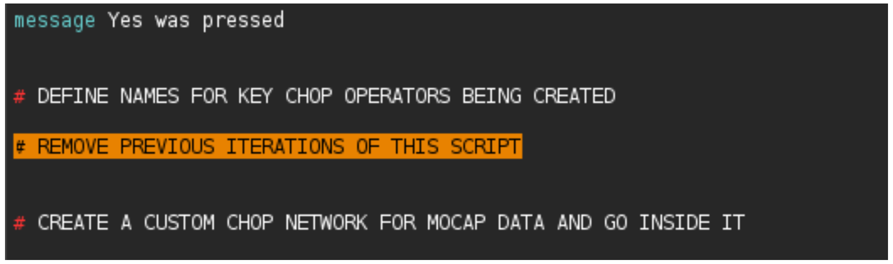
MODIFYING TASK COMMENTS

When developing a script, sometimes task comments need revision based on the script's development so far. In the current script, the IF YES… task comment makes reference to removing previous iterations of the script before proceeding. Currently the definitions of the MOCAP setup have not been declared, so removing previous iterations is not possible. **Modify the IF YES task comment, to better reflect the function of the If Statement.**



```
set ANSWER = `run("message -b Yes,No Found MOCAP CLIPS:\n$FBXOBJECTLIST\n\n Would you like

# IF YES, KEEP KEY VARIABLES AND CONTINUE - IF NO EXIT SCRIPT AND UNSET SCRIPT VARIABLES

if ( $ANSWER == 1 )
    set -u RANGELIST TPOSE FBXOBJECTLIST OBJECT OBJECT_TYPE ANSWER
    exit
else
    set -u OBJECT OBJECT_TYPE ANSWER
endif

message Yes was pressed
```

The task comment reference to removing previous iterations of the script can then be re-added after the task comment defining names of key CHOP operators.



```
message Yes was pressed

# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED

# REMOVE PREVIOUS ITERATIONS OF THIS SCRIPT


# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE IT
```

Keeping a script clear and legible through the use of task comments is vital, as it can help make sense of a script to a new user, or if the script is revisited at some point in the future. Good task commenting helps the readability of the script, so when it is revisited, the minutiae of the code does not have to be deciphered unless it needs to be improved upon.

See file mocap_fbx_stage2.hsc

THE MOCAP SETUP SCRIPT SO FAR…

```
# MOCAP SETUP HSCRIPT - TASK LIST
# DEFINE MOCAP SKELETON HIERARCHY NUMBERING
# DEFINE MOCAP REST POSE FBX NAME

##### USER DEFINED VARIBLES #####
# NOTES TO USER:
# Modify the RANGELIST variable to identify all beginning and end joint node
# chains of your FBX skeleton hierarchy...
# Modify the TPOSE variable to the name of your T-POSE / REST POSE fbx
# subnetwork

set RANGELIST = 1_6 7_11 12_19 20_24 25_27
set TPOSE = rest_pose_fbx

##### END USER DEFINED VARIABLES #####
```

```
# AT OBJ LEVEL CYCLE THROUGH EACH SCENE OBJECT TO DETERMINE IF
# IT IS A FBX MOCAP CLIP
opcf /obj
# create empty list for fbx objects
set FBXOBJECTLIST = " "

# examine each scene object in turn

foreach OBJECT ( `execute("opls")` )

   # DETERMINE THE OBJECT TYPE
   set OBJECT_TYPE = `execute("optype -t $OBJECT")`

   # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT
   # TYPE IS A SUBNET

   if ( `strmatch("*_fbx",$OBJECT)` == 1 && $OBJECT_TYPE == subnet)

   # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
   set FBXOBJECTLIST = `strcat($FBXOBJECTLIST + " ", $OBJECT)`
   endif
end

# ASK THE USER TO VERIFY THE FOUND FBX MOCAP CLIPS AND INDICATE
# IF THEY WISH TO PROCEED WITH THE REST OF THE SCRIPT
set ANSWER = `run("message -b Yes,No Found MOCAP
CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP
Setup Script?")`

# IF YES, KEEP KEY VARIABLES AND CONTINUE - IF NO EXIT SCRIPT AND
# UNSET SCRIPT VARIABLES

if ( $ANSWER == 1 )
   set -u RANGELIST TPOSE FBXOBJECTLIST OBJECT OBJECT_TYPE
ANSWER
   exit
else
   set -u OBJECT OBJECT_TYPE ANSWER
endif

message Yes was pressed

# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED
# REMOVE PREVIOUS ITERATIONS OF THIS SCRIPT
# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE
IT
      # CREATE A SWITCH CHOP
      # CREATE A RENAME CHOP
      # WIRE THE SWITCH CHOP INTO THE RENAME CHOP
      # ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP
```

# IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM CHOP NETWORK USING OBJECT CHAIN CHOPS

# LOCK THE OBJECT CHAIN CHOPS FOR THE REST POSE TO PREVENT RECURSION ERRORS

# WHEN ALL OBJECT CHAIN CHOPS FOR A MOCAP CLIP HAVE BEEN CREATED, MERGE THEM TOGETHER

# FEED THE MERGED OBJECT CHAIN CHOPS INTO THE SWITCH CHOP

# ACTIVATE THE ORANGE EXPORT FLAG ON THE RENAME CHOP

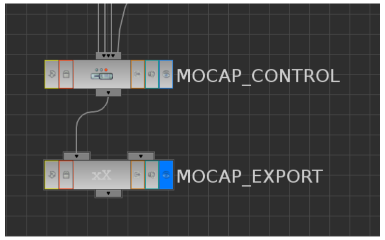
# TIDY UP THE AUTOMATICALLY GENERATED NODES

# DELETE SCRIPT VARIABLES

BUILDING THE MOCAP SETUP

The remainder of the script focuses on the actual construction of nodes relative to the FBX MOCAP clips found. This requires understanding of both the current key variables, and how this information relates to the nodes found in the MOCAP_CONTROL_REFERENCE CHOP Network.

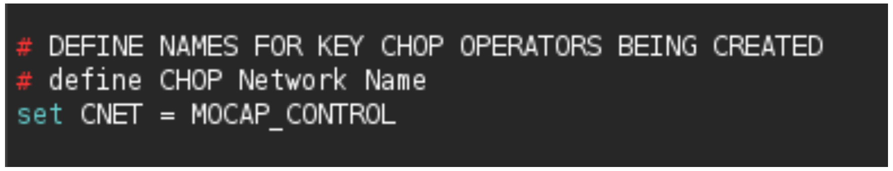
When determining which operators to create first, it is usually best to start from the end of the network and work backwards. In the case of the MOCAP_CONTROL_REFERENCE network, firstly the CHOP Network itself needs to be created, and inside it, a Switch CHOP and Rename CHOP can be created, and wired together.



This can be done using the commands **opadd** (**to create operators**), **opparm** (**to set parameters correctly**), **opwire** (**to wire nodes together**) and **opset** (**to activate node flags such as Display**).

As the CHOP Network itself may get regenerated when the script is run, it is a good idea to remove any previous instances of it (otherwise any new CHOP Network would get automatically renamed to MOCAP_CONTROL1, and as a result may break the script).

The first step of this process is therefore to create a key variable that defines the name of the CHOP Network.

