mGear: Maya Rigging Framework Documentation

Release 4.0.3

Jeremie Passerin, Miquel Campos

Jan 20, 2024

CONTENTS

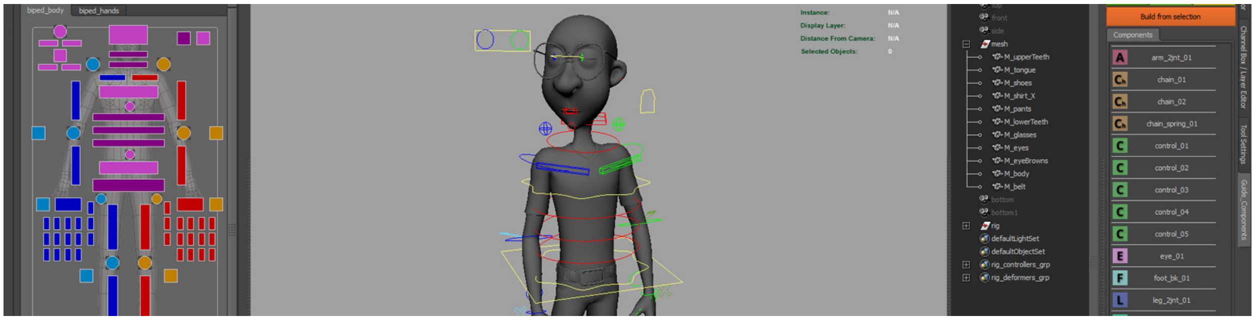


Fig. 1: mGear

mGear website Join mGear Community Forum

Contents:

CHAPTER

ONE

OVERVIEW

mGear is a rigging and animation framework for Autodesk Maya. mGear provides a set of convenient modules, tools and c++ solvers to streamline the development of rigging and animation tools.

Originally mGear was design and develope by Jeremie Passerin , since 2013 Miquel Campos continued the project and evolved to the current design. From 2018 mGear is developed by the mGear Dev team

Join mGear Community Forum

Big Thanks! to all contributors

Note:

from Miquel:

Big thanks to Jeremie Passerin , he is the real master behind mGear design. I am just following his steps and continuing what he started.

Also special big thanks to Ingo Clemens for the OSX solver compilation and Gaetan Guidet for the Linux solver compilation and guidance in my C++ desert of knowledge.

Thanks to Chad Vernon , for his fantastic tutorials and open-source tools. Some of mGear's parts wouldn't be possible without it.

And thanks!! to all the people who help me and support this project :)

Warning: I can't promise any kind of support and this tool is provided "AS IS". Please read the LICENSE for more information.

However, I am committed to continue developing mGear. So please, do not hesitate contact us if you found any bug or possible improvement.

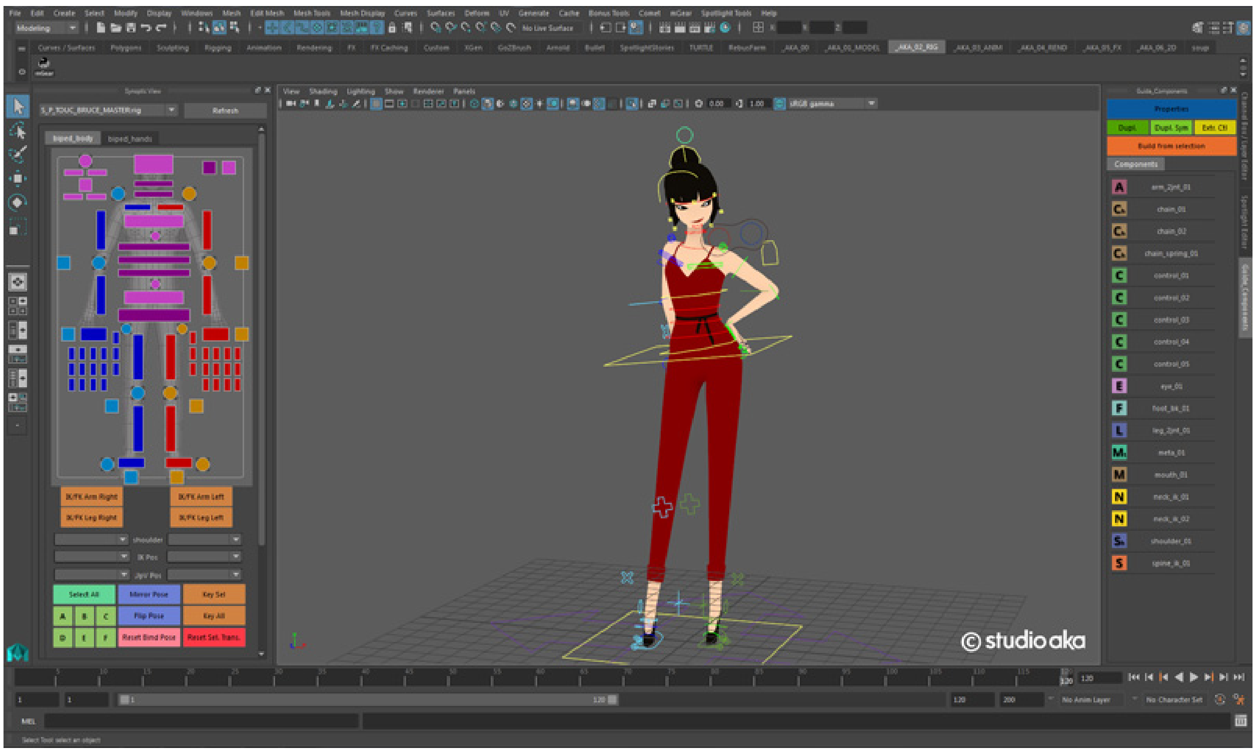


Fig. 1: mGear have been used at Studio AKA for San Pellegrino 'Dining By Starlight'. Rigged by Adam Avery

CHAPTER

TWO

QUICK START

This is a quick guide to get you up and running with mGear's Shifter. Shifter is the character rigging part of mGear, with which you can combine various components (arms, legs, spines, chains etc.) to rig most kinds of characters, creatures and mechanical contraptions. Shifter currently ships with over 40 different components, and more are constantly being added. Here we'll only focus on building a basic human biped.

2.1 Installation

To install mGear on your computer:

1) Download the latest mGear release from here.
2) Unzip it
3) Copy the content of the mGear **release** folder to your maya/modules folder:

a) Windows: Users<username>DocumentsMayamodules

b) Linux: ~/maya/modules

c) Mac OS X: ~/maya/modules

4) Start Maya

You should now have an mGear menu in Maya.

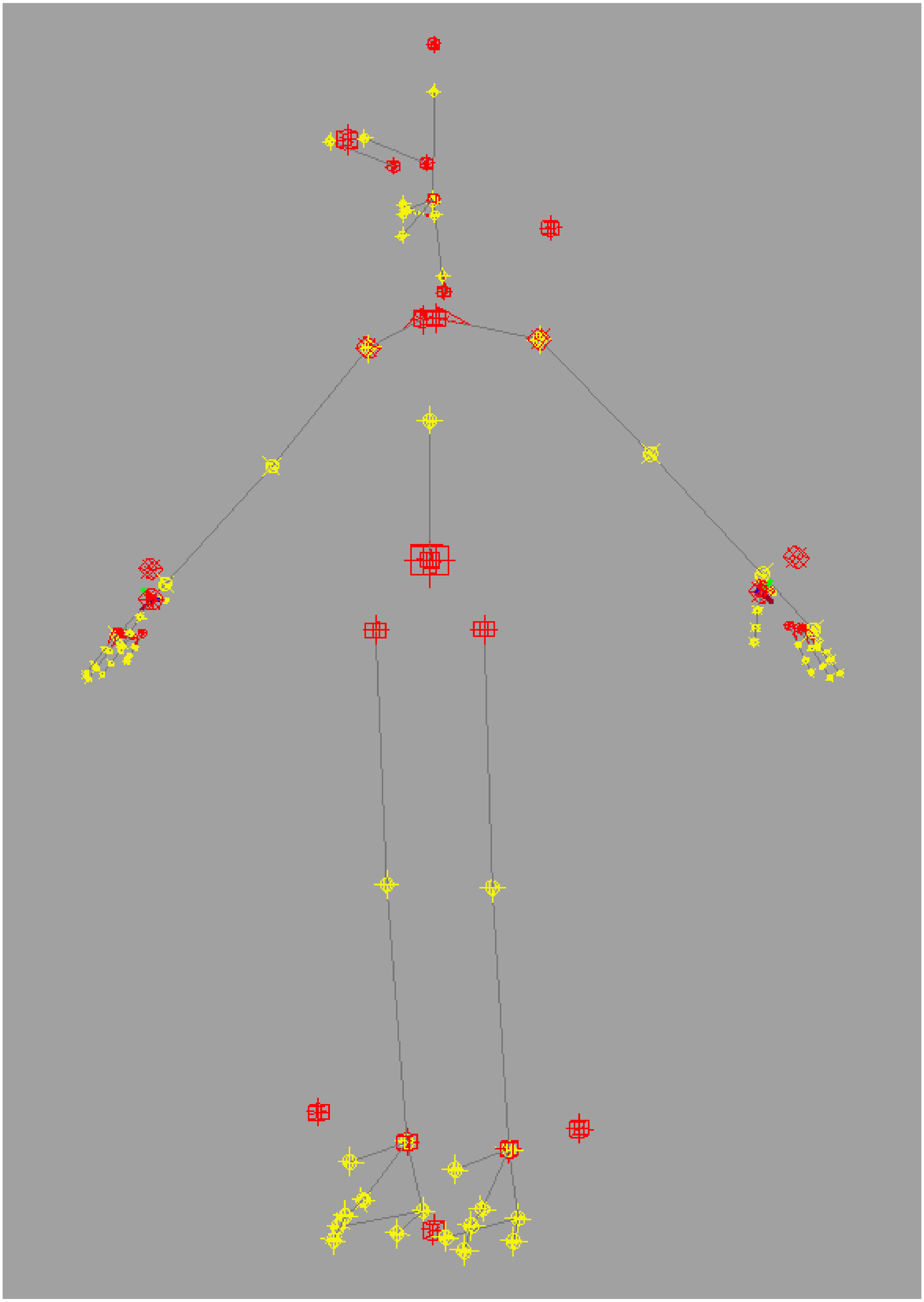
2.2 Rigging a Biped

Rigging a biped character is very quick and easy with mGear's Shifter, as it comes with ready made guide templates for both a standard biped and a quadruped. The steps involved in building a rig are:

1) Load the guide template
2) Position the guides to match your characters anatomy
3) Build the animation rig
4) Adjust the control curves if needed
5) Skin the character to the deformer joints
6) Animate!

2.2.1 Build Guides from a Template

Start by selecting mGear>Shifter>Guide Template Samples>Biped Template from the menu.



This builds all the guides we need to build a biped rig. The biped guides consist of multiple components that we will position to match the proportions of our character. Each guide component consists of a:

- **root:** (red cubes) These are the top level node of each component.
- **position:** (yellow spheres) These are the positions of each joint (e.g. elbow, wrist etc).
- **blade:** (red triangular wedges) These controls the up axis of some components.
- **curve:** (gray lines) These are only for reference, to show how evrything connects together.

Note: All components have a **root**, but not all have **position**, **blade** or **curve**.

You can position, rotate and scale the **root** and **position** guides to fit your character, and use the Blade Roll Offset attribute to adjust the **blade** guides.

Since we don't want to do double work, we can start by deleting the right side **root** guides; the **eye_R0_root**, **shouler_R0_root** and **leg_R0_root**. Once we're happy with the placement of the guides on the left side, we can simply use mGear>Shifter>Duplicate Sym to mirror them over to the right side.

Note: You can test your rig at any point, by simply selecting the **guide** node from the Outliner and running mGear>Shifter>Build from Selection. The will bulid the complete animation rig. Once finished testing, you simply delete the **rig** node from the Outliner, and continue adjusting your guides.

2.2.2 Positioning the Guides

Start by selecting the top **guide** node in the oulinter and scale it so the hip is in the correct place. Then work your way down the hierarchy, from the spine and out to the different bodyparts, positioning each one in turn. If you need to re-position a parent guide, after positioning it's children, you can temporarily de-parent the children, while moving the parent.

Note: We'll only be covering what you need to get your character rigged here. For details on each component, see the *Shifter Component Reference*. (TODO! Broken link)

Settings

Each **guide** has settings you can adjust to change the behaviour of that component. To access the settings of **guide**, simply got the mGear>Shifter>Settings menu. For this quick start, we won't be touching the settings, but it's good to know they are there.

Host Guides

You'll notice that next to the hand and feet, and above the head and one of the shouder there are **root** nodes that are seemingly not connected to anything. These are used to position Host controllers from which you can control things like IK/FK blending, arm/leg stretching and more.

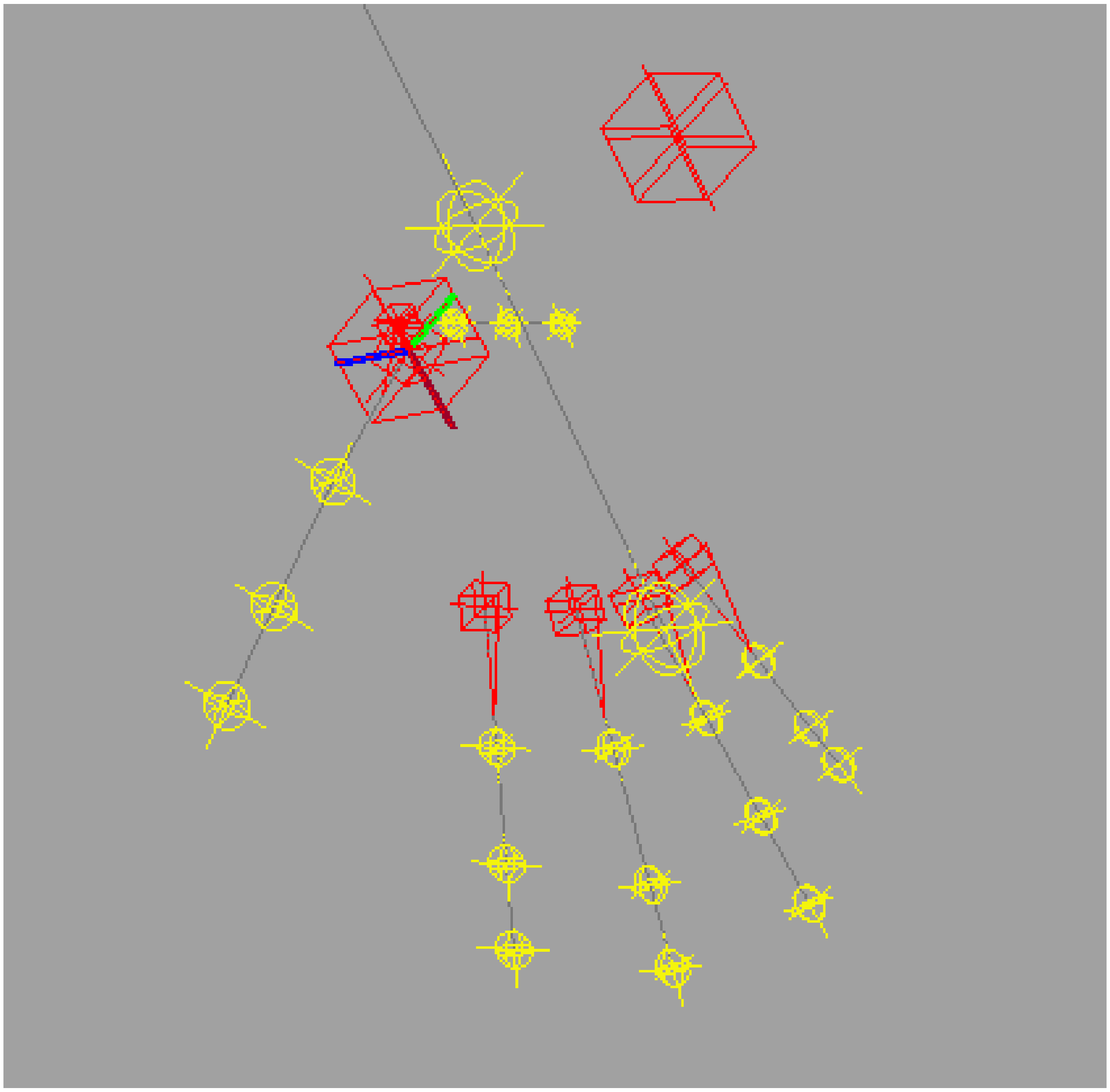
Hip and Spine

There are two **root** nodes at the hip. The bigger one controls the position of the hips and where the controler that moves the entire upper body will be, while the smaller one sets the position of the controller that only moves the hips, keeping the torso in place. The yellow **position** guide controls the length of the flexible part of the spine.

Neck

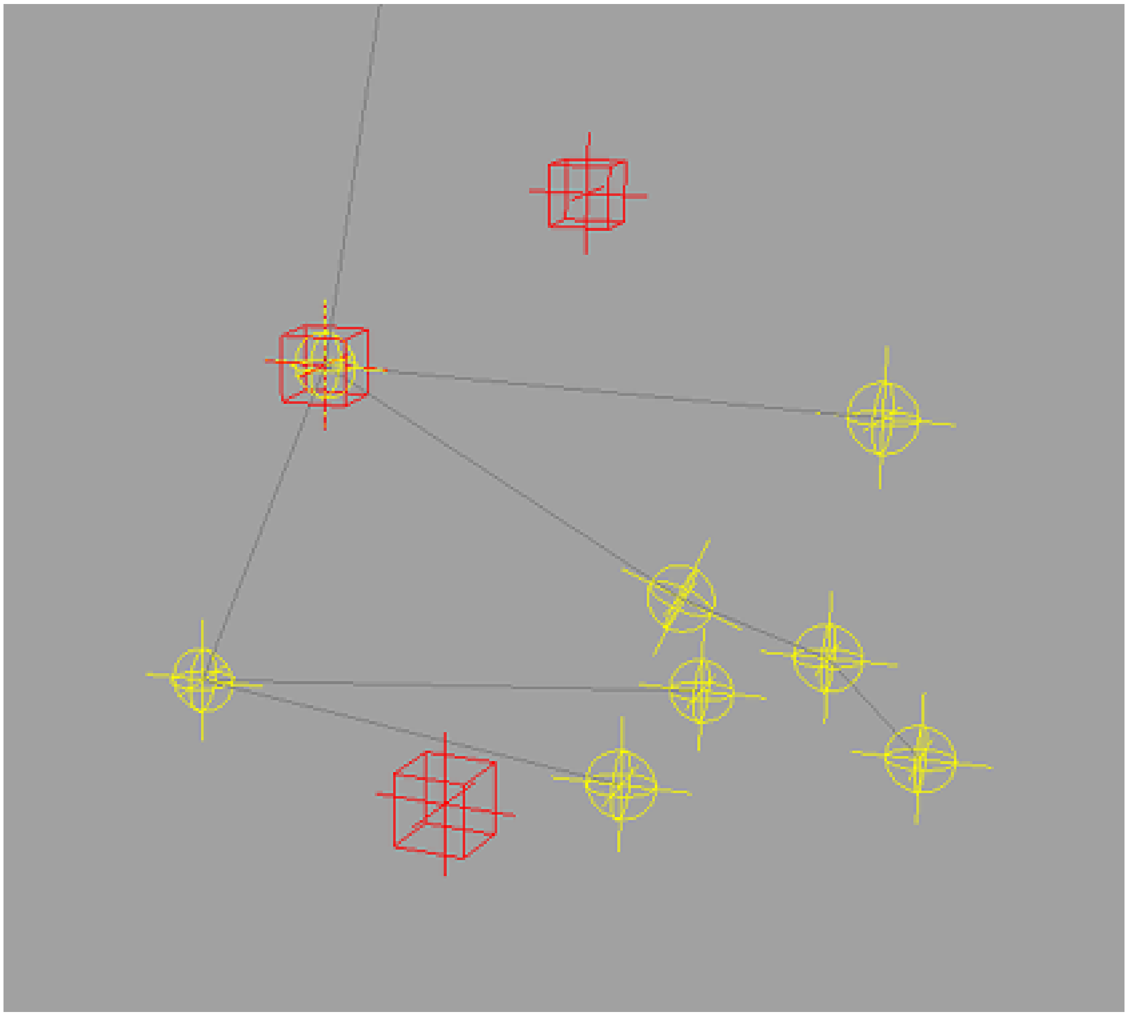
The neck has two yellow **position** tangent guides, that control the neck curve. These are more usefull for rigs with longer necks with more neck joints.

Thumb Rotation



You'll notice that the thumbRoll **root** guide has an RGB axis. This controls the orientation of the thumbRoll controller. In addition the thumbs and each of the fingers have a **blade** guides to control the plane fingers should rotate on.

Feet



The foot has a lot of guides, but most of them are quite straight forward. The bottom three, the **heel** and **in/outpivot** set the pivot points when you rotate the foot. The one in front of the ankle called **eff** controls the direction of the FK foot control. The remaining ones are simply to position the characters joints and toes.