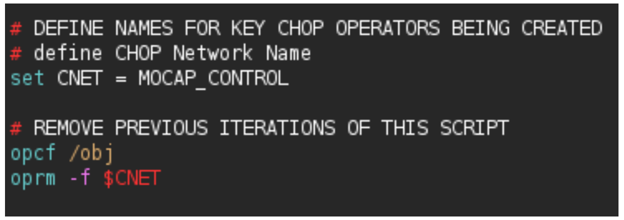


```
# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED
# define CHOP Network Name
set CNET = MOCAP_CONTROL
```

After the Task Comment defining names for key CHOP operators, use the set command to declare the name of the CHOP Network:

set CNET = MOCAP_CONTROL

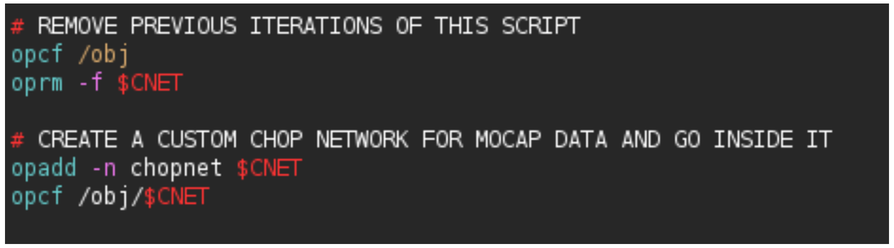
This variable can then be called to remove previous instances of the CHOP Network, as well as generating new versions of it.



```
# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED
# define CHOP Network Name
set CNET = MOCAP_CONTROL

# REMOVE PREVIOUS ITERATIONS OF THIS SCRIPT
opcf /obj
oprm -f $CNET
```

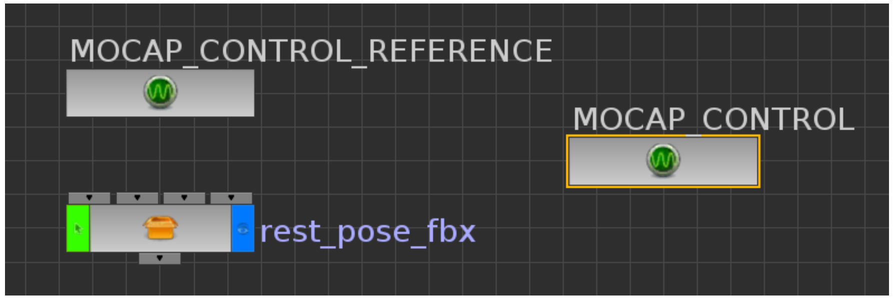
NOTE: the **oprm –f option** will delete an operator, but not return any messages if the operator cannot be found. For example, when the script is run for the first time, there is no MOCAP_CONTROL object to delete. If the –f flag was not specified, a message would be returned to the user citing that this object cannot be found.

After the deletion of previous iterations of the CHOP Network, a new CHOP Network can be generated.



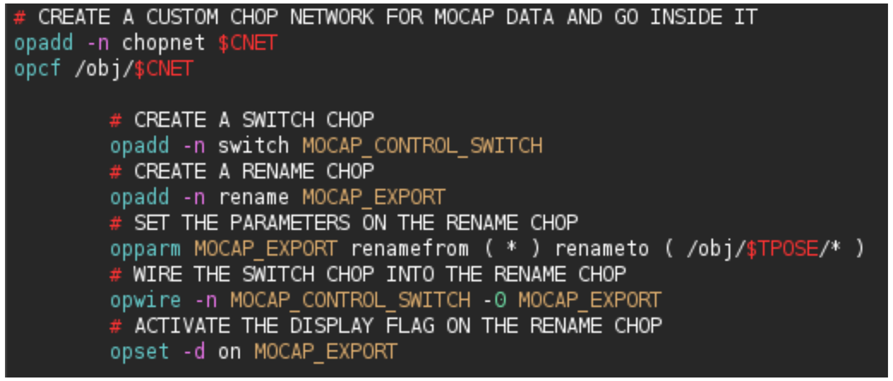
```
# REMOVE PREVIOUS ITERATIONS OF THIS SCRIPT
opcf /obj
oprm -f $CNET

# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE IT
opadd -n chopnet $CNET
opcf /obj/$CNET
```

When the script is run, now a **new CHOP Network** called MOCAP_CONTROL is created. Each time the script is run again, this object will get deleted (including its contents), and a new version of it will be generated.



NOTE: the opcf /obj/$CNET command will take Houdini inside the CHOP Network so that its internal CHOPs can be generated.

Once the MOCAP_CONTROL CHOP Network has been created, the end nodes of the CHOP Network can also be configured.

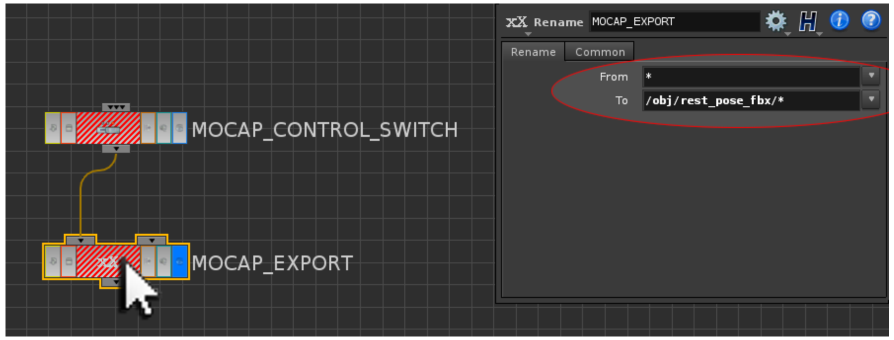


```
# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE IT
opadd -n chopnet $CNET
opcf /obj/$CNET

        # CREATE A SWITCH CHOP
        opadd -n switch MOCAP_CONTROL_SWITCH
        # CREATE A RENAME CHOP
        opadd -n rename MOCAP_EXPORT
        # SET THE PARAMETERS ON THE RENAME CHOP
        opparm MOCAP_EXPORT renamefrom ( * ) renameto ( /obj/$TPOSE/* )
        # WIRE THE SWITCH CHOP INTO THE RENAME CHOP
        opwire -n MOCAP_CONTROL_SWITCH -0 MOCAP_EXPORT
        # ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP
        opset -d on MOCAP_EXPORT
```

CREATE A SWITCH CHOP
opadd -n switch MOCAP_CONTROL_SWITCH
CREATE A RENAME CHOP
opadd -n rename MOCAP_EXPORT
SET THE PARAMETERS ON THE RENAME CHOP
opparm MOCAP_EXPORT renamefrom (*) renameto (/obj/$TPOSE/*)
WIRE THE SWITCH CHOP INTO THE RENAME CHOP
opwire -n MOCAP_CONTROL_SWITCH -0 MOCAP_EXPORT
ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP
opset -d on MOCAP_EXPORT

NOTE: the command **opparm** is being used to set the renaming of the channels so that they can be ported back onto the original **rest_pose_fbx** subnetwork.

When the script is run again, the end nodes of the CHOP Network are created and configured.



Rename MOCAP_EXPORT
From *
To /obj/rest_pose_fbx/*

As nothing is currently being wired into the MOCAP_CONTROL_SWITCH, errors may be returned on both the operators.

CREATING THE NETWORKS FOR THE SWITCH CHOP

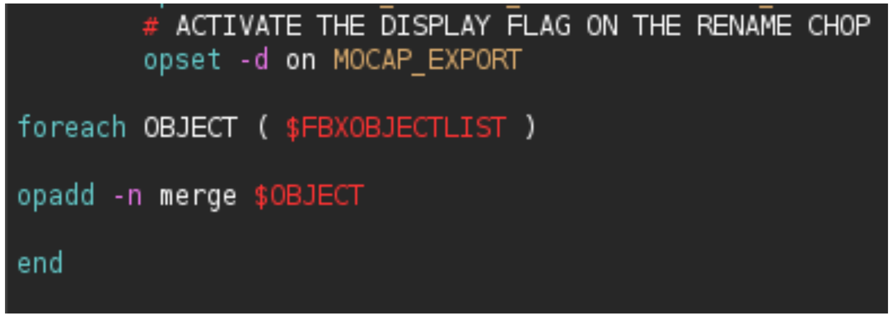
The mechanics of **For Each Loops** and **If Statements** can also be employed to procedurally generate the **Object Chain CHOP Networks** that will feed into the **Switch CHOP**. Again, the base principle for developing this aspect of the script is to begin with the nodes feeding into the Switch, before tackling any other nodes higher up the chain.

Examination of the **MOCAP_CONTROL_REFERENCE** network reveals that the **Object Chain CHOPs** are being fed into **Merge CHOPs** (**one for each found FBX MOCAP clip**).

A list of all the found **FBX MOCAP clips** is currently being stored in the **$FBXOBJECTLIST variable**. In this example, the list consists of:

rest_pose_fbx
run_fbx
splash_fbx
crouch_fbx

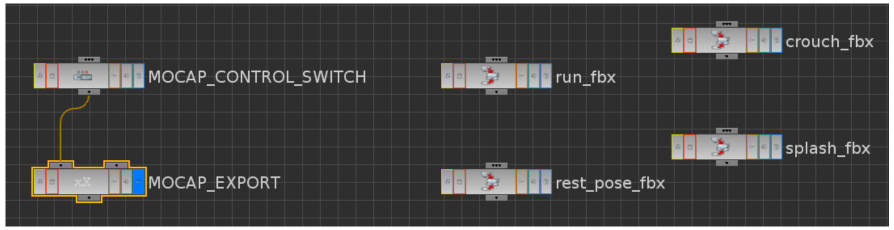
These names can be used as names for the created Merge CHOPS.



```
        # ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP
        opset -d on MOCAP_EXPORT

foreach OBJECT ( $FBXOBJECTLIST )

opadd -n merge $OBJECT

end
```

After activating the Display Flag on the Rename CHOP line of the script, create a **For Each Loop** using the **$FBXOBJECTLIST variable**. The **opadd** command can then be used to generate a **Merge CHOP** for each object found in the list.

When the script is run again, **Merge CHOPs** for each **FBX MOCAP** clip will be generated.



CREATING A COUNTING MECHANISM

As each of these **Merge CHOPs** will need to be **wired consecutively** into the **Switch CHOP** after creation, a **simple counting mechanism** can be created to facilitate this. Examination of the **opwire command Help Card** (**help opwire** typed into a Textport) reveals the following:

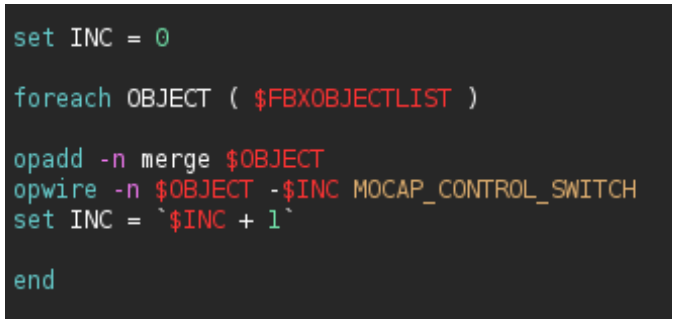
opwire box1 -0 merge1; opwire box2 -1 merge1

will connect box1 to the first input of merge1, box2 to the second input of merge1. It is recommended that for multiple input Ops like the merge SOP that the inputs are filled up consecutively.

This demonstrates that **for each Merge CHOP** created, they need to be **wired** into the **Switch CHOP** using an **Index Numbering System** where **-0** will wire into the **first input**, **-1** will wire into the **second input**, **-2** will wire into the **third input** etc.

If a **count variable** of **0** is declared **outside** of the **For Each Loop**, the For Each Loop mechanism can be also used to **increase** this **value incrementally** by **1** for each new object being assessed.

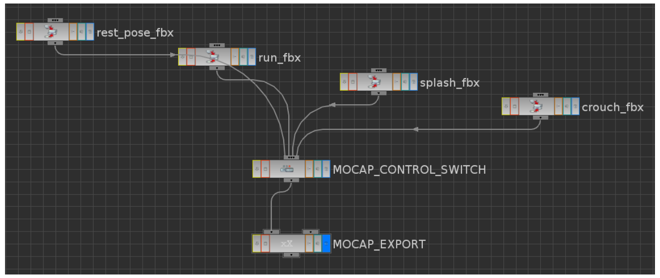
Before the start of the For Each Loop, use the **set command** to create a **counting variable** called **$INC** (increment).



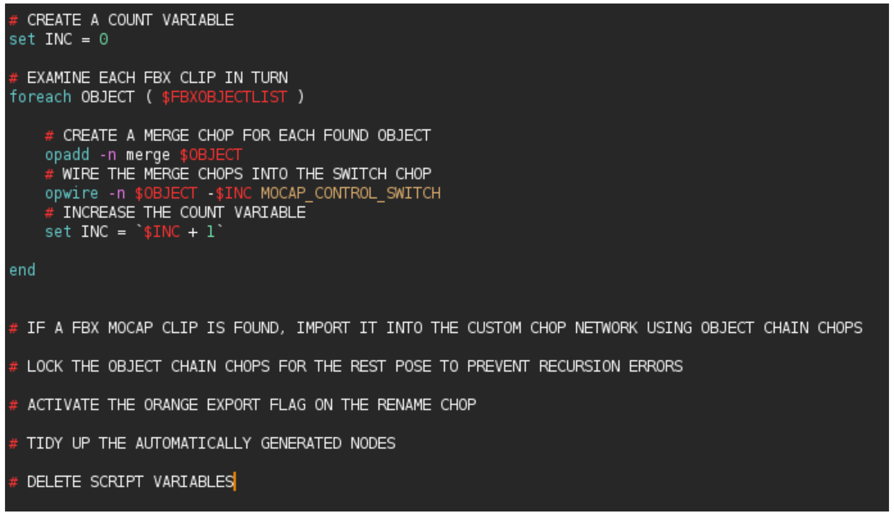
```
set INC = 0

foreach OBJECT ( $FBXOBJECTLIST )

opadd -n merge $OBJECT
opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
set INC = `$INC + 1`

end
```

This variable can then be called inside the For Each Loop, with the **opwire** command using **$INC's current value** to wire the **first Merge CHOP** into the **Switch CHOP**. When this has occurred, the **set** command can then be used to **reinitialize INC** with **`$INC + 1`**.