Note: You'll notice there are no guides for the pole vectors for the knees and elbows. These are positioned automatically based on the direction the knee or elbow is pointing in when you build the rig.

2.2.3 Building the Animation Rig

Once you are ready to build the rig, you can simply select the **guide** node from the Outliner and run mGear>Shifter>Build from Selection from the menu. This will bulid the complete animation rig, with all controls ready to use. You can now hide the **guide** node in the Outliner, and test out the rig. If you need to adjust something, simply delete the **rig** group from the outliner, adjust your **guides** and rebuild it.

Adjusting Controls

Some times the shape or size of the default control curves doesn't fit well with the proportions of your character. You can easily fix this by selecting the vertcies of the control curves, and positining and scaling them as needed.

Once you're happy with your new control curves, select the ones you've modified and store them by choosing mGear>Shifter>Extract Controls from the menu. This will store them under **guide**|**controllers_org**, so that if you delete the rig and rebuild it, it will get your modified control curves instead of the default ones.

Skinning

When you built the rig a selection set was added under **rig_sets_grp** called **rig_deformers_grp** that contains all the joints in the rig you can skin to. Simply skin your mesh to these joints, and the rig is ready for animation.

Note: You may be thinking, what do you do if you need to adjust the position of a joint after skinning and adding blend shapes? You can't simply delete the rig, and rebuild, as that will break the skinning. Shifter is built around the idea of Data Centric Rigging. In short this means that rather storing all the skinning data, blend shapes, model and rig in one file, we store each in separate files, and bring it all together when we build the rig. This is a bit beyond the scope of this quick start, so hop on over to the mGear YouTube channel and check out the Data Centric Rigging workshop.

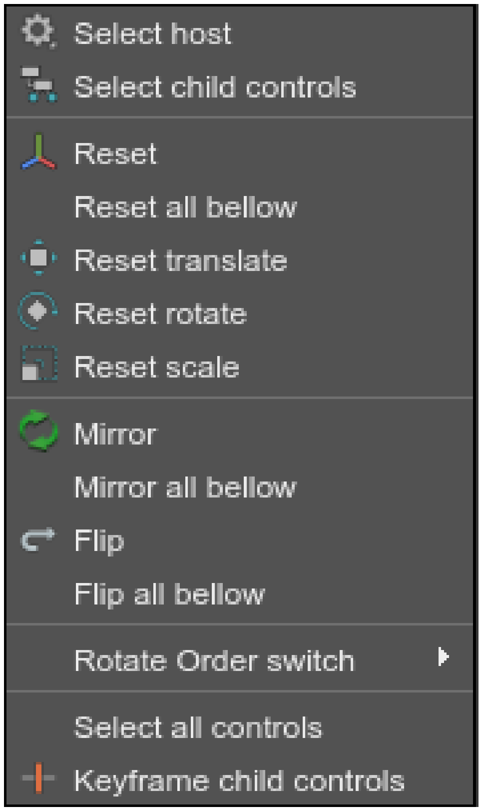
2.3 Animating the Biped

The Shifter biped rig comes with a lot of functionality straight out of the box, including IK/FK blending, stretchy arms, legs and spines, space shifting and even rubber hosing, should you need it. It also performs quite well, so should be able to run in real-time on most modern hardware. Before starting animation, let's cover some of the interfaces to make your work easier.

2.3.1 mGear Viewport Menu

At the top of the mGear menu, there is a checkbox for the mGear Viewport Menu. If you have this enabled it will replace Maya's default right click menu if you have Shifter animation controls selected. Note that the normal right click menu still work as normal, if you have something else selected.

This menu is dynamic, and changes based on what kind of controller you have selected. For most controllers and host controllers it will look like this:



Controller Viewport Menu

- **Select host**: Select the host controller for this bodypart, from which you can do IK/FK blending and more.
- **Select child controls**: Selects the child controls underneath the current one.
- **Reset**: Resets the selected control to it's default position (similar to bind pose)
- **Reset all below**: First does Select Child Controls, and then Reset like above.
- **Reset translate/rotate/scale**: Same as the normal Reset above, but for translate, rotate and scale specifically.
- **Mirror**: Mirrors the selected controllers pose over to the opposite side.
- **Mirror all below**: Same as Mirror, but for all the child controls
- **Flip**: Flip the selected controllers pose to the other side
- **Flip all below**: Same as fliip, but for all the child controls
- **Rotate Order switch**: Change the rotate order, while attempting to keep animation intact

Host Viewport Menu

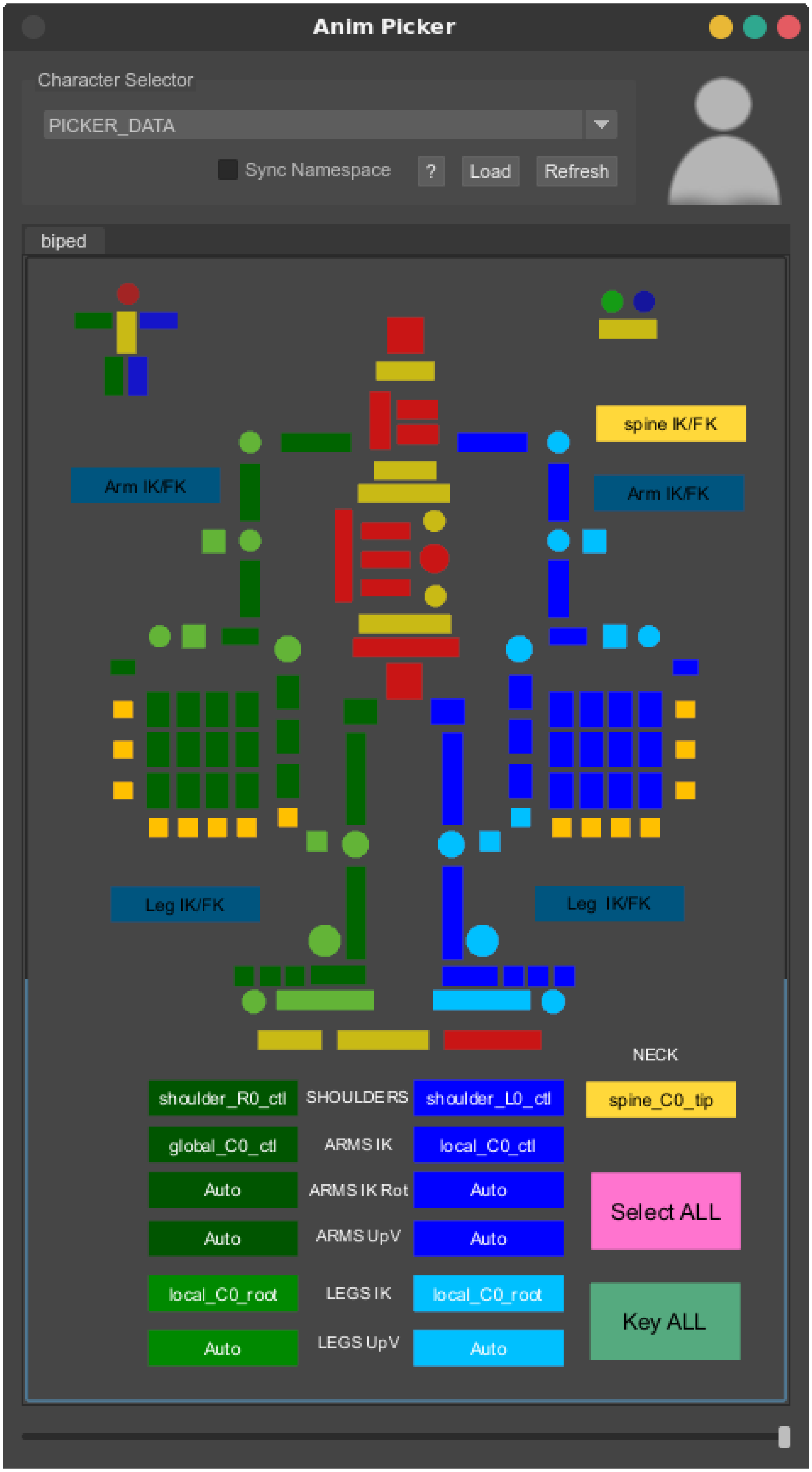
- **Switch arm/leg to Ik/Fk**: Toggle between IK and FK, while keeping the pose intact

- **Switch arm/leg to Ik/Fk + Key**: Toggle between IK and FK, while keeping the pose intact. Adds keyframes before and after the switch.
- **Range switch**: WIP
- **Parent [various]**: Drop down menus for various controllers, such as the IK controller, that set what space that controller follows. You can, for instance switch between the arm's IK controller following the shoulder, body or global controller. If you have keyframes on the component already, this will add keyframes before and after the switch.
- **Select all controls**: Selects all the controllers of the rig
- **Keyframe child controls**: Keyframes all the child controls of the current selection

2.3.2 Anim Picker

mGear comes with a customizable Anim Picker interface to give you easy access to all controls. Choose mGear>Anim Picker>Anim Picker. When you first open it up, it will be empty. Hit the **Load** button and the **Selecte File** and navigate to the biped.pkr file (it comes with mGear and can be found in anim_picker/picker_templates/biped.pkr).

Once you've selected the biped.pkr file, hit the **Load Picker** button. A dialog will pop up askin you to enter a node name for the character. This will create a node in Maya that stores the Anim Picker layout in your scene. You should now see a dialog that looks like this:



Anim Picker
Character Selector
PICKER_DATA
Sync Namespace
?
Load
Refresh
biped
spine IK/FK
Arm IK/FK
Arm IK/FK
Leg IK/FK
Leg IK/FK
NECK
shoulder_R0_ctl
SHOULDERS
shoulder_L0_ctl
spine_C0_tip
global_C0_ctl
ARMS IK
local_C0_ctl
Auto
ARMS IK Rot
Auto
Select ALL
Auto
ARMS UpV
Auto
local_C0_root
LEGS IK
local_C0_root
Key ALL
Auto
LEGS UpV
Auto

From this interface you can select any of the controls by simply clicking the different colored boxes. You can also box select or shift-select multiple controls at a time. The little guy at the top left of the interface gives you access to the **host** controllers, while the two dots and a square on the right are eye controllers. In addition, there are a number of buttons you can use to switch between different spaces or toggle between IK and FK.

Finally if you right-click outside the boxes, you can choose Frame Selection, to zoom in on parts of the interface, or Reset View, to reset the zoom.

Adding Multiple Characters to the Anim Picker

If you have multiple rigs in your scene, you can switch between the characters using the Character Selector menu in the Anim Picker. Note that you can have multiple Anim Pickers windows open at the same time, if you have the screen space for it.

2.3.3 Animate

That's it. You should now be ready to animate with mGear and Shifter. For more in-depth tutorials, please check out the mGear Youtube channel

CHAPTER

THREE

MGEAR RIGGING WORKFLOW

Note: This was the unofficial mGear Workflow document create by Chris Lesage on his website. But now is official ;)

This official-unofficial document is a quick overview of how the mGear Shifter workflow works. It is meant to describe all the parts, so as you go through the tutorials and documentation, you have a full sense of everything.

This document is still a messy work in progress! I started writing it in mGear 2.6.1 just before mGear 3.0.3 was released. I'll update this page, as I continue to learn. I'll improve the illustrations too.

If you have any questions about my workflow, you can find me on the mGear Forums . The entire community there is great for answering questions.

3.1 Quick Reference

mGear official Youtube video tutorials

Python Documentation

How to install mGear: tl;dw: use Maya.env and add this line to point to where you put the mGear folder. Don't copy/paste the scripts into your Maya prefs. Here are my example paths, as I develop my pipeline using Dropbox. The final configuration will be on our studio's network.

```
MAYA_MODULE_PATH = C:\Users\chrislesage\Dropbox\morgaaRigging\tools\mgear_2.6.1
MGEAR_SHIFTER_COMPONENT_PATH = C:\Users\chrislesage\Dropbox\morgaaRigging\tools\
↪mgearCustomModules
MGEAR_SHIFTER_CUSTOMSTEP_PATH = C:\Users\chrislesage\Dropbox\morgaaRigging\data\build
```

HELP! See below for some answers to things that can go wrong.

3.2 FIRST THINGS TO KNOW About mGear

1) **mGear** is an auto-rigging and animation framework. **Shifter** is the name of the rigging framework inside mGear. Shifter has individual rigging components you can build, and full rig templates you can start from.
2) It also has animation tools like IK/FK switching, space-switch baking, synoptic pickers, a basic shot sculpting tool, pose flipping and mirroring, etc. Some of these tools rely on the structure of the rig, so it is good to learn about them before you spend too much effort designing your own custom rig modules or editing the existing ones. If you change the rig, you'll have to also change the synoptic picker, for example.
3) There are a lot of Python modules and functions that you can use to script rigging tasks.

4) For example, `` `mgear.rigbits` `` there are functions for Vectors, creating transforms, building curve constraints, control icons, fcurves, skinning, logging and much more. A full list is in the documentation.
5) However, the docs just list the functions without much context or examples. All the tutorials on how to use mGear are found in the Youtube videos and they are in reverse chronological order.
6) As you learn how to use mGear, **you are going to need to design your own file structure** to organize all the data. **I recommend watching ALL of the Youtube videos first**, as you build your first test rigs, because Miguel offers most of the information you need, but it is contained within all the videos. This unfortunately takes a long time, but it will save you a lot of reorganizing later. For example, you'll want to organize a directory structure where you store all of your skinning, blendshape, and build scripts. And you will probably want to consider using version control like git. In this document, I will show you my example of how I organize my structure on a film which includes 33 characters.
7) Rigging is done by creating guide modules or using the existing template. When you first create a guide, it will give you some options, like the number of joints to use in a chain. When you click `` `Continue` `` it will build the guide. You then place and orient the guide. Additional settings are stored on the guide, like space switching, orientation, naming, etc.
8) When you build an mGear rig, you can define **PRE** and **POST** scripts which run before and after the main rig builds. If you open the settings of your main guide, you can access this interface. You can run your own files here. But if you click "New" in the interface, it will include some helpful boilerplate code, which will give you access to a special dictionary called stepDict from mgear.maya.shifter.customStep. This dictionary has information about the rig, so you don't just have to call the names of objects blindly from your scene.
9) The basic rigging workflow goes a bit like this:
    1) **QUICK OVERVIEW: Build Guide → Place Guide → Build Rig → Test rig → Save Control Icons, skinning, blendshapes, etc. → Delete rig → Tweak the guides → Write POST scripts -> Repeat**
    2) Open `` `mGear → Shifter → Guides UI` `` to access the UI for building modules.
    3) You use the Guide Templates, or build your own by assembling different modules.
    4) If you select a part of the guide and click "Settings" in the mGear UI, a window opens up with settings for that module. For example, you can set Chain_01 to be FK, IK, or FK/IK blend. You can also set the name, symmetry and more.
    5) All of your settings and guide placements are saved in the Guide. The root of the guide also has settings for the entire rig.
    6) There is no auto-symmetry (you could build it, in theory). Instead you delete the right side, and then **Duplicate Symmetry** on the left side in order to get symmetrical parts. (Or right to left.) In order for your guides to understand how to duplicate properly, you have to set your guides to be "Left", "Right" or "Center" in the guide settings. Center guides cannot be duplicated symmetrically.
    7) As you are building the rig, you can select certain components, or the entire guide and click "Build from Selection". This builds the rig you so you can test it. Do not be afraid to test early and test often.
    8) As you test your rig, you delete the rig, tweak the guides and your build scripts, and delete the rig and rebuild.
    9) When you build, you can add **PRE** and **POST** Python scripts, so you can customize the result to your own pipeline. Watch "Data-Centric Rigging" on mGear Youtube for more details. For example PRE: You could create a symmetry constraint, and then delete it in the PRE step.
    10) When creating **PRE** and **POST** steps, if you click "new" in the Custom Steps UI, then mGear will automatically populate your new script file with a bit of boilerplate. There are some functions

for passing dictionaries of objects between steps. However, you can also just run plain vanilla Python in these files too.

11) Try making some PRE and POST scripts early on as you are rigging so you can see how it works. But before you get too far, I recommend designing how you are going to organize the scripts for multiple characters or projects. For example, I have a folder called "common" where I store scripts that will be run on every single character. Then each character will have their own custom scripts stored in a separate folder, if needed. ALL of my scripts will live under one directory where I can track it using git.

12) You can set up hotkeys to do some of the frequent mGear actions. mGear → Utilities → Create mGear hotkeys. This adds a bunch of actions in the Hotkey Editor. Then it is up to you to map them. I map "Build From Selection" to F7. And "Settings" to F6. Hitting F6 opens the module settings of the rig component I have selected.

13) **Control Icons:** When the rig is built, you can tweak or replace your control icons, and click "Extr. Ctl". This stands for "Extract Control" and it saves the icons inside the guide. The next time you build the script, it will use your custom icons. To reset to the default, you can simply delete the icons from inside the guide|controllers_org group. You can also import icons from another character or another file. They are based on name. If names don't match, they won't be used.

14) Saving your Skinning

    When you build your rig, you can skin it, and then save the skinning to external .jSkin or .gSkin file. You can also save all your skinning in a bunch using a .jSkinPack file which basically just collects a list of all your separate .jSkin files. Even after saving a .jSkinPack file, you can still save the weights of an individual piece of geometry to its own .jSkin file. The Pack file will not change, unless you need to add or remove geometry. I recommend using the Ascii .jSkin files, instead of the binary .gSkin files. You can edit the Ascii files in a text editor, which lets you fix problems like mis-named joints. In the past, the binary files were a bit smaller on your hard drive, but since mGear 3.2, all skinning data files are about 90% smaller. **IMPORTANT:** This can be confusing to new users. When you export skinning, it will not automatically add it to your skinPack file. If you export a skinPack file, it will overwrite any existing skinPack file. So if you want to add a single skin to a skinPack file, the best way is to export a single skin file, add then add it manually to the skinPack file using a text editor. If you accidentally lose or overwrite your skinPack file, you might think you lost all your skinning. But the skin files are separate, and the skinPack file is just a text list of all those skin files. So you can often find and fix those kinds of problems. If you change the skinning on your character, and you already have a skinPack file, you just need to export the skin file. NOT the entire skinPack file again. I'll say it again: The skinPack file is just a text list of skinning data files, used for loading multiple skins all at once.