When the script is run again, the **Merge CHOPs** now get **procedurally wired** into the **Switch CHOP**.



As this part of the script touches upon some of the **remaining Comment Tasks**, these tasks can be modified, added to, and embedded next to the relevant areas of this part of the script.



```
# CREATE A COUNT VARIABLE
set INC = 0

# EXAMINE EACH FBX CLIP IN TURN
foreach OBJECT ( $FBXOBJECTLIST )

    # CREATE A MERGE CHOP FOR EACH FOUND OBJECT
    opadd -n merge $OBJECT
    # WIRE THE MERGE CHOPS INTO THE SWITCH CHOP
    opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
    # INCREASE THE COUNT VARIABLE
    set INC = `$INC + 1`

end

# IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM CHOP NETWORK USING OBJECT CHAIN CHOPS

# LOCK THE OBJECT CHAIN CHOPS FOR THE REST POSE TO PREVENT RECURSION ERRORS

# ACTIVATE THE ORANGE EXPORT FLAG ON THE RENAME CHOP

# TIDY UP THE AUTOMATICALLY GENERATED NODES

# DELETE SCRIPT VARIABLES
```

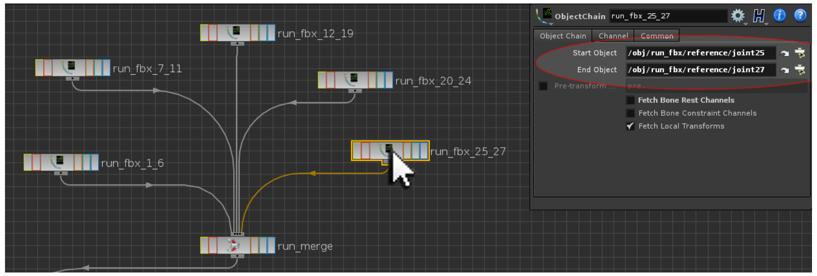
It is important to **review** and **modify Comment Tasks** as the script progresses, to ensure all parts of the developing script are properly annotated. See file **See file mocap_fbx_stage3.hsc**

COMPLETING THE SCRIPT

The final aspect of the script is to tackle the generation of **Object Chain CHOPs** that read in the FBX MOCAP Clip data. This can be achieved with understanding of how the information in the **$RANGELIST** variable can be manipulated. The **$RANGELIST** variable returns the following list:

1_6   7_11   12_19   20_24   25_27

Examination of the **MOCAP_CONTROL_REFERENCE CHOP Network** reveals that **Object Chain CHOPs** for each range value in the list needs to be created.



ObjectChain: run_fbx_25_27

Start Object: /obj/run_fbx/reference/joint25

End Object: /obj/run_fbx/reference/joint27

Fetch Bone Rest Channels

Fetch Bone Constraint Channels

✓ Fetch Local Transforms

The **parameters** for each **Chain CHOP** can also make reference to **$RANGELIST** with the **beginning number** of the range being used to define the **Start Object parameter's joint numbe**r, and the **last number** of the range being used to define the **End Object parameter's joint number**. This requires **extraction** of the **numeric components** of **each range value** (for example **1_6** needs to be extracted into **1** and **6** so these **individual numbers** can be assigned to the **Start Object** and **End Object parameters** accordingly).

IMPORTANT NOTE: The path listings for the Start Object and End Object parameters in the FBX MOCAP Clips run_fbx, crouch_fbx and splash_fbx also differ slightly from the path listings of the rest_pose_fbx clip. This may vary with other MOCAP clip data depending upon how it is exported from the MOCAP acquisition software, and may need to be factored when applying this current script to other Houdini MOCAP scenarios.

NESTING FOR EACH LOOPS

The **For Each Loop** creating each **Merge CHOP** can also be used to drive the generation of the **Object Chain CHOPs** required to complete the setup. This can be achieved by nesting a second For Each Loop inside the first one.

The pseudo code for this For Each Loop can help understanding of how a second For Each Loop can be nested inside it.

```
for each fbx clip found in the fbx clip list
{
        create a Merge CHOP and wire it into the Switch CHOP

        for each range found in the range list
        {
        create and configure and Object Chain CHOP
                wire it into the Merge CHOP
        }
        end
end
```

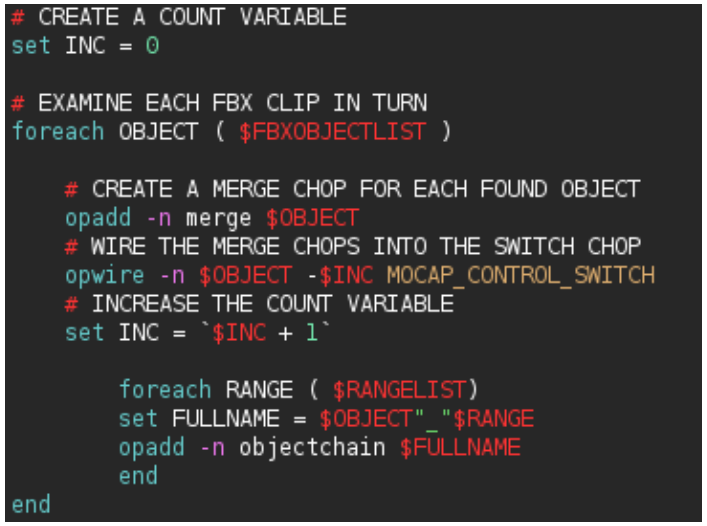
When the individual components of the fbx clip list, and the range list are piped into this pseudo code, the steps the process is taking become easier to determine.

```
for each fbx clip found in the fbx clip list [found rest_pose_fbx]
        {
                create a Merge CHOP and wire it into the Switch CHOP

                for each range found in the range list
                [found 1_6  7_11   12_19   20_24   25_27]
                {
                create and configure and Object Chain CHOP
                wire it into the Merge CHOP
                }
                end
        end
```

This pseudo code reveals that by nesting a For Each Loop inside and exsiting For Each Loop, it is possible to create all the individual Object Chain CHOPs required for each clip, before moving onto the next found clip in the list.

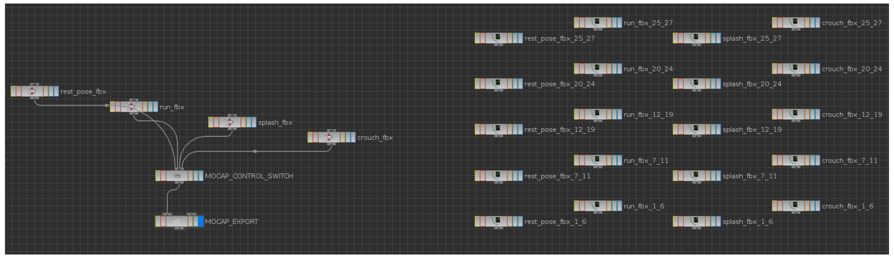
This principle can be applied to the HScript, where the name of the clip can also be utilized as a name for each found range. This can be done by creating a new variable called FULLNAME, which returns $OBJECT_$RANGE.