15) Loading your Skinning

During the rig build you can define a POST step to import your .jSkinPack files from a saved location.

```
from mgear.core import skin
skinPath = '/YOUR/PATH/HERE'
skin.importSkinPack(skinpath)
```

16) Saving your Blendshapes

I just keep my blendshapes stored on my character's geometry. When I delete the rig, and rebuild, I run a Post Python script that connects my blendshapes to the controllers I want. This is completely dependent on your own personal rig design. So I don't have specific tips. But in general, this is how I structure my blendshapes using a hook node

10) **Rigging the eyes and face and getting that data back when you rebuild**

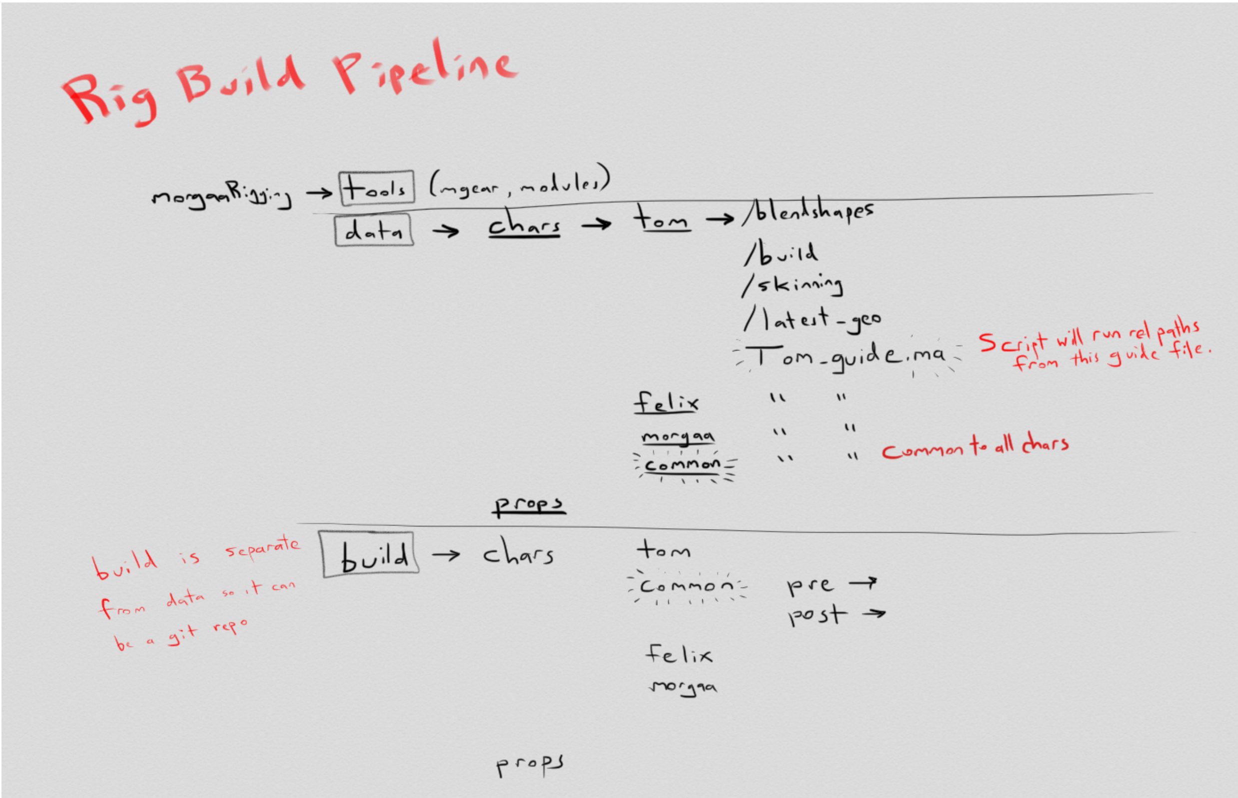
There is an autorigging utility to build the eyes and lips based on selecting edge loops and points. Then the autorigger computes skinning for you. Here is how that works:

1) Build your character from the guide. If you don't already have the biped template's face controls, I recommend making your own simple Control_01 modules for the upper lips, lower lips, and each eye. Configure them to have a joint. These controls will just be there so you can use that joint as a root for the face rig.
2) When the rig is built, open the lips rigging utility from the mGear menu.
3) From the lips rigging UI, you can just experiment with settings, until you get it right. If you get it wrong, delete your rig, rebuild, and try again. You can leave the lips rigging window open and not lose your configuration.
4) When you are happy with the result, you can edit your skinning a bit if needed. **This part is tricky. I personally generate my eyes skinning, and then export it to an ngSkinTools file. Then I generate my lips, and export it as well. Later, after I've skinned the rest of my character, I import these layers on top of all the other skinning.**
5) In the guide, turn OFF "Compute Topological Autoskin". Because if you have that checked, it will add the skinning before the lips exist, and mess up your weights.
6) Save your lips config to a json file.
7) Now that the json file is saved, you can add a POST step to your rig build, and use this code. Please be careful to note which version of the eyes and lips tool you are using. The legacy tools still exist. The new facial_rigger module is the one where the lips and eyes are in tabs of the same window. They use different functions, and save a different data format. I highly recommend using the new facial_rigger version.

```
from mgear import rigbits
import os
lipsConfigPath = os.path.abspath(os.path.join('YOUR PATH HERE'))
eyesConfigPath = os.path.abspath(os.path.join('YOUR PATH HERE'))

# OLD LEGACY eye_rigger and lips_rigger tool
rigbits.eye_rigger.eyesFromfile(eyesConfigPath)
rigbits.lips_rigger.lipsFromfile(lipsConfigPath)

# NEW "facial_rigger" module
rigbits.facial_rigger.eye_rigger.rig_from_file(eyesConfigPath)
rigbits.facial_rigger.lips_rigger.rig_from_file(lipsConfigPath)
```

8) That is going to load your lip rigging. AFTER you load the lip rigging, you want to load your skin weights. Make sure to run the lips rig first, and then import the skinning, otherwise the joints won't exist yet! In this pipeline, you need to think about the order that things happen.

3.2.1 PIPELINE



A sketch for how I'll organize pipeline data. Inside each character will be blendshapes, skinning, the latest_geo, and any other stuff. By putting the data inside a character, it is easier to clone for a new character. There will also be a "common" character folder, where build scripts will live that will run on every character. Shifter build scripts will be kept under a separate "build" directory, and tracked as a separate git repo.

- [ ] FIRST: Design the pipeline. The rigs will be built from guides. Figure out how to save this as a template asset that can easily be reused and iterated on.
- [ ] Maybe write this as a Google Doc or a Wiki doc.

The best way to install mGear is by putting it in a directory that can be accessed by your artists, and then pointed to by using the Maya environment variable MAYA_MODULE_PATH = /your/directory/here/foo

If you are creating custom modules, you should store them in a separate directory of your choosing, and then point to it in your Maya.env by using MGEAR_SHIFTER_COMPONENT_PATH = /your/directory/here/foo

Consider that in your studio, in the long-term, you will like be using mGear on multiple projects, supporting different versions of mGear, or customizing modules that are specific to a project.

Import the face template and place it (script this or improve the hooks?)

Do whatever custom clothes and hair needs rigging

3.3 HELP! How do I fix this???

In this section, I'll try to answer some questions that I got hung up on at first. When you are doing "Data-Centric rigging" like with mGear and Shifter, there is a level of complexity, and there are bound to be things that go wrong.

3.4 Q. I renamed all my controls and my custom icons are gone!

A. No problem. Custom icons are stored by name. "[NAME]_controlBuffer". If you look under your guide, there is a group called controllers_org. Your icons are probably still under that group, but named with the old control names. You can simply rename the objects under the controllers_org group to match your new controls. The next time you build it will be fixed. You can also import or manually add shapes into that group.

3.5 Q. I made some gimmick joints and skinned them, but the next time I build, they are lost!

A. Gimmick joints are not stored in the guide. You have to add them during the POST build stage. You can add them manually when you are first testing and skinning them. But once you are happy with them, add a POST step and use mgear.rigbits.addBlendedJoint(). There are 2 videos where Miguel shows how to create Gimmick Joints, but he doesn't explain how to rebuild them using Python until Data Centric Rigging 014: Gimmick joints custom step and first rebuild

Make sure to build your Gimmick Joints before importing your skinning, otherwise the joints won't exist yet!

3.6 Q. Is it possible to rig… X, Y or Z in mGear?

A. mGear Shifter is an open source rigging framework, and you can run any Python you like during your build process. So YES it is possible, but you will likely have to do some scripting or create your own custom modules. The existing Shifter modules do not cover every possibility you might want. As of January 2019, there are also very limited facial features, so you will have to design and rig a lot of your own face components. mGear is not a magic bullet for every rig you can imagine, but it is open enough that you can easily customize it to suit any needs.

CHAPTER

FOUR

FRAMEWORK

4.1 Animbits

Content:

- *Introduction*
- *Modules*

4.1.1 Introduction

Animbits is a animation commont library with tools and functions for the animators.

4.1.2 Modules

mgear.animbits
mgear.animbits.menu

mgear.animbits

mgear.animbits.menu

Functions

install()	Install Skinning submenu

mgear.animbits package

Subpackages

mgear.animbits.cache_manager package

Submodules

mgear.animbits.cache_manager.collapse_widget module

class mgear.animbits.cache_manager.collapse_widget.**QCollapse**(**args: Any*, ***kwargs: Any*)

Bases: `QWidget`

QCollapse is a collapsible widget with transition

Parameters

- **parent** (*QWidget*) – parent widget for the QCollapseWidget
- **title** (*str*) – Title name for the widget

SPEED = 150

set_layout(*layout*)

Applies the given layout to the scroll area widget

Parameters
layout (*QLayout*) – layout widget to add into the QCollapse

mgear.animbits.cache_manager.dialog module

mgear.animbits.cache_manager.mayautils module

mgear.animbits.cache_manager.model module

class mgear.animbits.cache_manager.model.**CacheManagerStringListModel**(**args: Any*, ***kwargs: Any*)

Bases: `QAbstractListModel`

Custom list model for the cache manager

Parameters

- **items** (*list*) – string list of rigs inside scene
- **parent** (*QtWidget*) – Parent widget

data(*index*, *role*)

Override QAbstractListModel method

data returns the item name and icon at the given index

rowCount(*parent*)

Override QAbstractListModel method

rowCount returns the number of items in the list model

mgear.animbits.cache_manager.query module

`mgear.animbits.cache_manager.query.`**`find_model_group_inside_rig`**(*geo_node*, *rig_node*)

Finds the given group name inside the hierarchy of a rig

Parameters

- **`geo_node`** (*`str`*) – geometry group transform node containing the shapes to cache
- **`rig_node`** (*`str`*) – Rig root node containing the geo_node

Returns
Full path to the geo_node if found else None

Return type
str or None

`mgear.animbits.cache_manager.query.`**`get_cache_destination_path`**()

Returns the cache destination path

This methods returns a path pointing to where the cache manager will store the GPU caches.

If the **MGEAR_CACHE_MANAGER_CACHE_DESTINATION** environment variable has been set it will return whatever path has been set if valid. If none has been set or fails to use that path this method tries then to return whatever settings has been set on the **cache manager preferences file**.

Finally if no environment variable or preference file is been set then we use the **OS TEMP** folder as destination path.

Returns
cache destination path

Return type
str

`mgear.animbits.cache_manager.query.`**`get_model_group`**(*ignore_selection=False*)

Returns the model group name to cache

This methods returns a string with the name of the transform node to use when caching the geometry/model

If the **MGEAR_CACHE_MANAGER_MODEL_GROUP** environment variable has been set it will return the name stored in it. If none has been set or fails to use that name this method tries then to return whatever settings has been set on the **cache manager preferences file**.

Finally if no environment variable or preference file is been set then we fall back to selection.

This doesn't check if the returner values is a valid value like if the transform node exists. This is because we do not know at this stage if the asset it inside a namespace or something scene specific. As this is generic only checks for that generic part are been made.

Parameters
`ignore_selection` (*`bool`*) – whether it falls back to the selected group

Returns
group name if anything or None

Return type
str or None

`mgear.animbits.cache_manager.query.`**`get_preference_file`**()

Returns the preference file path and name

Returns
preference file path and name

Return type
str

`mgear.animbits.cache_manager.query.get_preference_file_cache_destination_path()`

Returns the folder path set on the preference file

Returns
The path stored in the preference file or None if invalid

Return type
str or None

`mgear.animbits.cache_manager.query.get_preference_file_model_group()`

Returns the model group name set on the preference file

Returns

Model group name stored in the preference file
or None if invalid

Return type
str or None

`mgear.animbits.cache_manager.query.get_scene_rigs()`

The rigs from current Maya session

This method search for rigs in your current Maya scene. If the MGEAR_CACHE_MANAGER_RIG_ATTRIBUTE has been set it will try to find rigs based on the attribute set on the environment variable. Otherwise it will use the attribute **gear_version** in order to find rigs in scene.

Returns
mGear rig top node or None

Return type
list or None

`mgear.animbits.cache_manager.query.get_time_stamp()`

Returns the date and time in a file name friendly way

This is used to create the cache file name a unique name in order to avoid clashing files overwriting other cache files been used on other specific file scenes

Returns
time stamp (19-05-12_14-10-55) year-month-day_hour-minutes-seconds

Return type
str

`mgear.animbits.cache_manager.query.get_timeline_values()`

Returns the min and max keyframe values from the current playback range

In order to give more freedom to the artist we always evaluate the playback range and not the animation range so that artists can choose what range to use when creating the GPU cache

Returns
min and max value

Return type
float, float

`mgear.animbits.cache_manager.query.`**`is_rig`**`(`*rig_node*`)`

Returns whether the given rig node is in srig state or caching state

Parameters
`rig_node` (*`str`*) – rig node name

`mgear.animbits.cache_manager.query.`**`read_preference_key`**`(`*search_key*`)`

Returns the preference stored on the pref file for the given key

Returns
The path stored in the preference file or None if invalid

Return type
str or None

Module contents

ANIMATION CACHE MANAGER

mGear's animation cache manager is a tool that allows generating a Alembic GPU cache representation of references rigs inside Autodesk Maya.

module
mgear.animbits.cache_manager.__init__

Submodules

mgear.animbits.channel_master module

mgear.animbits.channel_master_node module

`mgear.animbits.channel_master_node.`**`create_channel_master_node`**`(`*name*`)`

Create a new channel master node

Parameters
`name` (*`str`*) – name of the nodes

Returns
name of the channel master node

Return type
str

`mgear.animbits.channel_master_node.`**`export_data`**`(`*node*, *tab=None*, *filePath=None*`)`

Export the node data

Parameters

- **`node`** (*`str`*) – node to export
- **`tab`** (*`str, optional`*) – if a tab name is set, only that taba will be exported
- **`filePath`** (*`str, optional`*) – the path to save the configuration file

Returns
Description

Return type
TYPE

`mgear.animbits.channel_master_node.get_external_data(node)`

`mgear.animbits.channel_master_node.get_node_data(node)`

Get the configuration data from a node Can get the data from the external data or from local data

Parameters

- **node** (*str*) – nme of the node
- **use_local_data** (*bool, optional*) – If true will force to use the node local data

Returns
configuration data

Return type
dict

`mgear.animbits.channel_master_node.import_data(filePath=None, node=None, add_data=False)`

Import and create channel master configuration nodes

Parameters

- **filePath** (*str, optional*) – Path to the channel master config file
- **node** (*None, str*) – Node to add the data. If None, will create a new node
- **add_data** (*bool, optional*) – If true, will add the data to existing node

Returns
Description

Return type
TYPE

`mgear.animbits.channel_master_node.list_channel_master_nodes()`

return a list of channel master nodes in the scene

Returns
List of channel master nodes

Return type
list

`mgear.animbits.channel_master_node.remove_external_config_path(node)`

`mgear.animbits.channel_master_node.set_external_config_path(node, filePath=None)`

Set the path to the external file configuration

Parameters

- **node** (*PyNode*) – Channel Master node
- **filePath** (*str, optional*) – Path to the file confituration

Returns
Description

Return type
TYPE

`mgear.animbits.channel_master_node.`**`set_node_data`**`(`*node*, *data*`)`

Set the node data attribute

Parameters

- **`node`** (*`str`*) – node name
- **`data`** (*`dict`*) – configuration dict

mgear.animbits.channel_master_utils module

`mgear.animbits.channel_master_utils.`**`channel_has_animation`**`(`*attr*`)`

Check if the current channel has animaton

Parameters
`attr` (*`str`*) – Attribute fullName

Returns
Return true if the attribute has animation

Return type
bool

`mgear.animbits.channel_master_utils.`**`current_frame_has_key`**`(`*attr*`)`

Check if the attribute has keyframe in the current frame

Parameters
`attr` (*`str`*) – Attribute fullName

Returns
Return true if the attribute has keyframe in the current frame

Return type
bool

`mgear.animbits.channel_master_utils.`**`get_anim_value_at_current_frame`**`(`*attr*`)`

Get the animation value in the current framwe from a given attribute

Parameters
`attr` (*`str`*) – Attribute fullName

Returns
animation current value

Return type
bol, int or float

`mgear.animbits.channel_master_utils.`**`get_attributes_config`**`(`*node*`)`

Get the configuration to all the keyable attributes

Parameters
`node` (*`str`*) – name of the node that have the attribute

Returns
All keyable attributes configuration

Return type
dict

`mgear.animbits.channel_master_utils.`**`get_ctl_with_namespace`**`(`*attr_config*, *namespace=None*`)` mgear.animbits.channel_master_utils.**get_keyable_attribute**(*node*)

Get keyable attributes from node

Parameters
node (*str*) – name of the node that have the attribute

Returns
list of keyable attributes

Return type
list

mgear.animbits.channel_master_utils.**get_single_attribute_config**(*node*, *attr*)

Summary

Parameters

- **node** (*str*) – name of the node that have the attribute
- **attr** (*str*) – attribute name

Returns
attribute configuration

Return type
dict

mgear.animbits.channel_master_utils.**get_table_config_from_selection**()

mgear.animbits.channel_master_utils.**init_channel_master_config_data**()

Initialize the dictionary to store channel master tabs configuration

mgear.animbits.channel_master_utils.**init_table_config_data**()

Initialize the dictionary to store the channel master table data

Items are the channels or attributes fullname in a list items_data is a dictionary with each channel configuration, the keys is the fullName

Returns
configuration dictionary

Return type
dict

mgear.animbits.channel_master_utils.**next_keyframe**(*attr*)

mgear.animbits.channel_master_utils.**previous_keyframe**(*attr*)

mgear.animbits.channel_master_utils.**remove_animation**(*attr*)

Remove the animation of an attribute

Parameters
attr (*str*) – Attribute fullName

mgear.animbits.channel_master_utils.**remove_key**(*attr*)

Remove the keyframe of an attribute at current frame

Parameters
attr (*str*) – Attribute fullName

`mgear.animbits.channel_master_utils.`**`reset_attribute`**`(`*attr_config*, *namespace=None*`)`

Reset the value of a given attribute for the attribute configuration

Parameters
`attr_config` (*`dict`*) – Attribute configuration

`mgear.animbits.channel_master_utils.`**`reset_creation_value_attribute`**`(`*attr_config*, *namespace=None*`)`

Reset the value of a given attribute for the attribute configuration

Parameters
`attr_config` (*`dict`*) – Attribute configuration

`mgear.animbits.channel_master_utils.`**`set_key`**`(`*attr*`)`

Keyframes the attribute at current frame

Parameters
`attr` (*`str`*) – Attribute fullName

`mgear.animbits.channel_master_utils.`**`sync_graph_editor`**`(`*attr_configs*, *namespace=None*`)`

sync the channels in the graph editor

Parameters
`attr_configs` (*`list`*) – list of attribute configuration

`mgear.animbits.channel_master_utils.`**`value_equal_keyvalue`**`(`*attr*, *current_time=False*`)`

Compare the animation value and the current value of a given attribute

Parameters
`attr` (*`str`*) – the attribute fullName

Returns
Return true is current value and animation value are the same

Return type
bool

mgear.animbits.channel_master_widgets module

mgear.animbits.menu module

`mgear.animbits.menu.`**`install`**`()`

Install Skinning submenu

mgear.animbits.softTweakWindowUI module

mgear.animbits.softTweaks module

mgear.animbits.space_recorder module

mgear.animbits.version module

Module contents

4.2 Base Modules

mgear	mGear init module
mgear.core	
mgear.core.applyop	**Apply operator module**
mgear.core.attribute	Attribute creation functions
mgear.core.curve	NurbsCurve creation functions
mgear.core.dag	Nvigate the DAG hierarchy
mgear.core.fcurve	
mgear.core.icon	Predefined nurbsCurve shapes to be use as a rigging control Icons
mgear.core.log	Logging Maya data
mgear.core.meshNavigation	Functions to help navigate the mesh topology
mgear.core.node	Functions to create and connect nodes.
mgear.core.primitive	Functions to create primitives (Non geometry)
mgear.core.skin	Functions to work with skinCluster data.
mgear.core.string	string management methods
mgear.core.transform	Functions to work with matrix and transformations
mgear.core.utils	Utilitie functions
mgear.core.vector	Functions to work with vectors

4.2.1 mgear

mGear init module

Functions

getInfos(level)	Get information from where the method has been fired.
getVersion()	Get mGear version
install()	
log(message[, severity, infos])	Log a message using severity and additional info from the file itself.
logInfos()	Log version of Gear
reloadModule([name])	Reload a module and its sub-modules from a given module name.
setDebug(b)	Set the debug mode to given value.
toggleDebug()	Toggle the debug mode value.
toggleLog()	Toggle the log value.

Exceptions

FakeException

4.2.2 mgear.core

Functions

aboutMgear(*args)	About mgear
getMayaVer()	Get Maya version

4.2.3 mgear.core.applyop

Apply operator module

Operators are any node that connected to other nodes creates a rig behaviour:

```
I.E: IK solvers and  constraints are operators
```

Functions

aimCns(obj, master[, axis, wupType, ...])	Apply a direction constraint
create_proximity_constraint(shape, in_trans)	Create a proximity constraint between a shape and a transform.
create_proximity_constraints(shape, ...)	Create a proximity constraint between a shape and a list of transforms.
curvecns_op(crv[, inputs])	
gear_curvecns_op(crv[, inputs])	create mGear curvecns node.
gear_curveslide2_op(outcrv, incrv[, ...])	Apply a sn_curveslide2_op operator
gear_ikfk2bone_op([out, root, eff, upv, ...])	Apply a sn_ikfk2bone_op operator
gear_intmatrix_op(mA, mB[, blend])	create mGear interpolate Matrix node.
gear_inverseRotorder_op(out_obj, in_obj)	Apply a sn_inverseRotorder_op operator
gear_matrix_cns(in_obj[, out_obj, ...])	Create and connect matrix constraint node
gear_mulmatrix_op(mA, mB[, target, transform])	Create mGear multiply Matrix node.
gear_raycast(in_mesh, ray_source, ray_direction)	Create and connect mraycast node
gear_rollsplinekine_op(out[, controlers, u, ...])	Apply a sn_rollsplinekine_op operator
gear_spinePointAtOp(cns, startobj, endobj[, ...])	Apply a SpinePointAt operator
gear_spinePointAtOpWM(cns, startobj, endobj)	Apply a SpinePointAt operator using world matrix
gear_spring_op(in_obj[, goal])	Apply mGear spring node.
gear_squashstretch2_op(out[, sclref, ...])	Apply a sn_squashstretch2_op operator
oriCns(driver, driven[, maintainOffset])	Apply orientation constraint
parentCns(driver, driven[, maintain_offset])	Apply a parent constraint from driver to driven, skipping locked attributes, and allowing additional keyword arguments for the parentConstraint command.
pathCns(obj, curve[, cnsType, u, tangent])	Apply a path constraint or curve constraint.
splineIK(name, chn[, parent, cParent, curve])	Apply a splineIK solver to a chain.

4.2.4 mgear.core.attribute

Attribute creation functions

Functions

addAttribute(node, longName, attributeType)	Add attribute to a node
addColorAttribute(node, longName[, value, ...])	Add a color attribute to a node
addEnumAttribute(node, longName, value, enum)	Add an enumerate attribute to a node
addFCurve(node[, name, keys])	FCurve attribute
addProxyAttribute(sourceAttrs, targets[, ...])	Add proxy paramenter to a list of target dagNode Duplicated channel policy, stablish the rule in case the channel already exist on the target.
addVector3Attribute(node, longName[, value, ...])	Add a vector3 attribute to a node
add_mirror_config_channels(ctl[, conf])	Add channels to configure the mirror posing
change_default_value(attributes, defaultValue)	Change the default value of the attr
collect_attrs(node, attrs, attrs_list[, shapes])	Collect the channel full path in a list.
connectSet(source, target, testInstance)	Connect or set attributes
connect_add_dynamic_pivot(pivots, driven)	connect dynamic pivot with option to add offset channels on XYZ
connect_dynamic_pivot(pivot, driven)	connects translation of pivot dagNode to rotatePivot and scalePivot of the driven transform
connect_message(source, attr)	Connects the 'message' attribute of one or more source nodes to a destination attribute.
disconnect_inputs(node[, attributes])	Disconnects only the input connections of the specified attributes of the provided node.
disconnect_outputs(node[, attributes])	Disconnects only the output connections of the specified attributes of the provided node.
getSelectedChannels([userDefine])	Get the selected channels on the channel box
getSelectedObjectChannels([oSel, ...])	Get the selected object channels.
get_channelBox()	Get the channel box
get_default_value(node, attribute)	Get the default attribute value
get_next_available_index(attr)	get the next available index from a multi attr This function is a workaround because the connect attr flag next available is not working.
get_selected_channels_full_path()	Get the selected channels full path from channel box This function will collect channels from any area of the channel box.
has_in_connections(node[, attributes])	Checks if the provided node has any input connections on the specified attributes.
lockAttribute(node[, attributes])	Lock attributes of a node.
moveChannel(attr, sourceNode, targetNode[, ...])	Move channels keeping the output connections.
move_input_connections(source, target[, ...])	Move the input connections from source node to target node.
move_output_connections(source, target[, ...])	Move the output connections from source node to target node.
reset_SRT([objects, attributes])	Reset Scale Rotation and translation attributes to default value
reset_selected_channels_value([objects, ...])	Reset the the selected channels if not attribute is provided
setInvertMirror(node[, invList])	Set invert mirror pose values
setKeyableAttributes(nodes[, params])	Set keyable attributes of a node.

continues on next page

Table 1 – continued from previous page

setNotKeyableAttributes(nodes[, attributes])	Set not keyable attributes of a node.
setRotOrder(node[, s])	Set the rotorder of the object.
set_default_value(node, attribute)	Set the default value to the attribute
smart_reset(*args)	Reset the SRT or the selected channels
toggle_bool_attr(attr)	
unlockAttribute(node[, attributes])	Unlock attributes of a node.

Classes

FCurveParamDef(scriptName[, keys, ...])	Create an Fcurve parameter definition.
ParamDef(read as Parameter Definition)	Encapsulate the attribute creation arguments in a handy object.
ParamDef2(scriptName, valueType, value[, ...])	ParamDef2 inherit from ParamDef
colorParamDef(scriptName[, value])	Create a Color parameter definition.
enumParamDef(scriptName, enum[, value])	Create an enumarator parameter definition.

4.2.5 mgear.core.curve

NurbsCurve creation functions

Functions

addCnsCurve(parent, name, centers[, degree])	Create a curve attached to given centers.
addCurve(parent, name, points[, close, ...])	Create a NurbsCurve with a single subcurve.
add_linear_skinning_to_curve(curve_name, ...)	Adds a skinCluster to a curve and sets the skinning weights linearly among the list of joints based on the number of control points.
average_curve(crv, shapes[, average, ...])	Average the shape, rotation and scale of the curve bettwen n number of curves
collect_curve_data(objs[, rplStr])	Generate a dictionary descriving the curve data
collect_curve_shapes(crv[, rplStr])	Collect curve shapes data
collect_selected_curve_data([objs, rplStr])	Generate a dictionary descriving the curve data from selected objs
cox_de_boor(u, i, p, knots)	Cox-De Boor algorithm to evaluate B-Spline basis function.
createCurveFromCurve(srcCrv, name, nbPoints)	Create a curve from a curve
createCurveFromOrderedEdges(edgeLoop, ...[, ...])	Create a curve for a edgeloop ordering the list from starting vertex
createCuveFromEdges(edgeList, name[, ...])	Create curve from a edge list.
create_curve_from_data(data[, replaceShape, ...])	Build the curves from a given curve data dict
create_curve_from_data_by_name(crv, data[, ...])	Build one curve from a given curve data dict
create_locator_at_curve_point(object_names, ...)	Create a locator at a point on a cubic NURBS curve in Maya.
crv_parenting(data, crv[, rplStr, model])	Parent the new created curves
curl_curve(crvs[, amount, frequency])	

continues on next page

Table 2 – continued from previous page

evaluate_cubic_nurbs(control_points, percentage)	Evaluate a cubic NURBS curve at a given percentage.
export_curve([filePath, objs, rplStr])	Export the curve data to a json file
findLenghtFromParam(crv, param)	Find lengtht from a curve parameter
getCurveParamAtPosition(crv, position)	Get curve parameter from a position
getParamPositionsOnCurve(srcCrv, nbPoints)	get param position on curve
get_color(node)	Get the color from shape node
get_uniform_world_positions_on_curve(curve, ...)	Get a specified number of uniformly distributed world positions along a NURBS curve.
import_curve([filePath, replaceShape, ...])	
keep_lock_length_state(func)	
keep_point_0_cnx_state(func)	
lock_first_point(crv)	
lock_length(crv[, lock])	
rebuild_curve(crvs, spans)	
set_color(node, color)	Set the color in the Icons.
set_thickness(crv[, thickness])	
smooth_curve(crvs[, smooth_factor])	
straighten_curve(crvs[, straighteness, ...])	
update_curve_from_data(data[, rplStr])	update the curves from a given curve data dict
update_curve_from_file([filePath, rplStr])	

4.2.6 mgear.core.dag

Nvigate the DAG hierarchy

Functions

findChild(node, name)	Returns the first child of input node, with a matching name.
findChildren(node, name)	Returns all the children of input node, with a matching name.
findChildrenPartial(node, name)	Returns the children of input node, with a partial matching name.
findComponentChildren(node, name, sideIndex)	Returns the component children of input component root.
findComponentChildren2(node, name, sideIndex)	Returns the component children of input component root.
findComponentChildren3(node, name, sideIndex)	Returns the component children of input component root.
getShapes(node)	Returns the shape of the dagNode
getTopParent(node)	Returns the first parent of the hierarchy.

4.2.7 mgear.core.fcurve

Functions

getFCurveValues(fcv_node, division[, factor])	Get X values evenly spaced on the FCurve.

4.2.8 mgear.core.icon

Predefined nurbsCurve shapes to be use as a rigging control Icons

Functions

`arrow`([parent, name, width, color, m, ...])	Create a curve with a ARROW shape.
`axis`([parent, name, width, color, m, ...])	Create a curve with a AXIS shape.
`circle`([parent, name, width, color, m, ...])	Create a curve with a CIRCLE shape.
`compas`([parent, name, width, color, m, ...])	Create a curve with a COMPAS shape.
`connection_display_curve`(name[, centers, degree])	Visual reference curves connectiong points.
`create`([parent, name, m, color, icon])	Icon master function
`cross`([parent, name, width, color, m, ...])	Create a curve with a CROSS shape.
`crossarrow`([parent, name, width, color, m, ...])	Create a curve with a CROSS ARROW shape.
`cube`([parent, name, width, height, depth, ...])	Create a curve with a CUBE shape.
`cubewithpeak`([parent, name, width, color, ...])	Create a curve with a CUBE WITH PEAK shape.
`cylinder`([parent, name, width, heigth, ...])	Create a curve with a CYLINDER shape.
`diamond`([parent, name, width, color, m, ...])	Create a curve with a DIAMOND shape.
`flower`([parent, name, width, color, m, ...])	Create a curve with a FLOWER shape.
`getPointArrayWithOffset`(point_pos[, ...])	Get Point array with offset
`guideBladeIcon`([parent, name, lenX, color, ...])	Create a curve with a BLADE GUIDE shape.
`guideLocatorIcon`([parent, name, width, ...])	Create a curve with a LOCATOR GUIDE shape.
`guideRootIcon`([parent, name, width, color, ...])	Create a curve with a ROOT GUIDE shape.
`guideRootIcon2D`([parent, name, width, ...])	Create a curve with a 2D ROOT GUIDE shape.
`null`([parent, name, width, color, m, ...])	Create a curve with a NULL shape.
`pyramid`([parent, name, width, height, ...])	Create a curve with a PYRAMIDE shape.
`setcolor`(node, color)	Set the color in the Icons.
`sphere`([parent, name, width, color, m, ...])	Create a curve with a SPHERE shape.
`square`([parent, name, width, depth, color, ...])	Create a curve with a SQUARE shape.

4.2.9 mgear.core.log

Logging Maya data

Functions

`matrix4`(m[, msg])	Print matrix 4x4 data.

4.2.10 mgear.core.meshNavigation

Functions to help navigate the mesh topology

Functions

bBoxData([obj, yZero])	Get bounding box data of a mesh object
bboxCenter(obj[, radius])	Get bounding box center of mesh object
edgeLoopBetweenVertices(startPos, endPos)	Computes edge loop between two vertices.
edgeRangeInLoopFromMid(edgeList, midPos, ...)	Return a range of edges in the same loop from a mid position
find_mirror_edge(obj, edgeIndx)	Return the mirror edge of an edge
getClosestPolygonFromTransform(geo, loc)	Get closest polygon from transform
getClosestVertexFromTransform(geo, loc)	Get closest vertex from transform
getConcentricVertexLoop(loop, nbLoops)	Get concentric vertex loops
getExtremeVertexFromLoop([edgeList, ...])	Get extreme vertex X and Y
getVertexRowsFromLoops(loopList)	Get vertex rows from edge loops
get_closes_edge_index(sourceGeo, targetGeo, ...)	Get the closes edge index from 2 diferent object.
get_edge_center(mesh_dag_path, edge_indices)	Summary
get_mesh_dag_path(node)	Get the mesh dag path from a given node.
get_selected_mesh()	Get the selected mesh dag path.

4.2.11 mgear.core.node

Functions to create and connect nodes.

Functions

add_controller_tag(ctl[, tagParent])	Add a controller tag
controller_tag_connect(ctt, tagParent)	Summary
createAddNode(inputA, inputB)	Create and connect a addition node.
createAddNodeMulti([inputs])	Create and connect multiple add nodes
createBlendNode(inputA, inputB[, blender])	Create and connect a createBlendNode node.
createClampNode(input, in_min, in_max)	Create and connect a clamp node
createClampNodeMulti(name[, inputs, in_min, ...])	Create and connect multiple clamp nodes
createConditionNode([firstTerm, secondTerm, ...])	Create and connect a condition node.
createCurveInfoNode(crv)	Create and connect a curveInfo node.
createDecomposeMatrixNode(m)	Create and connect a decomposeMatrix node.
createDistNode(objA, objB[, output])	Create and connect a distance node.
createDivNode(inputA, inputB[, output])	Create and connect a Divide node.
createDivNodeMulti(name[, inputs1, inputs2])	Create and connect multiple divide nodes
createMulDivNode(inputA, inputB[, ...])	Create and connect a Multiply or Divide node.
createMulNode(inputA, inputB[, output])	Create and connect a Multiply node.
createMulNodeMulti(name[, inputs])	Create and connect multiple multiply nodes
createMultMatrixNode(mA, mB[, target, transform])	Create Maya multiply Matrix node.
createNegateNodeMulti(name[, inputs])	Create and connect multiple negate nodes
createPairBlend([inputA, inputB, blender, ...])	Create and connect a PairBlend node.
createPickMatrix([m, out_m, scale, rotate, ...])	Summary
createPlusMinusAverage1D(input[, operation, ...])	Create a multiple average node 1D. :param input: The input values. :type input: attr, float or list :param operation: Node operation. 0=None, 1=sum, 2=subtract, 3=average :type operation: int :param output: The attribute to connect the result. :type output: attr.
createPowNode(inputA, inputB[, output])	Create and connect a power node.
createReverseNode(input[, output])	Create and connect a reverse node.
createSetRangeNode(input, oldMin, oldMax[, ...])	Create Set Range Node
createSubNode(inputA, inputB)	Create and connect a subtraction node.
createVertexPositionNode(inShape[, vId, ...])	Creates a mgear_vertexPosition node

4.2.12 mgear.core.primitive

Functions to create primitives (Non geometry)

Functions

add2DChain(parent, name, positions, normal)	Create a 2D joint chain.
add2DChain2(parent, name, positions, normal)	Experimental 2D Chain creation function.
addIkHandle(parent, name, chn[, solver, poleV])	Creates and connect an IKhandle to a joints chain.
addJoint(parent, name[, m, vis])	Create a joint dagNode.
addJointFromPos(parent, name[, pos])	Create a joint dagNode.
addLocator(parent, name[, m, size])	Create a space locator dagNode.
addLocatorFromPos(parent, name[, pos, size])	Create a space locator dagNode.
addTransform(parent, name[, m])	Create a transform dagNode.
addTransformFromPos(parent, name[, pos])	Create a transform dagNode.

4.2.13 mgear.core.skin

Functions to work with skinCluster data.

This module is derivated from Chad Vernon's Skin IO.

`Chad Vernon's github

<https://github.com/chadmv/cmt/tree/master/scripts/cmt/deform>`_

Functions

collectBlendWeights(skinCls, dagPath, ...)	
collectData(skinCls, dataDic)	
collectInfluenceWeights(skinCls, dagPath, ...)	
exportJsonSkinPack([packPath, objs])	
exportSkin([filePath, objs])	
exportSkinPack([packPath, objs, use_json])	
getCurrentWeights(skinCls, dagPath, components)	Get the skincluster weights
getGeometryComponents(skinCls)	Get the geometry components from skincluster
getObjsFromSkinFile([filePath])	
getSkinCluster(obj)	Get the skincluster of a given object
get_mesh_components_from_tag_expression(skinC	Get the mesh components from the component tag expression
importSkin([filePath])	
importSkinPack([filePath])	
selectDeformers(*args)	
setBlendWeights(skinCls, dagPath, ...)	
setData(skinCls, dataDic, compressed)	
setInfluenceWeights(skinCls, dagPath, ...)	
skinCopy([sourceMesh, targetMesh])	

4.2.14 mgear.core.string

string management methods

Functions

convertRLName(name)	Convert a string with underscore
convertRLName_old(name)	Convert a string with underscore
normalize(string)	Replace all invalid characters with "_"
normalize2(string)	Replace all invalid characters with "_".
normalize_path(string)	Ensure that string path use always forward slash
normalize_with_padding(string)	Replace all invalid characters with "_".
removeInvalidCharacter(string)	Remove all invalid character.
removeInvalidCharacter2(string)	Remove all invalid character.
replaceSharpWithPadding(string, index)	Replace a list of # symbol with properly padded index.

4.2.15 mgear.core.transform

Functions to work with matrix and transformations

Functions

convert2TransformMatrix(tm)	Convert a transformation Matrix
getChainTransform(positions, normal[, ...])	Get a tranformation list from a positions list and normal.
getChainTransform2(positions, normal[, ...])	Get a tranformation list from a positions list and normal.
getClosestPolygonFromTransform(geo, loc)	Get closest polygon from transform
getDistance2(obj0, obj1)	Get the distance between 2 objects.
getFilteredTransform(m[, translation, ...])	Retrieve a transformation filtered.
getInterpolateTransformMatrix(t1, t2[, blend])	Interpolate 2 matrix.
getOffsetPosition(node[, offset])	Get an offset position from dagNode
getPositionFromMatrix(in_m)	Get the position values from matrix
getRotationFromAxis(in_a, in_b[, axis, negate])	Get the matrix rotation from a given axis.
getSymmetricalTransform(t[, axis, fNegScale])	Get the symmetrical tranformation
getTransform(node)	Return the transformation matrix of the dagNode in worldSpace.
getTransformFromPos(pos)	Create a transformation Matrix from a given position.
getTransformLookingAt(pos, lookat, normal[, ...])	Return a transformation mstrix using vector positions.
getTranslation(node)	Return the position of the dagNode in worldSpace.
get_closes_transform(target_transform, ...)	Summary
get_orientation_from_polygon(face)	Summary
get_raycast_translation_from_mouse_click(...)	get the raycasted translation of the mouse position
interpolate_rotation(obj, targets, blends)	
interpolate_scale(obj, targets, blends)	
matchWorldTransform(source, target)	Match 2 dagNode transformations in world space.
quaternionDotProd(q1, q2)	Get the dot product of 2 quaternion.
quaternionSlerp(q1, q2, blend)	Get an interpolate quaternion based in slerp function.
resetTransform(node[, t, r, s])	Reset the scale, rotation and translation for a given dagNode.
setMatrixPosition(in_m, pos)	Set the position for a given matrix
setMatrixRotation(m, rot)	Set the rotation for a given matrix
setMatrixScale(m[, scl])	Set the scale for a given matrix

4.2.16 mgear.core.utils

Utilitie functions

Functions

`as_pynode`(obj)	Check and convert a given string to Pynode
`filter_nurbs_curve_selection`(func)	
`gatherCustomModuleDirectories`(envvarkey, ...)	returns component directory
`getModuleBasePath`(directories, moduleName)	search component path
`get_dag_path`(name)	Gets the dag path for the specified object name.
`get_frame_rate`()	Returns the current scene's fps.
`get_maya_path`()	Gets the path to the folder where Maya binary lives
`get_os`()	Gets the OS that Maya is running in.
`importFromStandardOrCustomDirectories`(...)	Return imported module
`is_odd`(num)	Check if the number is odd.
`one_undo`(func)	Decorator - guarantee close chunk.
`set_frame_rate`(fps)	Set Maya Scene's frame rate(fps).
`timeFunc`(func)	Use as a property to time any desired function
`viewport_off`(func)	Decorator - Turn off Maya display while func is running.

4.2.17 mgear.core.vector

Functions to work with vectors

Functions

`calculatePoleVector`(p1, p2, p3[, ...])	This function takes 3 PyMEL PyNodes as inputs.
`getDistance`(v0, v1)	Get the distance between 2 vectors
`getDistance2`(obj0, obj1)	Get the distance between 2 objects.
`getPlaneBiNormal`(v0, v1, v2)	Get the binormal vector of a plane (Defined by 3 positions).
`getPlaneNormal`(v0, v1, v2)	Get the normal vector of a plane (Defined by 3 positions).
`getTransposedVector`(v, position0, position1)	Get the transposed vector.
`linearlyInterpolate`(v0, v1[, blend])	Get the vector interpolated between 2 vectors.
`rotateAlongAxis`(v, axis, a)	Rotate a vector around a given axis defined by other vector.