NOTE: For combining variables together as strings, the underscore (_) must be wrapped in double quotes to facilitate this combining.



```
# CREATE A COUNT VARIABLE
set INC = 0

# EXAMINE EACH FBX CLIP IN TURN
foreach OBJECT ( $FBXOBJECTLIST )

    # CREATE A MERGE CHOP FOR EACH FOUND OBJECT
    opadd -n merge $OBJECT
    # WIRE THE MERGE CHOPS INTO THE SWITCH CHOP
    opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
    # INCREASE THE COUNT VARIABLE
    set INC = `$INC + 1`

        foreach RANGE ( $RANGELIST)
        set FULLNAME = $OBJECT"_"$RANGE
        opadd -n objectchain $FULLNAME
        end
end
```

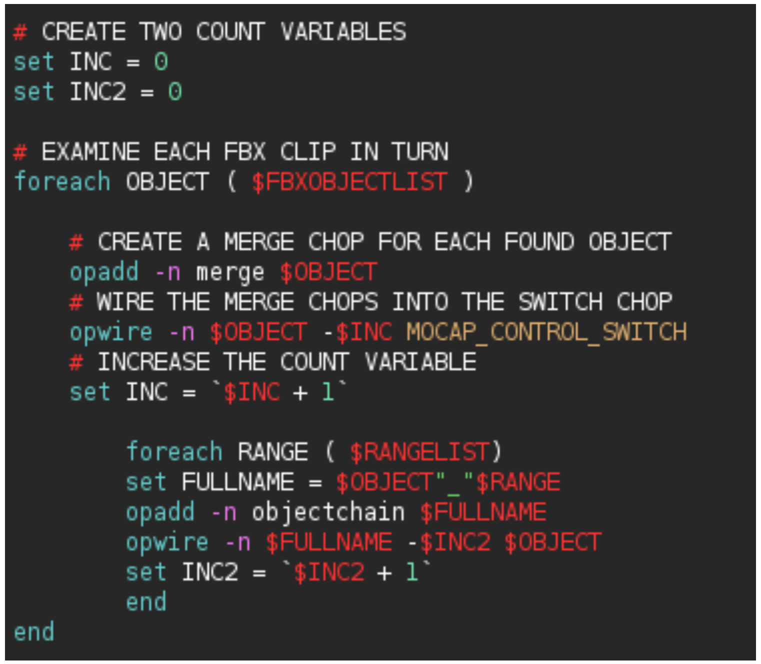
When the HScript is run again, individual Object Chain CHOPs get created and named for each clip and range found by the script.



WIRING THE OPERATORS TOGETHER

As before these individual Object Chain CHOPs need to be wired into their respective Merge CHOPs. This can be done in a similar way to before; however it needs to take place within the context of the nested For Each Loop.

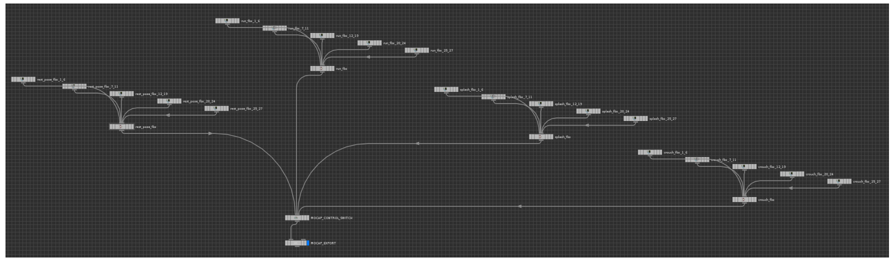
As the method for wiring the Merge CHOPs into the Switch CHOP was reliant upon a count variable to ensure the wiring happens correctly, a second count variable can be called within the nested For Each Loop in order to ensure the correct wiring of the Object Chain CHOPs takes place.



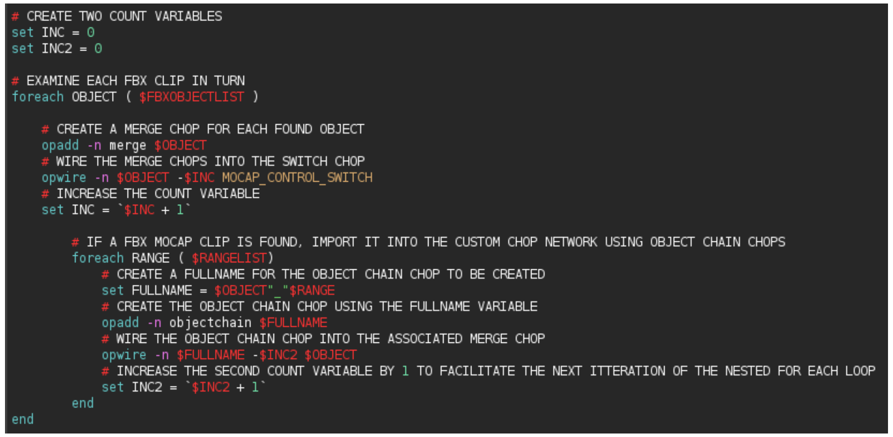
```
# CREATE TWO COUNT VARIABLES
set INC = 0
set INC2 = 0

# EXAMINE EACH FBX CLIP IN TURN
foreach OBJECT ( $FBXOBJECTLIST )

    # CREATE A MERGE CHOP FOR EACH FOUND OBJECT
    opadd -n merge $OBJECT
    # WIRE THE MERGE CHOPS INTO THE SWITCH CHOP
    opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
    # INCREASE THE COUNT VARIABLE
    set INC = `$INC + 1`

        foreach RANGE ( $RANGELIST)
        set FULLNAME = $OBJECT"_"$RANGE
        opadd -n objectchain $FULLNAME
        opwire -n $FULLNAME -$INC2 $OBJECT
        set INC2 = `$INC2 + 1`
        end
end
```

Create a second count variable ($INC2) underneath the first count variable, and modify the contents of the nested For Each Loop to also opwire each Object Chain CHOP into its respective Merge CHOP (using $INC2 as the wire input count number). Increase the value of $INC2 by adding 1, before the end of the nested For Each Loop.

When the HScript is run again, each Object Chain CHOP now gets wired correctly into each respective Merge CHOP.



As a final step, the nested For Each Loop can be annotated with Task Comments detailing what each line of the script is doing.