Classes

`Blade`([t])	The Blade object for shifter guides

4.2.18 mgear package

Subpackages

mgear.anim_picker package

Subpackages

mgear.anim_picker.handlers package

Submodules

mgear.anim_picker.handlers.action_handlers module

mgear.anim_picker.handlers.file_handlers module

mgear.anim_picker.handlers.maya_handlers module

mgear.anim_picker.handlers.mode_handlers module

mgear.anim_picker.handlers.python_handlers module

Module contents

mgear.anim_picker.widgets package

Submodules

mgear.anim_picker.widgets.basic module

mgear.anim_picker.widgets.overlay_widgets module

mgear.anim_picker.widgets.picker_widgets module

Module contents

Submodules

mgear.anim_picker.gui module

mgear.anim_picker.menu module

mgear.anim_picker.picker_node module

mgear.anim_picker.version module

Module contents

mgear.cfxbits package

Subpackages

mgear.cfxbits.xgenboost package

Submodules

mgear.cfxbits.xgenboost.guide module

mgear.cfxbits.xgenboost.ui module

mgear.cfxbits.xgenboost.ui_form module

mgear.cfxbits.xgenboost.xgen_handler module

mgear.cfxbits.xgenboost.xgen_handler.**connect_curve_to_xgen_guide**(*crv*, *xgen_description*)

mgear.cfxbits.xgenboost.xgen_handler.**create_curve_guide_setup**(*xgen_description*)

mgear.cfxbits.xgenboost.xgen_handler.**disconnect_curve_from_xgen_guide**(*crv*)

mgear.cfxbits.xgenboost.xgen_handler.**filter_curve_guides**(*crvs*, *xgen_description*)

mgear.cfxbits.xgenboost.xgen_handler.**get_connected_curve_guides**(*xgen_description*)

return the connectected curve guides from descriptions

Parameters
xgen_description (*TYPE*) – Description

Returns
curve guides list or empty list if none

Return type
list

mgear.cfxbits.xgenboost.xgen_handler.**get_curve_to_spline_node**(*xgen_description*)

get the curve to spline node of the guide modifier from xgen description

Parameters
xgen_description (*PyNode*) – xgen Description

Returns
xgen guide curve to spline node

Return type
PyNode

mgear.cfxbits.xgenboost.xgen_handler.**get_description**(*name*)

Get description from string name

Parameters
name (*str*) – name of the desctription

Returns
xgen Description

Return type
PyNode

`mgear.cfxbits.xgenboost.xgen_handler.`**`get_description_from_selection`**`()`

`mgear.cfxbits.xgenboost.xgen_handler.`**`get_scalp`**`(`*xgen_description*`)`

get the scalp object of the xgen description

Parameters
`xgen_description` (*PyNode*) – xgen Description

Returns
scalp object

Return type
PyNode

`mgear.cfxbits.xgenboost.xgen_handler.`**`refresh_curve_connections`**`(`*guide_modifier*`)`

Module contents

Submodules

mgear.cfxbits.menu module

`mgear.cfxbits.menu.`**`install`**`()`

Install CFXbits submenu

mgear.cfxbits.version module

Module contents

mgear.core package

Submodules

mgear.core.anim_utils module

mgear.core.applyop module

Apply operator module

Operators are any node that connected to other nodes creates a rig behaviour:

```
I.E: IK solvers and  constraints are operators
```

`mgear.core.applyop.`**`aimCns`**`(`*obj*, *master*, *axis='xy'*, *wupType='objectrotation'*, *wupVector=[0, 1, 0]*, *wupObject=None*, *maintainOffset=False*`)`

Apply a direction constraint

Parameters

- **obj** (*dagNode*) – Constrained object.
- **master** (*dagNode*) – Constraining Object.
- **axis** (*str*) – Define pointing axis and upvector axis (combination of xyz and -x-y-z).
- **wupType** (*str*) – scene, object, objectrotation, vector, or none.
- **wupVector** (*list of 3 float*) – world up vector. Exp: [0.0,1.0,0.0].
- **wupObject** (*pyNode*) – world up object.
- **maintainOffset** (*bool*) – Maintain offset.

Returns
Newly created constraint.

Return type
pyNode

mgear.core.applyop.**create_proximity_constraint**(*shape*, *in_trans*, *existing_pin=None*)

Create a proximity constraint between a shape and a transform.

Parameters

- **shape** (*PyNode or str*) – Driver shape
- **in_trans** (*PyNode or str*) – in transform
- **existing_pin** (*PyNode, optional*) – Existing proximityPin node to connect to. Defaults to None.

Returns
out_trans, pin

Return type
Tuple[PyNode, PyNode]

mgear.core.applyop.**create_proximity_constraints**(*shape*, *in_trans_list*)

Create a proximity constraint between a shape and a list of transforms.

Parameters

- **shape** (*PyNode or str*) – Driver shape
- **in_trans_list** (*List[PyNode] or List[str]*) – List of in transforms

Returns
List of output transforms

Return type
List[PyNode]

mgear.core.applyop.**curvecns_op**(*crv*, *inputs=[]*)

mgear.core.applyop.**gear_curvecns_op**(*crv*, *inputs=[]*)

create mGear curvecns node.

Parameters

- **crv** (*nurbsCurve*) – Nurbs curve.
- **inputs** (*List of dagNodes*) – Input object to drive the curve. Should be same number as crv points. Also the order should be the same as the points

Returns
The curvecns node.

Return type
pyNode

`mgear.core.applyop.`**`gear_curveslide2_op`**(*outcrv*, *incrv*, *position=0*, *maxstretch=1*, *maxsquash=1*, *softness=0*)

Apply a sn_curveslide2_op operator

Parameters

- **`outcrv`** (*`NurbsCurve`*) – Out Curve.
- **`incrv`** (*`NurbsCurve`*) – In Curve.
- **`position`** (*`float`*) – Default position value (from 0 to 1).
- **`maxstretch`** (*`float`*) – Default maxstretch value (from 1 to infinite).
- **`maxsquash`** (*`float`*) – Default maxsquash value (from 0 to 1).
- **`softness`** (*`float`*) – Default softness value (from 0 to 1).

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.`**`gear_ikfk2bone_op`**(*out=[]*, *root=None*, *eff=None*, *upv=None*, *fk0=None*, *fk1=None*, *fk2=None*, *lengthA=5*, *lengthB=3*, *negate=False*, *blend=0*)

Apply a sn_ikfk2bone_op operator

Parameters

- **`out`** (*`list of dagNodes`*) – The constrained outputs order must be respected (BoneA, BoneB, Center, CenterN, Eff), set it to None if you don't want one of the output.
- **`root`** (*`dagNode`*) – Object that will act as the root of the chain.
- **`eff`** (*`dagNode`*) – Object that will act as the eff controler of the chain.
- **`upv`** (*`dagNode`*) – Object that will act as the up vector of the chain.
- **`fk0`** (*`dagNode`*) – Object that will act as the first fk controler of the chain.
- **`fk1`** (*`dagNode`*) – Object that will act as the second fk controler of the chain.
- **`fk2`** (*`dagNode`*) – Object that will act as the fk effector controler of the chain.
- **`lengthA`** (*`float`*) – Length of first bone.
- **`lengthB`** (*`float`*) – Length of second bone.
- **`negate`** (*`bool`*) – Use with negative Scale.
- **`blend`** (*`float`*) – Default blend value (0 for full ik, 1 for full fk).

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.gear_intmatrix_op`(*mA*, *mB*, *blend=0*)

create mGear interpolate Matrix node.

Parameters

- **mA** (*matrix*) – Input matrix A.
- **mB** (*matrix*) – Input matrix A.
- **blend** (*float or connection*) – Blending value.

Returns
Newly created mGear_intMatrix node

Return type
pyNode

`mgear.core.applyop.gear_inverseRotorder_op`(*out_obj*, *in_obj*)

Apply a sn_inverseRotorder_op operator

Parameters

- **out_obj** (*dagNode*) – Output object.
- **in_obj** (*dagNode*) – Input object.

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.gear_matrix_cns`(*in_obj*, *out_obj=None*, *connect_srt='srt'*, *rot_off=[0, 0, 0]*, *rot_mult=[1, 1, 1]*, *scl_mult=[1, 1, 1]*)

Create and connect matrix constraint node

Parameters

- **in_obj** (*transform*) – the driver object or matrix
- **out_obj** (*transform, optional*) – the driven object
- **connect_srt** (*str, optional*) – scale rotation traanslation flag
- **rot_off** (*list, optional*) – rotation offset for XYZ
- **rot_mult** (*list, optional*) – rotation multiplier for XYZ
- **scl_mult** (*list, optional*) – scale multiplier for XYZ

Returns
The matrix constraint node

Return type
PyNode

`mgear.core.applyop.gear_mulmatrix_op`(*mA*, *mB*, *target=False*, *transform='srt'*)

Create mGear multiply Matrix node.

Note: This node have same functionality as the default Maya matrix multiplication.

Parameters

- **mA** (*matrix*) – input matrix A.
- **mB** (*matrix*) – input matrix B.
- **target** (*dagNode*) – object target to apply the transformation
- **transform** (*str*) – if target is True. out transform to SRT valid value s r t

Returns
Newly created mGear_multMatrix node

Return type
pyNode

`mgear.core.applyop.`**`gear_raycast`**(*in_mesh*, *ray_source*, *ray_direction*, *out_obj=None*, *connect_srt='t'*)

Create and connect mraycast node

Parameters

- **in_mesh** (*shape*) – Mesh shape
- **ray_source** (*transform*) – ray source
- **ray_direction** (*transform*) – ray direction
- **out_obj** (*transform, optional*) – Object to apply the raycast contact
- **connect_srt** (*str, optional*) – scale rotation traanslation flag

Returns
The raycast node

Return type
PyNode

`mgear.core.applyop.`**`gear_rollsplinekine_op`**(*out*, *controlers=[]*, *u=0.5*, *subdiv=10*)

Apply a sn_rollsplinekine_op operator

Parameters

- **out** (*dagNode*) – onstrained Object.
- **controlers** (*list of dagNodes*) – Objects that will act as controler of the bezier curve. Objects must have a parent that will be used as an input for the operator.
- **u** (*float*) – Position of the object on the bezier curve (from 0 to 1).
- **subdiv** (*int*) – spline subdivision precision.

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.`**`gear_spinePointAtOp`**(*cns*, *startobj*, *endobj*, *blend=0.5*, *axis='-Z'*)

Apply a SpinePointAt operator

Parameters

- **cns** (*Constraint*) – The constraint to apply the operator on (must be a curve, path or direction constraint).
- **startobj** (*dagNode*) – Start Reference.
- **endobj** (*dagNode*) – End Reference.

- **blend** (*float*) – Blend influence value from 0 to 1.
- **axis** (*string*) – Axis direction.

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.`**gear_spinePointAtOpWM**(*cns*, *startobj*, *endobj*, *blend=0.5*, *axis='-Z'*)

Apply a SpinePointAt operator using world matrix

Parameters

- **Constraint** (*cns*) – The constraint to apply the operator on (must be a curve, path or direction constraint).
- **startobj** (*dagNode*) – Start Reference.
- **endobj** (*dagNode*) – End Reference.
- **blend** (*float*) – Blend influence value from 0 to 1.
- **axis** (*str*) – Axis direction.

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.`**gear_spring_op**(*in_obj*, *goal=False*)

Apply mGear spring node.

Parameters

- **in_obj** (*dagNode*) – Constrained object.
- **goal** (*dagNode*) – By default is False.

Returns
Newly created node

Return type
pyNode

`mgear.core.applyop.`**gear_squashstretch2_op**(*out*, *sclref=None*, *length=5*, *axis='x'*, *scaleComp=None*)

Apply a sn_squashstretch2_op operator

Parameters

- **out** (*dagNode*) – Constrained object.
- **sclref** (*dagNode*) – Global scaling reference object.
- **length** (*float*) – Rest Length of the S&S.
- **axis** (*str*) – 'x' for scale all except x axis…
- **scaleComp** (*list of float*) – extra scale compensation to avoid double scale in some situations.

Returns
The newly created operator.

Return type
pyNode

`mgear.core.applyop.oriCns(driver, driven, maintainOffset=False)`

Apply orientation constraint

Apply orientation constraint changing XYZ default connexions by rotate compound connexions

Note: We have found an evaluation difference in the values if the connexion is compound or by axis

Parameters

- **driver** (*dagNode or dagNode list*) – Driver object.
- **driven** (*dagNode*) – Driven object.
- **maintainOffset** (*bool*) – Keep the offset.

Returns
Orientation constraintn node.

Return type
pyNode

Example

```
import mgear.core.applyop as aop
import pymel.core as pm
sphere = pm.polySphere(n='sphereDriver')
cube = pm.polyCube(n='cubeDriven')
ori_cns = aop.oriCns(sphere[0], cube[0], True)
```

`mgear.core.applyop.parentCns(driver, driven, maintain_offset=True, **kwargs)`

Apply a parent constraint from driver to driven, skipping locked attributes, and allowing additional keyword arguments for the parentConstraint command.

Parameters

- **driver** (*pm.nt.Transform*) – The driver object to apply the constraint.
- **driven** (*pm.nt.Transform*) – The driven object to be constrained.
- **maintain_offset** (*bool*) – Whether to maintain the current offset.

Returns
The created parent constraint node.

Return type
pm.nodetypes.ParentConstraint

Raises
ValueError – If all translate and rotate attributes on the driven are locked.

`mgear.core.applyop.pathCns(obj, curve, cnsType=False, u=0, tangent=False)`

Apply a path constraint or curve constraint.

Parameters

- **obj** (*dagNode*) – Constrained object.

- **curve** (*Nurbscurve*) – Constraining Curve.
- **cnsType** (*int*) – 0 for Path Constraint, 1 for Curve Constraint (Parametric).
- **u** (*float*) – Position of the object on the curve (from 0 to 100 for path constraint, from 0 to 1 for Curve cns).
- **tangent** (*bool*) – Keep tangent orientation option.

Returns
The newly created constraint.

Return type
pyNode

`mgear.core.applyop.`**`splineIK`**(*name*, *chn*, *parent=None*, *cParent=None*, *curve=None*)

Apply a splineIK solver to a chain.

Parameters

- **name** (*str*) – Name of the operator node.
- **chn** (*list of joints*) – List of joints. At less 2 joints should be in the list.
- **parent** (*dagNode*) – Parent for the ikHandle.
- **cParent** (*dagNode*) – Parent for the curve.
- **curve** (*dagNode*) – Specifies the curve to be used by the ikSplineHandle. This param is optional.

Returns
ikHandle node and splinecrv in a list

Return type
list

Example

```
>>> aop.splineIK(self.getName("rollRef"),
                 self.rollRef,
                 parent=self.root,
                 cParent=self.bone0 )
```

mgear.core.attribute module

Attribute creation functions

class `mgear.core.attribute.`**`FCurveParamDef`**(*scriptName*, *keys=None*, *interpolation=0*, *extrapolation=0*)

Bases: *ParamDef*

Create an Fcurve parameter definition.

Parameters

- **scriptName** (*str*) – Attribute fullName.
- **keys** (*list*) – The keyframes to define the function curve.
- **interpolation** (*int*) – the curve interpolation.

- **extrapolation** (*int*) – the curve extrapolation.

create(*node*)

Add a parameter to property using the parameter definition.

Parameters
node (*dagNode*) – The node to add the attribute

get_as_dict()

set_from_dict(*param_dict*)

class mgear.core.attribute.**ParamDef**(*read as Parameter Definition*)

Bases: object

Encapsulate the attribute creation arguments in a handy object. Also include a creation method.

Example

This can be use later to create attr or export the description to xml or json file

Arguments:
scriptName (str): Attribute fullName paranDef (dic): The stored param definition

create(*node*)

Add a parameter to property using the parameter definition.

Parameters
node (*dagNode*) – The node to add the attribute

get_as_dict()

set_from_dict(*param_dict*)

class mgear.core.attribute.**ParamDef2**(*scriptName*, *valueType*, *value*, *niceName=None*, *shortName=None*, *minimum=None*, *maximum=None*, *keyable=True*, *readable=True*, *storable=True*, *writable=True*)

Bases: ParamDef

ParamDef2 inherit from ParamDef

Parameters

- **scriptName** (*str*) – Parameter scriptname.
- **valueType** (*str*) – The Attribute Type. Exp: 'string', 'bool', 'long', etc..
- **value** (*float or int*) – Default parameter value.
- **niceName** (*str*) – Parameter niceName.
- **shortName** (*str*) – Parameter shortName.
- **minimum** (*float or int*) – mininum value.
- **maximum** (*float or int*) – maximum value.
- **keyable** (*boo*) – If true is keyable
- **readable** (*boo*) – If true is readable
- **storable** (*boo*) – If true is storable
- **writable** (*boo*) – If true is writable

Returns
The stored parameter definition.

Return type
ParamDef

`mgear.core.attribute.`**`addAttribute`**(*node*, *longName*, *attributeType*, *value=None*, *niceName=None*, *shortName=None*, *minValue=None*, *maxValue=None*, *keyable=True*, *readable=True*, *storable=True*, *writable=True*, *channelBox=False*, *softMinValue=None*, *softMaxValue=None*)

Add attribute to a node

Parameters

- **node** (*dagNode*) – The object to add the new attribute.
- **longName** (*str*) – The attribute name.
- **attributeType** (*str*) – The Attribute Type. Exp: 'string', 'bool', 'long', etc..
- **value** (*float or int*) – The default value.
- **niceName** (*str*) – The attribute nice name. (optional)
- **shortName** (*str*) – The attribute short name. (optional)
- **minValue** (*float or int*) – minimum value. (optional)
- **maxValue** (*float or int*) – maximum value. (optional)
- **keyable** (*bool*) – Set if the attribute is keyable or not. (optional)
- **readable** (*bool*) – Set if the attribute is readable or not. (optional)
- **storable** (*bool*) – Set if the attribute is storable or not. (optional)
- **writable** (*bool*) – Set if the attribute is writable or not. (optional)
- **channelBox** (*bool*) – Set if the attribute is in the channelBox or not, when the attribute is not keyable. (optional)

Returns
The long name of the new attribute

Return type
str

`mgear.core.attribute.`**`addColorAttribute`**(*node*, *longName*, *value=False*, *keyable=True*, *readable=True*, *storable=True*, *writable=True*, *niceName=None*, *shortName=None*)

Add a color attribute to a node

Parameters

- **node** (*dagNode*) – The object to add the new attribute.
- **longName** (*str*) – The attribute name.
- **value** (*list of flotat*) – The default value in a list for RGB. exp [1.0, 0.99, 0.13].
- **keyable** (*bool*) – Set if the attribute is keyable or not. (optional)
- **readable** (*bool*) – Set if the attribute is readable or not. (optional)
- **storable** (*bool*) – Set if the attribute is storable or not. (optional)
- **writable** (*bool*) – Set if the attribute is writable or not. (optional)

- **niceName** (*str*) – The attribute nice name. (optional)
- **shortName** (*str*) – The attribute short name. (optional)

Returns
The long name of the new attribute

Return type
str

`mgear.core.attribute.`**`addEnumAttribute`**(*node*, *longName*, *value*, *enum*, *niceName=None*, *shortName=None*, *keyable=True*, *readable=True*, *storable=True*, *writable=True*)

Add an enumerate attribute to a node

Parameters

- **node** (*dagNode*) – The object to add the new attribute.
- **longName** (*str*) – The attribute name.
- **value** (*int*) – The default value.
- **enum** (*list of str*) – The list of elements in the enumerate control
- **niceName** (*str*) – The attribute nice name. (optional)
- **shortName** (*str*) – The attribute short name. (optional)
- **keyable** (*bool*) – Set if the attribute is keyable or not. (optional)
- **readable** (*bool*) – Set if the attribute is readable or not. (optional)
- **storable** (*bool*) – Set if the attribute is storable or not. (optional)
- **writable** (*bool*) – Set if the attribute is writable or not. (optional)

Returns
The long name of the new attribute

Return type
str

`mgear.core.attribute.`**`addFCurve`**(*node*, *name='fcurve'*, *keys=[]*)

FCurve attribute

Just a animCurveUU node connected to an attribute

> **Warning:** This Method is deprecated.

Parameters

- **node** (*dagNode*) – The object to add the new fcurve attribute
- **name** (*str*) – The attribute name
- **key** (*list*) – list of keyframes and values

Returns
Fcurve and attribute name

mgear.core.attribute.**addProxyAttribute**(*sourceAttrs*, *targets*, *duplicatedPolicy=None*)

Add proxy paramenter to a list of target dagNode Duplicated channel policy, stablish the rule in case the channel already exist on the target.

Duplicate policy options

index	This policy will add an index to avoid clashing channel names
fullName	This policy will add the name of the source object to the channel
merge	This policy will merge the channels

Parameters

- **sourceAttrs** (*attr or list of attr*) – The parameters to be connected as proxy
- **targets** (*dagNode or list of dagNode*) – The list of dagNode to add the proxy paramenter
- **duplicatedPolicy** (*string, optional*) – Set the duplicated channel policy

mgear.core.attribute.**addVector3Attribute**(*node*, *longName*, *value=False*, *keyable=True*, *readable=True*, *storable=True*, *writable=True*, *niceName=None*, *shortName=None*, *childLabels=['X', 'Y', 'Z']*, *usedAsColor=False*, *attributeType='float3'*)

Add a vector3 attribute to a node

Parameters

- **node** (*dagNode*) – The object to add the new attribute.
- **longName** (*str*) – The attribute name.
- **value** (*list of flotat*) – The default value in a list for RGB. exp [1.0, 0.99, 0.13].
- **keyable** (*bool*) – Set if the attribute is keyable or not. (optional)
- **readable** (*bool*) – Set if the attribute is readable or not. (optional)
- **storable** (*bool*) – Set if the attribute is storable or not. (optional)
- **writable** (*bool*) – Set if the attribute is writable or not. (optional)
- **niceName** (*str*) – The attribute nice name. (optional)
- **shortName** (*str*) – The attribute short name. (optional)

Returns
The long name of the new attribute

Return type
str

mgear.core.attribute.**add_mirror_config_channels**(*ctl*, *conf=[0, 0, 0, 0, 0, 0, 0, 0, 0]*)

Add channels to configure the mirror posing

Parameters
ctl (*dagNode*) – Control Object

mgear.core.attribute.**change_default_value**(*attributes*, *defaultValue*)

Change the default value of the attr

Parameters

- **attributes** (*list or str or pynode*) – attributes to change the default val
- **defaultValue** (*numeric value*) – defaul value. Only numeric values are valid

mgear.core.attribute.**collect_attrs**(*node*, *attrs*, *attrs_list*, *shapes=False*)

Collect the channel full path in a list.

Checks that the channel is not repeated.

Parameters

- **node** (*PyNode*) – Node that owns the channels
- **attrs** (*str*) – the list to add the channels full path
- **attrs_list** (*list*) – the list of channels names
- **shapes** (*bool, optional*) – If True will search the attr only in shapes

class mgear.core.attribute.**colorParamDef**(*scriptName*, *value=False*)

Bases: *ParamDef*

Create a Color parameter definition.