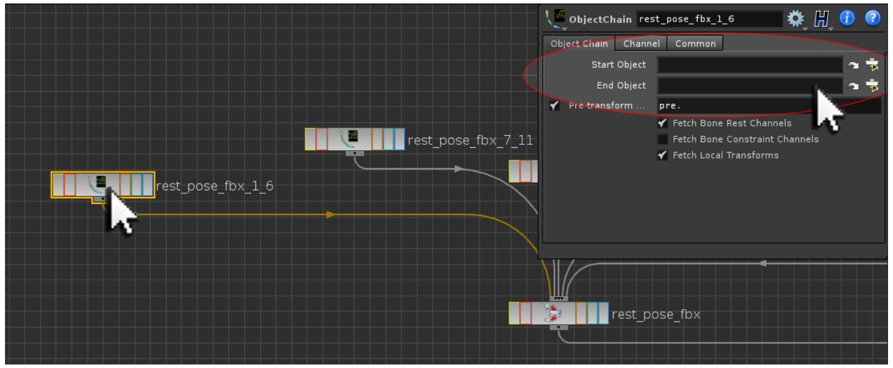


```
# CREATE TWO COUNT VARIABLES
set INC = 0
set INC2 = 0

# EXAMINE EACH FBX CLIP IN TURN
foreach OBJECT ( $FBXOBJECTLIST )

    # CREATE A MERGE CHOP FOR EACH FOUND OBJECT
    opadd -n merge $OBJECT
    # WIRE THE MERGE CHOPS INTO THE SWITCH CHOP
    opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
    # INCREASE THE COUNT VARIABLE
    set INC = `$INC + 1`

        # IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM CHOP NETWORK USING OBJECT CHAIN CHOPS
        foreach RANGE ( $RANGELIST)
            # CREATE A FULLNAME FOR THE OBJECT CHAIN CHOP TO BE CREATED
            set FULLNAME = $OBJECT"_"$RANGE
            # CREATE THE OBJECT CHAIN CHOP USING THE FULLNAME VARIABLE
            opadd -n objectchain $FULLNAME
            # WIRE THE OBJECT CHAIN CHOP INTO THE ASSOCIATED MERGE CHOP
            opwire -n $FULLNAME -$INC2 $OBJECT
            # INCREASE THE SECOND COUNT VARIABLE BY 1 TO FACILITATE THE NEXT ITTERATION OF THE NESTED FOR EACH LOOP
            set INC2 = `$INC2 + 1`
        end
end
```

See file **mocap_fbx_stage4.hsc**

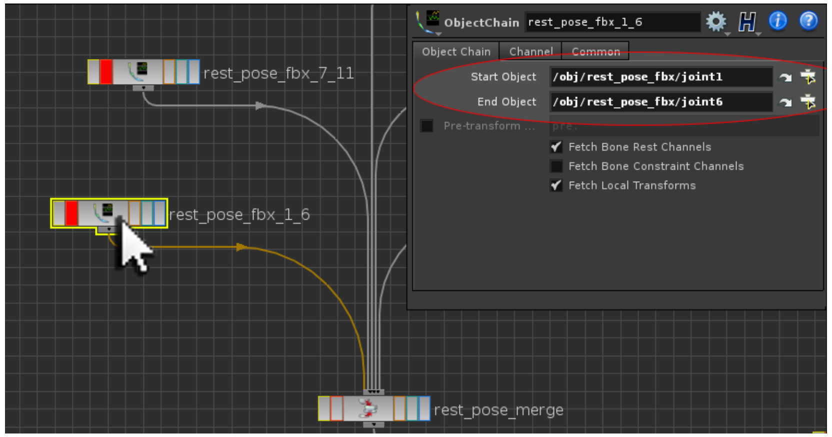
COMPLETING THE MOCAP SETUP HSCRIPT

Examination of the Object Chain CHOPs reveal that despite now being wired correctly, their parameters are empty.



These parameters can be procedurally configured, again within the context of the nested For Each Loop.

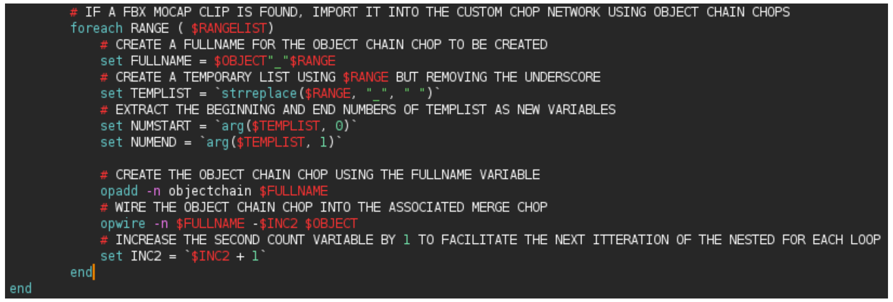
Examination of the MOCAP_CONTROL_REFERENCE CHOP Network reveals that the Start and End Object needs to be set to the range number of the Object Chain CHOP (for example 1_6); with the beginning number (for example 1) being fed into the Start Object parameter (as joint1), and the end number (for example 6) being fed into the End Object parameter (as joint6).



Start Object /obj/rest_pose_fbx/joint1
End Object /obj/rest_pose_fbx/joint6

This means whenever a range is identified from the $RANGELIST variable (for example 1_6); the components of $RANGE need to be broken into separate components (for example 1, _, 6) so they can be used as the joint numbers.

String manipulation of this type is a common phenomena within any scripting language, so it is worth understanding the nuance of this step.



```
        # IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM CHOP NETWORK USING OBJECT CHAIN CHOPS
        foreach RANGE ( $RANGELIST)
            # CREATE A FULLNAME FOR THE OBJECT CHAIN CHOP TO BE CREATED
            set FULLNAME = $OBJECT"_"$RANGE
            # CREATE A TEMPORARY LIST USING $RANGE BUT REMOVING THE UNDERSCORE
            set TEMPLIST = `strreplace($RANGE, "_", " ")`
            # EXTRACT THE BEGINNING AND END NUMBERS OF TEMPLIST AS NEW VARIABLES
            set NUMSTART = `arg($TEMPLIST, 0)`
            set NUMEND = `arg($TEMPLIST, 1)`

            # CREATE THE OBJECT CHAIN CHOP USING THE FULLNAME VARIABLE
            opadd -n objectchain $FULLNAME
            # WIRE THE OBJECT CHAIN CHOP INTO THE ASSOCIATED MERGE CHOP
            opwire -n $FULLNAME -$INC2 $OBJECT
            # INCREASE THE SECOND COUNT VARIABLE BY 1 TO FACILITATE THE NEXT ITTERATION OF THE NESTED FOR EACH LOOP
            set INC2 = `$INC2 + 1`
        end
end
```

Modify the contents of the nested For Each Loop to create a Temporary List ($TEMPLIST) that extracts the numbers found in $RANGE (for example 1_6) as individual numbers stored as new variables. This can be done using the H-Expressions ‘**strreplace’** (string replacement) and ‘**arg’** (to identify the arguments found in a variable).

NOTE: The combination of H-Expressions and the HScript Language is again a common phenomenon, where H-Expressions can be embedded in HScripts to add functionality.

```
foreach RANGE ( $RANGELIST)
    # CREATE A FULLNAME FOR THE OBJECT CHAIN CHOP
      set FULLNAME = $OBJECT"_"$RANGE
    # CREATE A TEMPORARY LIST USING $RANGE BUT REMOVING THE UNDERSCORE
     set TEMPLIST = `strreplace($RANGE, "_", " ")`
     # EXTRACT THE BEGINNING AND END NUMBERS OF TEMPLIST AS NEW VARIABLES
     set NUMSTART = `arg($TEMPLIST, 0)`
     set NUMEND = `arg($TEMPLIST, 1)`
```

Now each beginning and end number found in each $RANGE are stored in their own variables ($NUMSTART and $NUMEND). These variables can now be called to set the parameters of each Object Chain CHOP correctly.

IMPORTANT NOTE: Examination of the reference CHOP network also reveals that there are path listing differences between the rest_pose_fbx Object Chain CHOPs and the Object Chain CHOPS for the other clips.

The rest_pose_fbx Object Chain CHOPs have a Start and End Object parameter paths of:

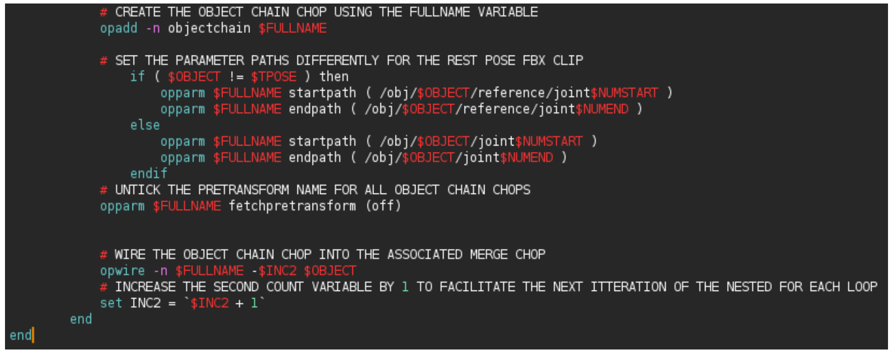
Start Object	**/obj/rest_pose_fbx/jointx**
End Object	**/obj/rest_pose_fbx/jointy**

Whereas the remaining Object Chain CHOPs have a a Start and End Object parameter paths of:

Start Object	**/obj/run_fbx/reference/jointx**
End Object	**/obj/run_fbx/reference/jointy**

This can also be anticipated in the HScript with an **If Statement** specifying that for the rest_pose_fbx Object Chain CHOPs set the shorter path, but for all other Object Chain CHOPs set the longer path.

Within the nested For Each Loop, after the creation of the Object Chain CHOP, use an If Statement to set different path listings for the rest_pose_fbx Object Chain CHOPs.



```
            # CREATE THE OBJECT CHAIN CHOP USING THE FULLNAME VARIABLE
            opadd -n objectchain $FULLNAME

            # SET THE PARAMETER PATHS DIFFERENTLY FOR THE REST POSE FBX CLIP
                if ( $OBJECT != $TPOSE ) then
                    opparm $FULLNAME startpath ( /obj/$OBJECT/reference/joint$NUMSTART )
                    opparm $FULLNAME endpath ( /obj/$OBJECT/reference/joint$NUMEND )
                else
                    opparm $FULLNAME startpath ( /obj/$OBJECT/joint$NUMSTART )
                    opparm $FULLNAME endpath ( /obj/$OBJECT/joint$NUMEND )
                endif
            # UNTICK THE PRETRANSFORM NAME FOR ALL OBJECT CHAIN CHOPS
            opparm $FULLNAME fetchpretransform (off)


            # WIRE THE OBJECT CHAIN CHOP INTO THE ASSOCIATED MERGE CHOP
            opwire -n $FULLNAME -$INC2 $OBJECT
            # INCREASE THE SECOND COUNT VARIABLE BY 1 TO FACILITATE THE NEXT ITTERATION OF THE NESTED FOR EACH LOOP
            set INC2 = `$INC2 + 1`
        end
end
```

CREATE THE OBJECT CHAIN CHOP USING THE FULLNAME VARIABLE
opadd -n objectchain $FULLNAME

SET THE PARAMETER PATHS DIFFERENTLY FOR THE REST POSE FBX CLIP
if ($OBJECT != $TPOSE) then

opparm $FULLNAME startpath (/obj/$OBJECT/reference/joint$NUMSTART)

opparm $FULLNAME endpath (/obj/$OBJECT/reference/joint$NUMEND)

else

opparm $FULLNAME startpath (/obj/$OBJECT/joint$NUMSTART)

opparm $FULLNAME endpath (/obj/$OBJECT/joint$NUMEND)

endif

As a final step, the Pre-Transform Name option for each Object Chain CHOP can also be deactivated.

UNTICK THE PRETRANSFORM NAME FOR ALL OBJECT CHAIN CHOPS
opparm $FULLNAME fetchpretransform (off)

When the script is run again, the parameters for each Object Chain CHOP are now correctly configured.

FINAL STEPS

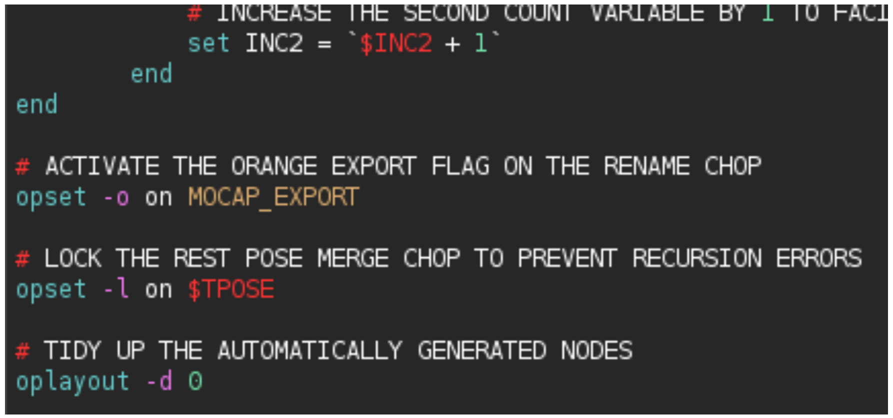
To prevent recursion errors from occurring (ie to stop the overwriting of the rest_pose_fbx object with the exported MOCAP CHOP data), the rest_pose_fbx Object Chain CHOP nodes can be locked. The simplest way to do this, is not locking the rest_pose_fbx Object Chain CHOPs themselves but rather to **lock the rest_pose_fbx Merge CHOP** instead.

IMPORTANT NOTE: For the network to read in the MOCAP data correctly, the **Orange Export Flag** on the **MOCAP_EXPORT Rename CHOP** needs to be **activated before** the **rest_pose_fbx Merge CHOP is locked**. If these steps are done the other way around, the Rename CHOP will return a warning error citing this operator has no effect.

After the **end of the For Each Loops**, use the **opset** command to activate the **Orange Export Flag on the Rename CHOP** to prevent recursion errors.

After activating the Orange Export Flag, **lock the rest_pose_fbx Merge CHOP** (the **$TPOSE** variable can be called to identify this node correctly).

Finally, use the **oplayout** command to redraw the automatically generated nodes.



```
            # INCREASE THE SECOND COUNT VARIABLE BY 1 TO FACI
            set INC2 = `$INC2 + 1`
        end
end

# ACTIVATE THE ORANGE EXPORT FLAG ON THE RENAME CHOP
opset -o on MOCAP_EXPORT

# LOCK THE REST POSE MERGE CHOP TO PREVENT RECURSION ERRORS
opset -l on $TPOSE

# TIDY UP THE AUTOMATICALLY GENERATED NODES
oplayout -d 0
```

When the HScript is run again, the MOCAP_CONTROL CHOP Network gets fully configured, and the MOCAP data can now be used in a production context.

NOTE: When a HScript reaches its end, any extraneous variables get automatically delelted.

See file **mocap_fbx_complete.hsc**

FURTHER SCRIPT DEVELOPMENT

A slightly higher-level implementation of this script could also include the automatic importing of fbx clips as part of this HScript. Typing **help –k fbx** into a **Textport** will return any fbx-associated commands that can be investigated to facilitate this.