Parameters

- **scriptName** (*str*) – Attribute fullName
- **value** (*list of float*) – The default value in a list for RGB. exp [1.0, 0.99, 0.13].

create(*node*)

Add a parameter to property using the parameter definition.

Parameters

node (*dagNode*) – The node to add the attribute

get_as_dict()

set_from_dict(*param_dict*)

mgear.core.attribute.**connectSet**(*source*, *target*, *testInstance*)

Connect or set attributes

Connects or set attributes depending if is instance of a instance check

Parameters

- **source** (*str or Attr*) – Striname of the attribute or PyNode attribute
- **target** (*str or Attr*) – Striname of the attribute or PyNode attribute
- **testInstance** (*tuple*) – Tuple of types to check

mgear.core.attribute.**connect_add_dynamic_pivot**(*pivots*, *driven*)

connect dynamic pivot with option to add offset channels on XYZ

Parameters

- **pivot** (*dagNode list*) – pivot translation
- **driven** (*dagNode list*) – Driven object

mgear.core.attribute.**connect_dynamic_pivot**(*pivot*, *driven*)

connects translation of pivot dagNode to rotatePivot and scalePivot of the driven transform

Parameters

- **pivot** (*dagNode*) – pivot translation
- **driven** (*dagNode*) – Driven object

`mgear.core.attribute.`**`connect_message`**(*source*, *attr*)

Connects the 'message' attribute of one or more source nodes to a destination attribute.

Parameters

- **source** (*str, pm.PyNode, list*) – The source node(s) with a 'message' attribute.
- **attr** (*str, pm.PyNode*) – The destination attribute to connect to.

Raises

TypeError – If the destination attribute is not a message attribute.

`mgear.core.attribute.`**`disconnect_inputs`**(*node*, *attributes=['scale', 'sx', 'sy', 'sz', 'translate', 'tx', 'ty', 'tz', 'rotate', 'rx', 'ry', 'rz']*)

Disconnects only the input connections of the specified attributes of the provided node.

Parameters

- **node** (*PyNode*) – The PyNode object for which to disconnect input
- **connections.** –
- **attributes** (*list, optional*) – A list of attributes to disconnect
- **translation,** (*input connections from. Defaults to disconnecting all*) –
- **directions.** (*rotation, and scale attributes in all*) –

`mgear.core.attribute.`**`disconnect_outputs`**(*node*, *attributes=['scale', 'sx', 'sy', 'sz', 'translate', 'tx', 'ty', 'tz', 'rotate', 'rx', 'ry', 'rz']*)

Disconnects only the output connections of the specified attributes of the provided node.

Parameters

- **node** (*PyNode*) – The PyNode object for which to disconnect output
- **connections.** –
- **attributes** (*list, optional*) – A list of attributes to disconnect
- **translation,** (*output connections from. Defaults to disconnecting all*) –
- **directions.** (*rotation, and scale attributes in all*) –

class `mgear.core.attribute.`**`enumParamDef`**(*scriptName*, *enum*, *value=False*)

Bases: `ParamDef`

Create an enumarator parameter definition.

Parameters

- **scriptName** (*str*) – Attribute fullName
- **enum** (*list of str*) – The list of elements in the enumerate control.
- **value** (*int*) – The default value.

`create`(*node*)

Add a parameter to property using the parameter definition.

Parameters

node (*dagNode*) – The node to add the attribute

get_as_dict()

set_from_dict(*param_dict*)

mgear.core.attribute.**getSelectedChannels**(*userDefine=False*)

Get the selected channels on the channel box

Parameters
userDefine (*bool, optional*) – If True, will return only the user defined channels. Other channels will be skipped.

Returns
The list of selected channels names

Return type
list

mgear.core.attribute.**getSelectedObjectChannels**(*oSel=None*, *userDefine=False*, *animatable=False*)

Get the selected object channels.

Parameters

- **oSel** (*None, optional*) – The pynode with channels to get
- **userDefine** (*bool, optional*) – If True, will return only the user defined channels. Other channels will be skipped.
- **animatable** (*bool, optional*) – If True, only animatable parameters will be return

Returns
The list of the selected object channels names

Return type
list

mgear.core.attribute.**get_channelBox**()

Get the channel box

Returns
channel box path

Return type
str

mgear.core.attribute.**get_default_value**(*node*, *attribute*)

Get the default attribute value

Parameters

- **node** (*str, PyNode*) – The object with the attribute
- **attribute** (*str*) – The attribute to get the value

Returns
The attribute value

Return type
variant

mgear.core.attribute.**get_next_available_index**(*attr*)

get the next available index from a multi attr This function is a workaround because the connect attr flag next available is not working.

The connectAttr to the children attribute is giving error

i.e: pm.connectAttr(ctt.attr("parent"),
tpTagNode.attr("children"), na=True)

if using the next available option flag I was expecting to use ctt.setParent(tagParent) but doest't work as expected. After reading the documentation this method looks prety useless. Looks like is boolean and works based on selection :(

Parameters
attr (*attr*) – Attr multi

Returns
index

Return type
int

`mgear.core.attribute.`**`get_selected_channels_full_path`**`()`

Get the selected channels full path from channel box This function will collect channels from any area of the channel box. This include, Main, shape, input and output

Returns
list of channels full path

Return type
list

`mgear.core.attribute.`**`has_in_connections`**`(`*node*, *attributes=['translate', 'tx', 'ty', 'tz', 'rotate', 'rx', 'ry', 'rz', 'scale', 'sx', 'sy', 'sz']*`)`

Checks if the provided node has any input connections on the specified attributes.

Parameters

- **node** (*PyNode*) – The PyNode object for which to check connections.
- **attributes** (*list, optional*) – A list of attributes to check for connections. Defaults to checking all translation,
- **directions.** (*rotation, and scale attributes in all*) –

Returns
True if any connections are found, False otherwise.

Return type
bool

Examples

```
>>> jnt = pm.PyNode('myJoint')
>>> has_connections(jnt)
Found some connections.
>>> has_connections(jnt, ['visibility'])
No connections found.
```

`mgear.core.attribute.`**`lockAttribute`**`(`*node*, *attributes=['tx', 'ty', 'tz', 'rx', 'ry', 'rz', 'sx', 'sy', 'sz', 'v']*`)`

Lock attributes of a node.

By defaul will lock the rotation, scale and translation.

Parameters

- **node** (*dagNode*) – The node with the attributes to lock.
- **attributes** (*list of str*) – The list of the attributes to lock.

Example

```
>>> att.lockAttribute(self.root_ctl, ["sx", "sy", "sz", "v"])
```

mgear.core.attribute.**moveChannel**(*attr*, *sourceNode*, *targetNode*, *duplicatedPolicy=None*)

Move channels keeping the output connections. Duplicated channel policy, stablish the rule in case the channel already exist on the target.

NOTE: For the moment move channel only supports type double and enum

Duplicate policy options

index	This policy will add an index to avoid clashing channel names
fullName	This policy will add the name of the source object to the channel
merge	This policy will merge the channels

Parameters

- **attr** (*str*) – Name of the channel to move
- **sourceNode** (*PyNoe or str*) – The source node with the channel
- **targetNode** (*PyNoe or str*) – The target node for the channel
- **duplicatedPolicy** (*None, str*) – Set the duplicated channel policy

mgear.core.attribute.**move_input_connections**(*source*, *target*, *type_filter=None*)

Move the input connections from source node to target node. The connections can be filtered by type of the incoming node

exp: attribute.move_input_connections(

cns_obj, cns_off, type_filter="parentConstraint"

)

Parameters

- **source** (*PyNode*) – node with input connection
- **target** (*PyNode*) – target to receive the input connections
- **type_filter** (*None, str*) – node type to filter connections list

mgear.core.attribute.**move_output_connections**(*source*, *target*, *type_filter=None*)

Move the output connections from source node to target node. The connections can be filtered by type of the outgoing node

exp: attribute.move_output_connections(

cns_obj, cns_off, type_filter="parentConstraint"

)

Parameters

- **source** (*PyNode*) – node with output connection
- **target** (*PyNode*) – target to receive the input connections
- **type_filter** (*None, str*) – node type to filter connections list

mgear.core.attribute.**reset_SRT**(*objects=None*, *attributes=['tx', 'ty', 'tz', 'rx', 'ry', 'rz', 'sx', 'sy', 'sz', 'v', 'ro']*)

Reset Scale Rotation and translation attributes to default value

Parameters

- **objects** (*None, optional*) – The objects to reset the channels
- **attribute** (*list*) – The attribute to reset

mgear.core.attribute.**reset_selected_channels_value**(*objects=None*, *attributes=None*)

Reset the the selected channels if not attribute is provided

Parameters

- **objects** (*None, optional*) – The objects to reset the channels
- **attribute** (*list, optional*) – The attribute to reset

mgear.core.attribute.**setInvertMirror**(*node*, *invList=None*)

Set invert mirror pose values

Parameters

node (*dagNode*) – The object to set invert mirror Values

mgear.core.attribute.**setKeyableAttributes**(*nodes*, *params=['tx', 'ty', 'tz', 'ro', 'rx', 'ry', 'rz', 'sx', 'sy', 'sz']*)

Set keyable attributes of a node.

By defaul will set keyable the rotation, scale and translation.

Parameters

- **node** (*dagNode*) – The node with the attributes to set keyable.
- **attributes** (*list of str*) – The list of the attributes to set keyable. Attrs not in the list will be locked if None, [“tx”, “ty”, “tz”, “rorder”, “rx”, “ry”, “rz”, “sx”, “sy”,

  ”sz”] is used

mgear.core.attribute.**setNotKeyableAttributes**(*nodes*, *attributes=['tx', 'ty', 'tz', 'ro', 'rx', 'ry', 'rz', 'sx', 'sy', 'sz', 'v']*)

Set not keyable attributes of a node.

By defaul will set not keyable the rotation, scale and translation.

Parameters

- **node** (*dagNode*) – The node with the attributes to set keyable.
- **attributes** (*list of str*) – The list of the attributes to set not keyable

mgear.core.attribute.**setRotOrder**(*node*, *s='XYZ'*)

Set the rotorder of the object.

Parameters

- **node** (*dagNode*) – The object to set the rot order on.
- **s** (*str*) – Value of the rotorder. Possible values : (“XYZ”, “XZY”, “YXZ”, “YZX”, “ZXY”, “ZYX”)

mgear.core.attribute.**set_default_value**(*node*, *attribute*)

Set the default value to the attribute

Parameters

- **node** (*str, PyNode*) – The object with the attribute to reset
- **attribute** (*str*) – The attribute to reset

mgear.core.attribute.**smart_reset**(**args*)

Reset the SRT or the selected channels

Checks first if we have channels selected. If not, will try to reset SRT

Parameters

***args** – Dummy

mgear.core.attribute.**toggle_bool_attr**(*attr*)

mgear.core.attribute.**unlockAttribute**(*node*, *attributes=['tx', 'ty', 'tz', 'rx', 'ry', 'rz', 'sx', 'sy', 'sz', 'v']*)

Unlock attributes of a node.

By defaul will unlock the rotation, scale and translation.

Parameters

- **node** (*dagNode*) – The node with the attributes to unlock.
- **attributes** (*list of str*) – The list of the attributes to unlock.

Example

```
>>> att.unlockAttribute(self.root_ctl, ["sx", "sy", "sz", "v"])
```

mgear.core.callbackManager module

API for creating, deleting, debuging callbacks

examples ——————————————————————— # module cb.selectionChangedCB("testingSessssslection", cb.testFunc) cb.RECORDED_CALLBACKS cb.removeAllSessionCB()

manager A e = cb.CallbackManager() e.namespace e.debug = False e.selectionChangedCB("synopticUI1", cb.testFunc) e.newSceneCB("synopticNewScene", cb.testFunc) e.removeManagedCB("synopticUI1") e.removeManagedCB("synopticNewScene")

e.attributeChangedCB("attrChanged", cb.testFunc, "pSphere1", ["tx"]) e.removeManagedCB("attrChanged")

e.MANAGER_CALLBACKS e.removeAllManagedCB()

manager b r = cb.CallbackManager() r.selectionChangedCB("synopticUI1", cb.testFunc) r.MANAGER_CALLBACKS r.removeAllManagedCB()

__author__ = "Rafael Villar" __email__ = "rav@ravrigs.com"

class mgear.core.callbackManager.**AttributeChangedManager**(*m_node*, *attributes*, *func*)

Bases: object

mini class that will be called upon when attrChanged callback is run this will check the plugs passed in to see if it is an attr of desired name `attributes`

[tx, ty] of SHORT NAMED attrs to monitor

Type
list

`func`

to call when criteria met

Type
function

`m_node`

mobject

Type
om.MOBJECT

`attributeChanged`(*id*, *plug1*, *plug2*, *payload*)

actual function that will be called when attrChanged callback is created

Parameters

- **id** (*int*) – 2056 is the desired, attr changed id
- **plug1** (*om.MPlug*) – MPlug attr to query

Returns
n/a

Return type
n/a

class `mgear.core.callbackManager.CallbackManager`

Bases: `object`

Convenience to manage callbacks

`debug`

should callbacks created by manager produce print outs

Type
bool

`MANAGER_CALLBACKS`

record of all created callbacks

Type
dict

`namespace`

namespace to put callbacks under

Type
str

`addNamespace`(*callback_name*)

used in the decorator, add namespace to any name provided

Parameters
callback_name (*str*) – name of callback

Returns
provided name with namespace

Return type
str

attributeChangedCB(*callback_name*, *func*, *node*, *attributes*)

checkDebug(*debugInfo*, **args*)

safely check if manager is in debug mode

Parameters

- **debugInfo** (*list*) – callback name, function being called with args
- ***args** – args to pass to the function associated with callback

newSceneCB(*callback_name*, *func*)

registerManagerCB()

decorator, adds debug and namespace to every callback created

Parameters
func (*function*) – function to wrap

Returns
wrapped function

Return type
function

removeAllManagedCB()

remove all the callbacks created my this manager

removeManagedCB(*callback_name*)

remove specific callback under this manager

Parameters
callback_name (*str*) – name

selectionChangedCB(*callback_name*, *func*)

setNamespace(*namespace*)

set the namespace to put callbacks under

Parameters
namespace (*str*) – desired namespace

timeChangedCB(*callback_name*, *func*)

userTimeChangedCB(*callback_name*, *func*)

wrapWithDebug(*debugInfo*, *func*)

so every function that is associated with a callback is swapped out for this one, so it will check for debugging

Parameters

- **debugInfo** (*list*) – callback name, mayacallback id, functions to call
- **func** (*function*) – to wrap with this debug checker

Returns
partial function that will check for debug

Return type
function

class mgear.core.callbackManager.**UserTimeChangedManager**(*func*)

Bases: object

mini class that will be called upon when timeChanged callback is run this will check to see if playback is active, if so BREAK

attributes
[tx, ty] of SHORT NAMED attrs to monitor

Type
list

func
to call when criteria met

Type
function

m_node
mobject

Type
om.MOBJECT

userTimeChanged(**args*)
Check if playback is active, if so return without calling func

mgear.core.callbackManager.**attributeChangedCB**(*callback_name*, *func*, *node*, *attributes*)
call the provided function any of the provided attributes are changed

Parameters

- **callback_name** (*str*) – name of the callback
- **func** (*function*) – to be called upon
- **node** (*str*) – name of node to montior for attr changes
- **attributes** (*list*) – of SHORTNAMED attributes to monitor

Returns
maya id to created callback

Return type
long

mgear.core.callbackManager.**checkAndRecordCB**(*callback_name*, *callback_id*, *callback_info={}*)
will remove any callbacks of the same name prior to creating a new one

Parameters

- **callback_name** (*str*) – desired name of the callback, readable
- **callback_id** (*long*) – api method of identifying callbacks

mgear.core.callbackManager.**getMObject**(*node*)
get the mobject of any name provided, so it can have cb's assined to it

Parameters
node (*str*) – of node

Returns
MOBJECT

Return type
om.MObject

`mgear.core.callbackManager.newSceneCB(callback_name, func)`

When a new scene is opened, call the provided function

Parameters

- **callback_name** (*str*) – name you want to assign cb
- **func** (*function*) – will be called upon

Returns
maya id to created callback

Return type
long

`mgear.core.callbackManager.registerSessionCB(func)`

decorator to ensure that every callback created is recorded

Parameters
func (*function*) – function that will create the callback

Returns
function

Return type
function

`mgear.core.callbackManager.removeAllCBFromNode(node)`

remove all callbacks from the provided node

Parameters
node (*str*) – name of node to remove callbacks from

`mgear.core.callbackManager.removeAllSessionCB()`

Remove all the callbacks created in this session, provided they are in the RECORDED_CALLBACKS dict

`mgear.core.callbackManager.removeCB(callback_identifier, callback_info={})`

Remove callback from scene and RECORDED_CALLBACKS(or provided dict)

Parameters

- **callback_identifier** (*str*) – name of callback
- **callback_info** (*dict, optional*) – dict to remove callback from

`mgear.core.callbackManager.removeCBviaMayaID(mayaID, callback_info={})`

This if have the maya pointer only, this will remove it from the recorded_callbacks as well

Parameters

- **mayaID** (*long*) – maya point to a callback
- **callback_info** (*dict, optional*) – remove it from desired cb recorder

`mgear.core.callbackManager.removeNamespaceCB(namespace)`

Remove all callbacks under the provided namespace

Parameters
`namespace` (*str*) – uuid or other type of namespace

`mgear.core.callbackManager.sampleCallback(callback_name, func)`

argument order is important. Callback_name and func must always be first must always return the mayaID to the callback :param callback_name: name you want to assign cb :type callback_name: str :param func: will be called upon :type func: function

Returns
maya id to created callback

Return type
long

`mgear.core.callbackManager.selectionChangedCB(callback_name, func)`

When the selection is changed call the provided function

Parameters

- `callback_name` (*str*) – name you want to assign cb
- `func` (*function*) – will be called upon

Returns
maya id to created callback

Return type
long

`mgear.core.callbackManager.testFunc(*args)`

test function used for debugging/dev

Parameters
`*args` – things that will printed

`mgear.core.callbackManager.timeChangedCB(callback_name, func)`

ANYTIME the time is changed, call the provided function

Parameters

- `callback_name` (*str*) – name you want to assign cb
- `func` (*function*) – will be called upon

Returns
maya id to created callback

Return type
long

`mgear.core.callbackManager.userTimeChangedCB(callback_name, func)`

Callback triggers during user timeChange, skips PLAYBACK

Parameters

- `callback_name` (*str*) – name you want to assign cb
- `func` (*function*) – will be called upon

Returns
maya id to created callback

Return type
long

mgear.core.curve module

NurbsCurve creation functions

`mgear.core.curve.`**`addCnsCurve`**(*parent*, *name*, *centers*, *degree=1*)

Create a curve attached to given centers. One point per center

Parameters

- **parent** (*dagNode*) – Parent object.
- **name** (*str*) – Name
- **centers** (*list of dagNode*) – Object that will drive the curve.
- **degree** (*int*) – 1 for linear curve, 3 for Cubic.

Returns
The newly created curve.

Return type
dagNode

`mgear.core.curve.`**`addCurve`**(*parent*, *name*, *points*, *close=False*, *degree=3*, *m=pymel.core.datatypes.Matrix*, *op=False*)

Create a NurbsCurve with a single subcurve.

Parameters

- **parent** (*dagNode*) – Parent object.
- **name** (*str*) – Name
- **points** (*list of float*) – points of the curve in a one dimension array [point0X, point0Y, point0Z, 1, point1X, point1Y, point1Z, 1, . . .].
- **close** (*bool*) – True to close the curve.
- **degree** (*bool*) – 1 for linear curve, 3 for Cubic.
- **m** (*matrix*) – Global transform.
- **op** (*bool, optional*) – If True will add a curve that pass over the points This is equivalent of using"editPoint " flag

No Longer Returned:
dagNode: The newly created curve.

`mgear.core.curve.`**`add_linear_skinning_to_curve`**(*curve_name*, *joint_list*)

Adds a skinCluster to a curve and sets the skinning weights linearly among the list of joints based on the number of control points.

Parameters

- **curve_name** (*str*) – The name of the curve to add the skinCluster to.
- **joint_list** (*list*) – A list of joint names to be included in the skinCluster.

Returns
The name of the created skinCluster.

Return type
PyNode

`mgear.core.curve.`**`average_curve`**(*crv*, *shapes*, *average=2*, *avg_shape=False*, *avg_scl=False*, *avg_rot=False*)

Average the shape, rotation and scale of the curve bettwen n number of curves

Parameters

- **crv** (*dagNode*) – curve to average shape
- **shapes** (*[dagNode]]*) – imput curves to average the shapes
- **average** (*int, optional*) – Number of curves to use on the average
- **avg_shape** (*bool, optional*) – if True will interpolate curve shape
- **avg_scl** (*bool, optional*) – if True will interpolate curve scale
- **avg_rot** (*bool, optional*) – if True will interpolate curve rotation

`mgear.core.curve.`**`collect_curve_data`**(*objs*, *rplStr=['', '']*)

Generate a dictionary descriving the curve data

Suport multiple objects

Parameters

- **objs** (*dagNode*) – Curve object to store
- **collect_trans** (*bool, optional*) – if false will skip the transformation matrix
- **rplStr** (*list, optional*) – String to replace in names. This allow to change the curve names before store it. [old Name to replace, new name to set]

Returns
Curves data

Return type
dict

`mgear.core.curve.`**`collect_curve_shapes`**(*crv*, *rplStr=['', '']*)

Collect curve shapes data

Parameters

- **crv** (*dagNode*) – Curve object to collect the curve shapes data
- **rplStr** (*list, optional*) – String to replace in names. This allow to change the curve names before store it. [old Name to replace, new name to set]

Returns
Curve shapes dictionary and curve shapes names

Return type
dict, list

`mgear.core.curve.`**`collect_selected_curve_data`**(*objs=None*, *rplStr=['', '']*)

Generate a dictionary descriving the curve data from selected objs

Parameters
objs (*None, optional*) – Optionally a list of object can be provided

`mgear.core.curve.`**`cox_de_boor`**(*u*, *i*, *p*, *knots*)

Cox-De Boor algorithm to evaluate B-Spline basis function.

Parameters

- **u** (*float*) – Parameter value.

- **i** (*int*) – Index of control point.
- **p** (*int*) – Degree of the curve.
- **knots** (*list*) – Knot vector.

Returns
Evaluated B-Spline basis function value.

Return type
float

mgear.core.curve.**createCurveFromCurve**(*srcCrv*, *name*, *nbPoints*, *parent=None*)

Create a curve from a curve

Parameters

- **srcCrv** (*curve*) – The source curve.
- **name** (*str*) – The new curve name.
- **nbPoints** (*int*) – Number of control points for the new curve.
- **parent** (*dagNode*) – Parent of the new curve.

Returns
The newly created curve.

Return type
dagNode

mgear.core.curve.**createCurveFromOrderedEdges**(*edgeLoop*, *startVertex*, *name*, *parent=None*, *degree=3*)

Create a curve for a edgeloop ordering the list from starting vertex

Parameters

- **edgeLoop** (*list*) – List of edges
- **startVertex** (*vertex*) – Starting vertex
- **name** (*str*) – Name of the new curve.
- **parent** (*dagNode*) – Parent of the new curve.
- **degree** (*int*) – Degree of the new curve.

Returns
The newly created curve.

Return type
dagNode

mgear.core.curve.**createCuveFromEdges**(*edgeList*, *name*, *parent=None*, *degree=3*, *sortingAxis='x'*)

Create curve from a edge list.

Parameters

- **edgeList** (*list*) – List of edges.
- **name** (*str*) – Name of the new curve.
- **parent** (*dagNode*) – Parent of the new curve.
- **degree** (*int*) – Degree of the new curve.
- **sortingAxis** (*str*) – Sorting axis x, y or z

Returns
The newly created curve.

Return type
dagNode

`mgear.core.curve.`**`create_curve_from_data`**(*data*, *replaceShape=False*, *rebuildHierarchy=False*, *rplStr=['', '']*, *model=None*)

Build the curves from a given curve data dict

Hierarchy rebuild after all curves are build to avoid lost parents

Parameters

- **data** (*dict*) – serialized curve data
- **replaceShape** (*bool, optional*) – If True, will replace the shape on existing objects
- **rebuildHierarchy** (*bool, optional*) – If True, will regenerate the hierarchy

`mgear.core.curve.`**`create_curve_from_data_by_name`**(*crv*, *data*, *replaceShape=False*, *rebuildHierarchy=False*, *rplStr=['', '']*, *model=None*)

Build one curve from a given curve data dict

Parameters

- **crv** (*str*) – name of the crv to create
- **data** (*dict*) – serialized curve data
- **replaceShape** (*bool, optional*) – If True, will replace the shape on existing objects
- **rebuildHierarchy** (*bool, optional*) – If True, will regenerate the hierarchy
- **rplStr** (*list, optional*) – String to replace in names. This allow to change the curve names before store it. [old Name to replace, new name to set]
- **model** (*dagNode, optional*) – Model top node to help find the correct parent, if several objects with the same name

`mgear.core.curve.`**`create_locator_at_curve_point`**(*object_names*, *percentage*)

Create a locator at a point on a cubic NURBS curve in Maya.

Parameters

- **object_names** (*list*) – The names of the objects representing control points in Maya.
- **percentage** (*float*) – Curve position as a percentage (0 to 100).

Example usage in Maya Select objects representing control points in Maya before running the script

object_names = cmds.ls(selection=True) create_locator_at_curve_point(object_names, 100)

`mgear.core.curve.`**`crv_parenting`**(*data*, *crv*, *rplStr=['', '']*, *model=None*)

Parent the new created curves

Parameters

- **data** (*dict*) – serialized curve data
- **crv** (*str*) – name of the curve to parent
- **rplStr** (*list, optional*) – String to replace in names. This allow to change the curve names before store it. [old Name to replace, new name to set]

- **model** (*dagNode, optional*) – Model top node to help find the correct parent, if several objects with the same name

mgear.core.curve.**curl_curve**(*crvs*, *amount=0.3*, *frequency=10*)

mgear.core.curve.**evaluate_cubic_nurbs**(*control_points*, *percentage*, *knots=None*, *weights=None*)

Evaluate a cubic NURBS curve at a given percentage.

Parameters

- **control_points** (*list*) – List of control points, each as [x, y, z].
- **percentage** (*float*) – Curve position as a percentage (0 to 100).
- **knots** (*list, optional*) – Knot vector.
- **weights** (*list, optional*) – List of weights corresponding to control points.

Returns

Evaluated point as [x, y, z].

Return type

list

mgear.core.curve.**export_curve**(*filePath=None*, *objs=None*, *rplStr=['', '']*)

Export the curve data to a json file

Parameters

- **filePath** (*None, optional*) – Description
- **objs** (*None, optional*) – Description

Returns

Description

Return type

TYPE

mgear.core.curve.**findLenghtFromParam**(*crv*, *param*)

Find lengtht from a curve parameter

Parameters

- **param** (*float*) – The parameter to get the legth
- **crv** (*curve*) – The source curve.

Returns

Curve uLength

Return type

float

Example

```
oParam, oLength = cur.getCurveParamAtPosition(upRope, cv)
uLength = cur.findLenghtFromParam(upRope, oParam)
u = uLength / oLength
```

mgear.core.curve.**getCurveParamAtPosition**(*crv*, *position*)

Get curve parameter from a position

Parameters

- **position** (*list of float*) – Represents the position in worldSpace exp: [1.4, 3.55, 42.6]
- **crv** (*curve*) – The source curve to get the parameter.

Returns

paramenter and curve length

Return type

list

mgear.core.curve.**getParamPositionsOnCurve**(*srcCrv*, *nbPoints*)

get param position on curve

Parameters

- **srcCrv** (*curve*) – The source curve.
- **nbPoints** (*int*) – Number of points to return.

Returns

world positions.

Return type

tuple

mgear.core.curve.**get_color**(*node*)

Get the color from shape node

Parameters

node (*TYPE*) – shape

Returns

Description

Return type

TYPE

mgear.core.curve.**get_uniform_world_positions_on_curve**(*curve*, *num_positions*)

Get a specified number of uniformly distributed world positions along a NURBS curve.

Parameters

- **curve** (*str or PyNode*) – The name or PyNode of the NURBS curve.
- **num_positions** (*int*) – The number of uniformly distributed positions to return.

Returns

A list of tuples, where each tuple represents a world

position (x, y, z).

Return type

tuple

mgear.core.curve.**import_curve**(*filePath=None*, *replaceShape=False*, *rebuildHierarchy=False*, *rplStr=['', '']*)

mgear.core.curve.**keep_lock_length_state**(*func*)

mgear.core.curve.**keep_point_0_cnx_state**(*func*)

mgear.core.curve.**lock_first_point**(*crv*)

mgear.core.curve.**lock_length**(*crv*, *lock=True*)

mgear.core.curve.**rebuild_curve**(*crvs*, *spans*)

mgear.core.curve.**set_color**(*node*, *color*)

Set the color in the Icons.

Parameters

- **node** (*dagNode*) – The object
- **color** (*int or list of float*) – The color in index base or RGB.

mgear.core.curve.**set_thickness**(*crv*, *thickness=-1*)

mgear.core.curve.**smooth_curve**(*crvs*, *smooth_factor=1*)

mgear.core.curve.**straighten_curve**(*crvs*, *straighteness=0.1*, *keep_lenght=1*)

mgear.core.curve.**update_curve_from_data**(*data*, *rplStr=['', '']*)

update the curves from a given curve data dict

Parameters

data (*dict*) – serialized curve data

mgear.core.curve.**update_curve_from_file**(*filePath=None*, *rplStr=['', '']*)

mgear.core.dag module

Nvigate the DAG hierarchy

mgear.core.dag.**findChild**(*node*, *name*)

Returns the first child of input node, with a matching name.

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search

Returns

The first child

Return type

dagNode

```
>>> parent = dag.findChild(self.model,
                           mgear.string.convertRLName(
                               comp_guide.root.name()))
```

`mgear.core.dag.findChildren(node, name)`

Returns all the children of input node, with a matching name.

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search

Returns
The children dagNodes

Return type
dagNode list

`mgear.core.dag.findChildrenPartial(node, name)`

Returns the children of input node, with a partial matching name.

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search

Returns
The children dagNodes

Return type
dagNode list

`mgear.core.dag.findComponentChildren(node, name, sideIndex)`

Returns the component children of input component root.

Note: This method is specific to work with shifter guides naming conventions

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search
- **sideIndex** (*str*) – the side

Returns
The children dagNodes

Return type
dagNode list

```
>>> objList =  dag.findComponentChildren(self.parent,
                                         oldName,
                                         oldSideIndex)
```

`mgear.core.dag.findComponentChildren2(node, name, sideIndex)`

Returns the component children of input component root.

This function is using Maya cmds instead of PyMel

Note: This method is specific to work with shifter guides naming conventions

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search
- **sideIndex** (*str*) – the side

Returns
The children dagNodes

Return type
dagNode list

```
>>> objList =  dag.findComponentChildren(self.parent,
                                         oldName,
                                         oldSideIndex)
```

mgear.core.dag.**findComponentChildren3**(*node*, *name*, *sideIndex*)

Returns the component children of input component root.

This function is using Maya cmds instead of PyMel

Note: This method is specific to work with shifter guides naming conventions

Parameters

- **node** (*dagNode*) – The input node to search
- **name** (*str*) – The name to search
- **sideIndex** (*str*) – the side

Returns
The children dagNodes

Return type
dagNode list

```
>>> objList =  dag.findComponentChildren(self.parent,
                                         oldName,
                                         oldSideIndex)
```

mgear.core.dag.**getShapes**(*node*)

Returns the shape of the dagNode

Parameters
node (*dagNode*) – The input node to search the shape

Returns
The shapes of the node

Return type
list

mgear.core.dag.**getTopParent**(*node*)

Returns the first parent of the hierarchy.

usually the 'Model' in Softimage terminology

Parameters
`node` (*dagNode*) – The input node to search.

Returns
The top parent of the input node

Return type
dagNode

mgear.core.dagmenu module

mgear.core.dragdrop module

mgear.core.fcurve module

`mgear.core.fcurve.`**`getFCurveValues`**(*fcv_node*, *division*, *factor=1*)

Get X values evenly spaced on the FCurve.

Parameters

- **`fcv_node`** (*pyNode or str*) – The FCurve to evaluate.
- **`division`** (*int*) – The number of division you want to evaluate on the FCurve.
- **`factor`** (*float*) – Multiplication factor. Default = 1. (optional)

Returns
The values in a list float.

Return type
list of float

```
>>> self.st_value = fcu.getFCurveValues(self.settings["st_profile"],
                                        self.divisions)
```

mgear.core.icon module

Predefined nurbsCurve shapes to be use as a rigging control Icons

`mgear.core.icon.`**`arrow`**(*parent=None*, *name='arrow'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a ARROW shape.

Parameters

- **`parent`** (*dagNode*) – The parent object of the newly created curve.
- **`name`** (*str*) – Name of the curve.
- **`width`** (*float*) – Width of the shape.
- **`color`** (*int or list of float*) – The color in index base or RGB.
- **`m`** (*matrix*) – The global transformation of the curve.
- **`pos_offset`** (*vector*) – The xyz position offset of the curve from its center.
- **`rot_offset`** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.axis`(*parent=None*, *name='axis'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a AXIS shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.circle`(*parent=None*, *name='circle'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*, *degree=3*)

Create a curve with a CIRCLE shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.compas`(*parent=None*, *name='compas'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*, *degree=3*)

Create a curve with a COMPAS shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.

- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.`**`connection_display_curve`**(*name*, *centers=[]*, *degree=1*)

Visual reference curves connectiong points.

Display curve object is a simple curve to show the connection between different guide element..

Parameters

- **name** (*str*) – Local name of the element.
- **centers** (*list of dagNode*) – List of object to define the curve.
- **degree** (*int*) – Curve degree. Default 1 = lineal.

Returns
The newly creted curve.

Return type
dagNode

`mgear.core.icon.`**`create`**(*parent=None*, *name='icon'*, *m=pymel.core.datatypes.Matrix*, *color=[0, 0, 0]*, *icon='cube'*, ***kwargs*)

Icon master function

Create icon master function. This function centralize all the icons creation

Parameters

- **parent** (*dagNode*) – The parent for the new icon
- **name** (*str*) – Name of the Icon.
- **m** (*matrix*) – Transformation matrix of the icon
- **color** (*int or list of float*) – The color in index base or RGB.
- **icon** (*str*) – Icon type. Options: “cube”, “pyramid”, “square”, “flower”, “circle”, “cylinder”, “compas”, “diamond”,

  ”cubewithpeak”, “sphere”, “arrow”, “crossarrow”, “cross”, “null”
- **kwargs** – The keyword arguments can vary depending of the icon type. Please refear to the specific icon method for more info.

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.cross`(*parent=None*, *name='cross'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a CROSS shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.crossarrow`(*parent=None*, *name='crossArrow'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a CROSS ARROW shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.cube`(*parent=None*, *name='cube'*, *width=1*, *height=1*, *depth=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a CUBE shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **height** (*float*) – Height of the shape.
- **depth** (*float*) – Depth of the shape.

- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.`**cubewithpeak**(*parent=None*, *name='cubewithpeak'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a CUBE WITH PEAK shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.`**cylinder**(*parent=None*, *name='cylinder'*, *width=1*, *heigth=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*, *degree=3*)

Create a curve with a CYLINDER shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **height** (*float*) – Height of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

mgear.core.icon.**diamond**(*parent=None*, *name='diamond'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a DIAMOND shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **height** (*float*) – Height of the shape.
- **depth** (*float*) – Depth of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

mgear.core.icon.**flower**(*parent=None*, *name='flower'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*, *degree=3*)

Create a curve with a FLOWER shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

mgear.core.icon.**getPointArrayWithOffset**(*point_pos*, *pos_offset=None*, *rot_offset=None*)

Get Point array with offset

Convert a list of vector to a List of float and add the position and rotation offset.

Parameters

- **point_pos** (*list of vector*) – Point positions.
- **pos_offset** (*vector*) – The position offset of the curve from its center.
- **rot_offset** (*vector*) – The rotation offset of the curve from its center. In radians.

Returns
the new point positions

Return type
list of vector

`mgear.core.icon.`**`guideBladeIcon`**(*parent=None*, *name='blade'*, *lenX=1.0*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a BLADE GUIDE shape.

Note: This icon is specially design for **Shifter** blade guides

Parameters

- **`parent`** (*`dagNode`*) – The parent object of the newly created curve.
- **`name`** (*`str`*) – Name of the curve.
- **`lenX`** (*`float`*) – Width of the shape.
- **`color`** (*`int or list of float`*) – The color in index base or RGB.
- **`m`** (*`matrix`*) – The global transformation of the curve.
- **`pos_offset`** (*`vector`*) – The xyz position offset of the curve from its center.
- **`rot_offset`** (*`vector`*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.`**`guideLocatorIcon`**(*parent=None*, *name='locator'*, *width=0.5*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a LOCATOR GUIDE shape.

Note: This icon is specially design for **Shifter** locator guides

Parameters

- **`parent`** (*`dagNode`*) – The parent object of the newly created curve.
- **`name`** (*`str`*) – Name of the curve.
- **`width`** (*`float`*) – Width of the shape.
- **`color`** (*`int or list of float`*) – The color in index base or RGB.
- **`m`** (*`matrix`*) – The global transformation of the curve.
- **`pos_offset`** (*`vector`*) – The xyz position offset of the curve from its center.
- **`rot_offset`** (*`vector`*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.guideRootIcon(parent=None, name='root', width=0.5, color=[0, 0, 0], m=pymel.core.datatypes.Matrix, pos_offset=None, rot_offset=None)`

Create a curve with a ROOT GUIDE shape.

Note: This icon is specially design for **Shifter** root guides

Parameters

- `parent` (*dagNode*) – The parent object of the newly created curve.
- `name` (*str*) – Name of the curve.
- `width` (*float*) – Width of the shape.
- `color` (*int or list of float*) – The color in index base or RGB.
- `m` (*matrix*) – The global transformation of the curve.
- `pos_offset` (*vector*) – The xyz position offset of the curve from its center.
- `rot_offset` (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.guideRootIcon2D(parent=None, name='root', width=0.5, color=[0, 0, 0], m=pymel.core.datatypes.Matrix, pos_offset=None, rot_offset=None)`

Create a curve with a 2D ROOT GUIDE shape.

Note: This icon is specially design for **Shifter** root guides

Parameters

- `parent` (*dagNode*) – The parent object of the newly created curve.
- `name` (*str*) – Name of the curve.
- `width` (*float*) – Width of the shape.
- `color` (*int or list of float*) – The color in index base or RGB.
- `m` (*matrix*) – The global transformation of the curve.
- `pos_offset` (*vector*) – The xyz position offset of the curve from its center.
- `rot_offset` (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.null`(*parent=None*, *name='null'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a NULL shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.pyramid`(*parent=None*, *name='pyramid'*, *width=1*, *height=1*, *depth=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a PYRAMIDE shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **height** (*float*) – Height of the shape.
- **depth** (*float*) – Depth of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.setcolor`(*node*, *color*)

Set the color in the Icons.

Parameters

- **node** (*dagNode*) – The object
- **color** (*int or list of float*) – The color in index base or RGB.

`mgear.core.icon.sphere`(*parent=None*, *name='sphere'*, *width=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*, *degree=3*)

Create a curve with a SPHERE shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

`mgear.core.icon.square`(*parent=None*, *name='square'*, *width=1*, *depth=1*, *color=[0, 0, 0]*, *m=pymel.core.datatypes.Matrix*, *pos_offset=None*, *rot_offset=None*)

Create a curve with a SQUARE shape.

Parameters

- **parent** (*dagNode*) – The parent object of the newly created curve.
- **name** (*str*) – Name of the curve.
- **width** (*float*) – Width of the shape.
- **depth** (*float*) – Depth of the shape.
- **color** (*int or list of float*) – The color in index base or RGB.
- **m** (*matrix*) – The global transformation of the curve.
- **pos_offset** (*vector*) – The xyz position offset of the curve from its center.
- **rot_offset** (*vector*) – The xyz rotation offset of the curve from its center. xyz in radians

Returns
The newly created icon.

Return type
dagNode

mgear.core.log module

Logging Maya data

`mgear.core.log.`**`matrix4`**(*m*, *msg='matrix4'*)

Print matrix 4x4 data.

Parameters

- **m** (*matrix*) – 4x4 Matrix
- **msg** (*str*) – Message in front of the data print.

mgear.core.menu module

mgear.core.meshNavigation module

Functions to help navigate the mesh topology

`mgear.core.meshNavigation.`**`bBoxData`**(*obj=None*, *yZero=False*, **args*)

Get bounding box data of a mesh object

Parameters

- **obj** (*dagNode*) – Mesh object
- **yZero** (*bool*) – If True, sets the Y axis value to 0 in world space
- **args** –

Returns

center, radio, bounding box full data

Return type

list

`mgear.core.meshNavigation.`**`bboxCenter`**(*obj*, *radius=False*)

Get bounding box center of mesh object

Parameters

- **obj** (*dagNode*) – mesh object
- **radius** (*bool*) – If True return a list the center + the radius

Returns

the bounding box center in world space

Return type

list of float

```
>>> center = mnav.bboxCenter(source, radius=False)
```

`mgear.core.meshNavigation.`**`edgeLoopBetweenVertices`**(*startPos*, *endPos*)

Computes edge loop between two vertices.

Parameters

- **startPos** (*vertex*) – Start of edge loop
- **endPos** (*vertex*) – End of edge loop

Returns
Edge loop, if one exists. Otherwise None.