HScript and Python

HScript and Python basically give the same functionality within Houdini. For someone new to scripting, using HScript is much cleaner and simpler as a scripting language, compared with the overly verbose Python language.

When developing a script, a handmade network can be examined and converted into a script as per this example.

When progressing into the Python scripting language, a developed HScript can also be used to start deciphering the Python commands needed to recreate it.

Included in the lecture files is a Python version of the MOCAP Setup Script for your reference. Take a moment to compare and contrast the two scripting languages (and see how verbose Python is compared to HScript). Please note that this Python version of the script was developed by a MScCAVE1112 student and has not been formally tested; however should get you up and running with core Python commands if you need to start learning them.

```
# MOCAP SETUP HSCRIPT – THE COMPLETED SCRIPT

# DEFINE MOCAP SKELETON HIERARCHY NUMBERING
# DEFINE MOCAP REST POSE FBX NAME

##### USER DEFINED VARIBLES #####

# NOTES TO USER:
# Modify the RANGELIST variable to identify all beginning
# and end joint node chains of your FBX skeleton heirachy...
# Modify the TPOSE variable to the name of your
# T-POSE / REST POSE fbx subnetwork

set RANGELIST = 1_6 7_11 12_19 20_24 25_27
set TPOSE = rest_pose_fbx

##### END USER DEFINED VARIABLES #####

# AT OBJ LEVEL CYCLE THROUGH EACH SCENE OBJECT TO DETERMINE IF
IT IS A FBX MOCAP CLIP

opcf /obj

# create empty list for fbx objects
set FBXOBJECTLIST = " "

# examine each scene object in turn
foreach OBJECT ( `execute("opls")` )

   # DETERMINE THE OBJECT TYPE
   set OBJECT_TYPE = `execute("optype -t $OBJECT")`

   # DETERMINE IF THE OBJECT NAME MATCHES *_fbx, AND THE OBJECT
TYPE IS A SUBNET

   if ( `strmatch("*_fbx",$OBJECT)` == 1 && $OBJECT_TYPE == subnet)

   # IF YES, ADD THE FOUND FBX CLIP TO A CLIP LIST
   set FBXOBJECTLIST = `strcat($FBXOBJECTLIST + " ", $OBJECT)`
   endif

end

# ASK THE USER TO VERIFY THE FOUND FBX MOCAP CLIPS AND INDICATE
IF THEY WISH TO PROCEED WITH THE REST OF THE SCRIPT

set ANSWER = `run("message -b Yes,No Found MOCAP
CLIPS:\n$FBXOBJECTLIST\n\n Would you like to proceed with the MOCAP Setup
Script?")`

# IF YES, KEEP KEY VARIABLES AND CONTINUE - IF NO EXIT SCRIPT AND
UNSET SCRIPT VARIABLES
```

```
if ( $ANSWER == 1 )
   set -u RANGELIST TPOSE FBXOBJECTLIST OBJECT OBJECT_TYPE
ANSWER
   exit
else
   set -u OBJECT OBJECT_TYPE ANSWER
endif

message Yes was pressed


# DEFINE NAMES FOR KEY CHOP OPERATORS BEING CREATED
# define CHOP Network Name
set CNET = MOCAP_CONTROL

# REMOVE PREVIOUS ITERATIONS OF THIS SCRIPT
opcf /obj
oprm -f $CNET

# CREATE A CUSTOM CHOP NETWORK FOR MOCAP DATA AND GO INSIDE
IT
opadd -n chopnet $CNET
opcf /obj/$CNET

      # CREATE A SWITCH CHOP
      opadd -n switch MOCAP_CONTROL_SWITCH
      # CREATE A RENAME CHOP
      opadd -n rename MOCAP_EXPORT
      # SET THE PARAMETERS ON THE RENAME CHOP
      opparm MOCAP_EXPORT renamefrom ( * ) renameto ( /obj/$TPOSE/* )
      # WIRE THE SWITCH CHOP INTO THE RENAME CHOP
      opwire -n MOCAP_CONTROL_SWITCH -0 MOCAP_EXPORT
      # ACTIVATE THE DISPLAY FLAG ON THE RENAME CHOP
      opset -d on MOCAP_EXPORT


# CREATE TWO COUNT VARIABLES
set INC = 0
set INC2 = 0

# EXAMINE EACH FBX CLIP IN TURN
foreach OBJECT ( $FBXOBJECTLIST )

   # CREATE A MERGE CHOP FOR EACH FOUND OBJECT
   opadd -n merge $OBJECT
   # WIRE THE MERGE CHOPS INTO THE SWITCH CHOP
   opwire -n $OBJECT -$INC MOCAP_CONTROL_SWITCH
   # INCREASE THE COUNT VARIABLE
   set INC = `$INC + 1`

      # IF A FBX MOCAP CLIP IS FOUND, IMPORT IT INTO THE CUSTOM
CHOP NETWORK USING OBJECT CHAIN CHOPS
      foreach RANGE ( $RANGELIST)
```

```
        # CREATE A FULLNAME FOR THE OBJECT CHAIN CHOP TO BE
CREATED
        set FULLNAME = $OBJECT"_"$RANGE
        # CREATE A TEMPORARY LIST USING $RANGE BUT REMOVING THE
UNDERSCORE
        set TEMPLIST = `strreplace($RANGE, "_", " ")`
        # EXTRACT THE BEGINNING AND END NUMBERS OF TEMPLIST AS
NEW VARIABLES
        set NUMSTART = `arg($TEMPLIST, 0)`
        set NUMEND = `arg($TEMPLIST, 1)`

        # CREATE THE OBJECT CHAIN CHOP USING THE FULLNAME
VARIABLE
        opadd -n objectchain $FULLNAME

        # SET THE PARAMETER PATHS DIFFERENTLY FOR THE REST POSE
FBX CLIP
          if ( $OBJECT != $TPOSE ) then
            opparm $FULLNAME startpath (
/obj/$OBJECT/reference/joint$NUMSTART )
            opparm $FULLNAME endpath (
/obj/$OBJECT/reference/joint$NUMEND )
          else
            opparm $FULLNAME startpath ( /obj/$OBJECT/joint$NUMSTART )
            opparm $FULLNAME endpath ( /obj/$OBJECT/joint$NUMEND )
          endif

        # UNTICK THE PRETRANSFORM NAME FOR ALL OBJECT CHAIN
CHOPS
        opparm $FULLNAME fetchpretransform (off)


        # WIRE THE OBJECT CHAIN CHOP INTO THE ASSOCIATED MERGE
CHOP
        opwire -n $FULLNAME -$INC2 $OBJECT
        # INCREASE THE SECOND COUNT VARIABLE BY 1 TO FACILITATE
THE NEXT ITTERATION OF THE NESTED FOR EACH LOOP
        set INC2 = `$INC2 + 1`
      end
end

# ACTIVATE THE ORANGE EXPORT FLAG ON THE RENAME CHOP
opset -o on MOCAP_EXPORT

# LOCK THE REST POSE MERGE CHOP TO PREVENT RECURSION ERRORS
opset -l on $TPOSE

# TIDY UP THE AUTOMATICALLY GENERATED NODES
oplayout -d 0
```