`mgear.core.meshNavigation.`**`edgeRangeInLoopFromMid`**(*edgeList*, *midPos*, *endA*, *endB*)

Return a range of edges in the same loop from a mid position

Parameters

- **edgeList** (*list*) – selection edge loop
- **midPos** (*vertex*) – mid vertex
- **endA** (*vertex*) – endA vertex
- **endB** (*vertex*) – endB vertex

Returns
loop range

Return type
list

`mgear.core.meshNavigation.`**`find_mirror_edge`**(*obj*, *edgeIndx*)

Return the mirror edge of an edge

Parameters

- **obj** (*PyNode or str*) – Mesh object to get the mirror edge
- **edge** (*int*) – Index of the edge to find the mirror

Returns
Mirror edge as a pynode

Return type
PyNode

`mgear.core.meshNavigation.`**`getClosestPolygonFromTransform`**(*geo*, *loc*)

Get closest polygon from transform

Parameters

- **geo** (*dagNode*) – Mesh object
- **loc** (*matrix*) – location transform

Returns
Closest Polygon

`mgear.core.meshNavigation.`**`getClosestVertexFromTransform`**(*geo*, *loc*)

Get closest vertex from transform

Parameters

- **geo** (*dagNode or str*) – Mesh object
- **loc** (*matrix*) – location transform

Returns
Closest Vertex

```
>>> v = mn.getClosestVertexFromTransform(geometry, joint)
```

mgear.core.meshNavigation.**getConcentricVertexLoop**(*loop*, *nbLoops*)

Get concentric vertex loops

Parameters

- **loop** (*list*) – Vertex loop list
- **nbLoops** (*int*) – Number of loops to search

Returns
the loop list

Return type
list

mgear.core.meshNavigation.**getExtremeVertexFromLoop**(*edgeList=None*, *sideRange=False*, *z_up=False*)

Get extreme vertex X and Y

min and max positions from edge loop

Parameters

- **edgeList** (*list*) – Edge list
- **sideRange** (*bool*) – If True will calculate the extreme position of Z instead of X

Returns
upPos, lowPos, inPos, outPos, edgeList, vertexList

Return type
list

mgear.core.meshNavigation.**getVertexRowsFromLoops**(*loopList*)

Get vertex rows from edge loops

Parameters
loopList (*list*) – Edge loop list

Returns
vertex rows

Return type
list

mgear.core.meshNavigation.**get_closes_edge_index**(*sourceGeo*, *targetGeo*, *edgeIndx*)

Get the closes edge index from 2 diferent object.

In some situation even with same topology and vertez index order. The edge index may change.

Parameters

- **sourceGeo** (*str*) – Name of the source object
- **targetGeo** (*str*) – Name of the target object
- **edgeIndx** (*int*) – Edge Index

Returns
Description

Return type
PyNode

`mgear.core.meshNavigation.get_edge_center(mesh_dag_path, edge_indices)`

Summary

Parameters

- **mesh_dag_path** (*dag paht*) – mesh dag path
- **edge_indices** (*TYPE*) – edge index tuple

Returns
position

Return type
vector

Usage example:
mesh_dag_path = get_selected_mesh() edge_indices = [40, 60] # Replace with your desired edge indices center_position = get_edge_center(mesh_dag_path, edge_indices)

`mgear.core.meshNavigation.get_mesh_dag_path(node)`

Get the mesh dag path from a given node.

Parameters
node (*str or PyNode*) – The name of the node or the PyNode itself

Returns
The dag path to the mesh

Return type
mesh_dag_path

Raises
ValueError – If the node does not exist or is not a mesh

`mgear.core.meshNavigation.get_selected_mesh()`

Get the selected mesh dag path.

Returns
selected mesh dag path

Return type
seleted mesh

Raises
ValueError – Non selected object

mgear.core.node module

Functions to create and connect nodes.

`mgear.core.node.add_controller_tag(ctl, tagParent=None)`

Add a controller tag

Parameters

- **ctl** (*dagNode*) – Controller to add the tar
- **tagParent** (*dagNode*) – tag parent for the connection

mgear.core.node.**controller_tag_connect**(*ctt*, *tagParent*)

Summary

Parameters

- **ctt** (*TYPE*) – Teh control tag
- **tagParent** (*TYPE*) – The object with the parent control tag

mgear.core.node.**createAddNode**(*inputA*, *inputB*)

Create and connect a addition node.

Parameters

- **inputA** (*attr or float*) – The attribute input A
- **inputB** (*attr or float*) – The attribute input B

Returns
the newly created node.

Return type
pyNode

```
>>> add_node = nod.createAddNode(self.roundness_att, .001)
```

mgear.core.node.**createAddNodeMulti**(*inputs=[]*)

Create and connect multiple add nodes

Parameters
inputs (*list of attr*) – The list of attributes to add

Returns
The output attributes list.

Return type
list

```
>>> angle_outputs = nod.createAddNodeMulti(self.angles_att)
```

mgear.core.node.**createBlendNode**(*inputA*, *inputB*, *blender=0.5*)

Create and connect a createBlendNode node.

Parameters

- **inputA** (*attr or list of 3 attr*) – The attribute input A
- **inputB** (*attr or list of 3 attr*) – The attribute input B
- **blender** (*float or attr*) – Float in 0 to 1 range or attribute string name.

Returns
the newly created node.

Return type
pyNode

```
>>> blend_node = nod.createBlendNode(
        [dm_node+".outputRotate%s"%s for s in "XYZ"],
        [cns+".rotate%s"%s for s in "XYZ"],
        self.lock_ori_att)
```

`mgear.core.node.`**`createClampNode`**`(`*input*, *in_min*, *in_max*`)`

Create and connect a clamp node

Parameters

- **input** (*attr, float or list*) – The input value to clamp
- **in_min** (*float*) – The minimun value to clamp
- **in_max** (*float*) – The maximun value to clamp

Returns
the newly created node.

Return type
pyNode

```
>>> clamp_node = nod.createClampNode(
    [self.roll_att, self.bank_att, self.bank_att],
    [0, -180, 0],
    [180,0,180])
```

`mgear.core.node.`**`createClampNodeMulti`**`(`*name*, *inputs=[]*, *in_min=[]*, *in_max=[]*`)`

Create and connect multiple clamp nodes

Parameters

- **name** (*str*) – The name for the new node.
- **inputs** (*list of attr*) – The list of attributes
- **in_min** (*list of attr*) – The list of attributes
- **in_max** (*list of attr*) – The list of attributes

Returns
The output attributes list.

Return type
list

`mgear.core.node.`**`createConditionNode`**`(`*firstTerm=False*, *secondTerm=False*, *operator=0*, *ifTrue=False*, *ifFalse=False*`)`

Create and connect a condition node.

operator	index
==	0
!=	1
>	2
>=	3
<	4
<=	5

Parameters

- **firstTerm** (*attr*) – The attribute string name for the first conditions.
- **secondTerm** (*attr*) – The attribute string for the second conditions.
- **operator** (*int*) – The operator to make the condition.

- **ifTrue** (*bool or attr*) – If an attribute is provided will connect ifTrue output.
- **ifFalse** (*bool or attr*) – If an attribute is provided will connect ifFalse output.

Returns
the newly created node.

Return type
pyNode

```
>>> cond1_node = nod.createConditionNode(self.soft_attr,
                                         0,
                                         2,
                                         subtract3_node+".output1D",
                                         plusTotalLength_node+".output1D")
```

mgear.core.node.**createCurveInfoNode**(*crv*)

Create and connect a curveInfo node.

Parameters
crv (*dagNode*) – The curve.

Returns
the newly created node.

Return type
pyNode

```
>>> crv_node = nod.createCurveInfoNode(self.slv_crv)
```

mgear.core.node.**createDecomposeMatrixNode**(*m*)

Create and connect a decomposeMatrix node.

Parameters
m (*str or attr*) – The matrix attribute name.

Returns
the newly created node.

Return type
pyNode

```
>>> dm_node = nod.createDecomposeMatrixNode(mulmat_node+".output")
```

mgear.core.node.**createDistNode**(*objA*, *objB*, *output=None*)

Create and connect a distance node.

Parameters

- **objA** (*dagNode*) – The dagNode A.
- **objB** (*dagNode*) – The dagNode B.
- **output** (*attr*) – Output attribute.

Returns
the newly created node.

Return type
pyNode

```
>>> distA_node = nod.createDistNode(self.tws0_loc, self.tws1_loc)
```

`mgear.core.node.`**`createDivNode`**`(`*inputA*, *inputB*, *output=None*`)`

Create and connect a Divide node.

Parameters

- **inputA** (*attr, float or list of float*) – The attribute input A
- **inputB** (*attr, float or list of float*) – The attribute input B
- **output** (*attr or list of attr*) – The attribute to connect the output.

Returns
the newly created node.

Return type
pyNode

Example

```
# Classic Maya style creation and connection = 4 lines
div1_node = pm.createNode("multiplyDivide")
div1_node.setAttr("operation", 2)
div1_node.setAttr("input1X", 1)
pm.connectAttr(self.rig.global_ctl+".sx",
               div1_node+".input2X")

# mGear style = 1 line
div1_node = nod.createDivNode(1.0,
                              self.rig.global_ctl+".sx")
```

`mgear.core.node.`**`createDivNodeMulti`**`(`*name*, *inputs1=[]*, *inputs2=[]*`)`

Create and connect multiple divide nodes

Parameters

- **name** (*str*) – The name for the new node.
- **inputs1** (*list of attr*) – The list of attributes
- **inputs2** (*list of attr*) – The list of attributes

Returns
The output attributes list.

Return type
list

`mgear.core.node.`**`createMulDivNode`**`(`*inputA*, *inputB*, *operation=1*, *output=None*`)`

Create and connect a Multiply or Divide node.

Parameters

- **inputA** (*attr, float or list of float*) – The attribute input A
- **inputB** (*attr, float or list of float*) – The attribute input B
- **output** (*attr or list of attr*) – The attribute to connect the output.

Returns
the newly created node.

Return type
pyNode

`mgear.core.node.`**`createMulNode`**`(`*inputA*, *inputB*, *output=None*`)`

Create and connect a Multiply node.

Parameters

- **inputA** (*attr, float or list of float*) – The attribute input A
- **inputB** (*attr, float or list of float*) – The attribute input B
- **output** (*attr or list of attr*) – The attribute to connect the output.

Returns
the newly created node.

Return type
pyNode

`mgear.core.node.`**`createMulNodeMulti`**`(`*name*, *inputs=[]*`)`

Create and connect multiple multiply nodes

Parameters

- **name** (*str*) – The name for the new node.
- **inputs** (*list of attr*) – The list of attributes to multiply

Returns
The output attributes list.

Return type
list

`mgear.core.node.`**`createMultMatrixNode`**`(`*mA*, *mB*, *target=False*, *transform='srt'*`)`

Create Maya multiply Matrix node.

Note: This node have same functionality as the default Maya matrix multiplication.

Parameters

- **mA** (*matrix*) – input matrix A.
- **mB** (*matrix*) – input matrix B.
- **target** (*dagNode*) – object target to apply the transformation
- **transform** (*str*) – if target is True. out transform to SRT valid value s r t

Returns
Newly created mGear_multMatrix node

Return type
pyNode

`mgear.core.node.`**`createNegateNodeMulti`**`(`*name*, *inputs=[]*`)`

Create and connect multiple negate nodes

Parameters

- **name** (*str*) – The name for the new node.
- **inputs** (*list of attr*) – The list of attributes to negate

Returns
The output attributes list.

Return type
list

`mgear.core.node.`**`createPairBlend`**`(`*inputA=None*, *inputB=None*, *blender=0.5*, *rotInterpolation=0*, *output=None*, *trans=True*, *rot=True*`)`

Create and connect a PairBlend node.

Parameters

- **inputA** (*dagNode*) – The transfomr input 1
- **inputB** (*dagNode*) – The transfomr input 2
- **blender** (*float or attr*) – Float in 0 to 1 range or attribute string name.
- **rotInterpolation** (*int*) – Rotation interpolation option. 0=Euler. 1=Quaternion.
- **output** (*dagNode*) – The output node with the blend transfomr applied.
- **trans** (*bool*) – If true connects translation.
- **rot** (*bool*) – If true connects rotation.

Returns
the newly created node.

Return type
pyNode

Example

```
blend_node = nod.createPairBlend(self.legBonesFK[i],
                                 self.legBonesIK[i],
                                 self.blend_att,
                                 1)
pm.connectAttr(blend_node + ".outRotate", x+".rotate")
pm.connectAttr(blend_node + ".outTranslate", x+".translate")
```

`mgear.core.node.`**`createPickMatrix`**`(`*m=None*, *out_m=None*, *scale=True*, *rotate=True*, *translate=True*, *shear=True*`)`

Summary

Parameters

- **m** (*None, optional*) – input matrix
- **out_m** (*None, optional*) – output matrix attr to connect
- **scale** (*bool, optional*) – use scale
- **rotate** (*bool, optional*) – use rotate

- **translate** (*bool, optional*) – use translate
- **shear** (*bool, optional*) – use shear

`mgear.core.node.`**`createPlusMinusAverage1D`**(*input*, *operation=1*, *output=None*)

Create a multiple average node 1D. :param input: The input values. :type input: attr, float or list :param operation: Node operation. 0=None, 1=sum, 2=subtract,

3=average

Parameters
output (*attr*) – The attribute to connect the result.

Returns
the newly created node.

Return type
pyNode

`mgear.core.node.`**`createPowNode`**(*inputA*, *inputB*, *output=None*)

Create and connect a power node.

Parameters

- **inputA** (*attr, float or list of float*) – The attribute input A
- **inputB** (*attr, float or list of float*) – The attribute input B
- **output** (*attr or list of attr*) – The attribute to connect the output.

Returns
the newly created node.

Return type
pyNode

`mgear.core.node.`**`createReverseNode`**(*input*, *output=None*)

Create and connect a reverse node.

Parameters

- **input** (*attr or list of 3 attr*) – The attribute input.
- **output** (*attr or list of 3 attr*) – The attribute to connect the output.

Returns
the newly created node.

Return type
pyNode

```
>>> fkvis_node = nod.createReverseNode(self.blend_att)
```

`mgear.core.node.`**`createSetRangeNode`**(*input*, *oldMin*, *oldMax*, *newMin=0*, *newMax=1*, *output=None*, *name='setRange'*)

Create Set Range Node

`mgear.core.node.`**`createSubNode`**(*inputA*, *inputB*)

Create and connect a subtraction node.

Parameters

- **inputA** (*attr or float*) – The attribute input A

- **inputB** (*attr or float*) – The attribute input B

Returns
the newly created node.

Return type
pyNode

```
>>> sub_nod = nod.createSubNode(self.roll_att, angle_outputs[i-1])
```

mgear.core.node.**createVertexPositionNode**(*inShape*, *vId=0*, *output=None*, *name='mgear_vertexPosition'*)
Creates a mgear_vertexPosition node

mgear.core.pickWalk module

Custom Pick walk

mgear.core.pickWalk.**cleanOrphaneControllerTags**(*tag*)
Security check, delete tags without controlObject plug

Parameters
tag (*controllers tag list*) – The tags to check

Returns
The valid tags with controller object plugged

Return type
list

mgear.core.pickWalk.**controllerWalkDown**(*node*, *add=False*, *multi=False*)
Walk down in the hierachy using the controller tag

Parameters

- **node** (*dagNode or list of dagNode*) – Node with controller tag
- **add** (*bool, optional*) – If true add to selection

mgear.core.pickWalk.**controllerWalkLeft**(*node*, *add=False*, *multi=False*)
Pick walks the next sibling to the left using controller tag

Parameters

- **node** (*TYPE*) – Description
- **add** (*bool, optional*) – If true add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

mgear.core.pickWalk.**controllerWalkRight**(*node*, *add=False*, *multi=False*)
Pick walks the next sibling to the right using controller tag

Parameters

- **node** (*TYPE*) – Description
- **add** (*bool, optional*) – If true add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

`mgear.core.pickWalk.controllerWalkUp(node, add=False)`

Walk up in the hierachy using the controller tag

Parameters

- **node** (*dagNode or list of dagNode*) – Node with controller tag
- **add** (*bool, optional*) – If true add to selection

`mgear.core.pickWalk.getMirror(node)`

Get the mirrored node usin _L and _R replacement

Parameters
node (*dagNode or list of dagNodes*) – The dagNode to look for a mirror

Returns

The dagNode contrapart on the other
side _L or _R

Return type
dagNode or list of dagNodes

`mgear.core.pickWalk.getWalkTag(node)`

Get Controller tag

Parameters
node (*dagNode*) – Controller object with tag

Returns
Controller tag

Return type
tag

`mgear.core.pickWalk.get_all_tag_children(node)`

Gets all child tag controls from the given tag node

Parameters
node (*str*) – Name of controller object with tag

Returns
List of child controls (Maya transform nodes)

Return type
list

`mgear.core.pickWalk.reorderControllerChildrenTags(tag)`

Clean the order on the children connection.

This is important for the Left and right pick walk. Becasue is using the index of the connection.

Parameters
tag (*controller tag*) – The tag to clean the children order

`mgear.core.pickWalk.transformWalkDown(node, add=False, multi=False)`

Walks to the child transform dagNode on the hierarcy

Parameters

- **node** (*dagNode or list of dagNode*) – dagNode to walk
- **add** (*bool, optional*) – if True, will add to the selection
- **multi** (*bool, optional*) – if True will select all the childrens

`mgear.core.pickWalk.`**`transformWalkLeft`**`(`*node*, *add=False*, *multi=False*`)`

Pick walks to the left the next sibling transform on the hierarchy

Parameters

- **node** (*dagNode or list of dagNode*) – dagNode transform to navegate the hierarchy
- **add** (*bool, optional*) – If true add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

`mgear.core.pickWalk.`**`transformWalkRight`**`(`*node*, *add=False*, *multi=False*`)`

Pick walks to the right the next sibling transform on the hierarchy

Parameters

- **node** (*dagNode or list of dagNode*) – dagNode transform to navegate the hierarchy
- **add** (*bool, optional*) – If true add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

`mgear.core.pickWalk.`**`transformWalkUp`**`(`*node*, *add=False*`)`

Walks to the parent transform dagNode on the hierarcy

Parameters

- **node** (*dagNode or list of dagNode*) – dagNode to walk
- **add** (*bool, optional*) – if True, will add to the selection

`mgear.core.pickWalk.`**`walkDown`**`(`*node*, *add=False*, *multi=False*`)`

Walk Down

Parameters

- **node** (*dagNode or list of dagNode*) – the starting object for the pickwalk
- **add** (*bool, optional*) – If True add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

`mgear.core.pickWalk.`**`walkLeft`**`(`*node*, *add=False*, *multi=False*`)`

Walk left

Parameters

- **node** (*dagNode or list of dagNode*) – the starting object for the pickwalk
- **add** (*bool, optional*) – If True add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

`mgear.core.pickWalk.`**`walkMirror`**`(`*node*, *add=False*`)`

Select the mirror dagNode

Parameters

- **node** (*dagNode or list of dagNode*) – The dagNode to look for a mirror
- **add** (*bool, optional*) – If true add to selection

`mgear.core.pickWalk.`**`walkRight`**`(`*node*, *add=False*, *multi=False*`)`

Walk right

Parameters

- **node** (*dagNode or list of dagNode*) – the starting object for the pickwalk
- **add** (*bool, optional*) – If True add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

mgear.core.pickWalk.**walkUp**(*node*, *add=False*, *multi=False*)

Walk up

Parameters

- **node** (*dagNode or list of dagNode*) – the starting object for the pickwalk
- **add** (*bool, optional*) – If True add to selection
- **multi** (*bool, optional*) – If true, selects all the siblings

mgear.core.primitive module

Functions to create primitives (Non geometry)

mgear.core.primitive.**add2DChain**(*parent*, *name*, *positions*, *normal*, *negate=False*, *vis=True*, *axis='xz'*)

Create a 2D joint chain. Like Softimage 2D chain.

Warning: This function will create un expected results if all the positions are not in the same 2D plane.

Parameters

- **parent** (*dagNode*) – The parent for the chain.
- **name** (*str*) – The node name.
- **positions** (*list of vectors*) – the positons to define the chain.
- **normal** (*vector*) – The normal vector to define the direction of the chain.
- **negate** (*bool*) – If True will negate the direction of the chain

Returns;

list of dagNodes: The list containg all the joints of the chain

```
>>> self.rollRef = pri.add2DChain(
    self.root,
    self.getName("rollChain"),
    self.guide.apos[:2],
    self.normal,
    self.negate)
```

mgear.core.primitive.**add2DChain2**(*parent*, *name*, *positions*, *normal*, *negate=False*, *vis=True*)

Experimental 2D Chain creation function.

Create a 2D joint chain. Like Softimage 2D chain.

Warning: This function is WIP and not ready for production.

Warning: This function will create un expected results if all the positions are not in the same 2D plane.

Parameters

- **parent** (*dagNode*) – The parent for the chain.
- **name** (*str*) – The node name.
- **positions** (*list of vectors*) – the positons to define the chain.
- **normal** (*vector*) – The normal vector to define the direction of the chain.
- **negate** (*bool*) – If True will negate the direction of the chain

Returns;
list of dagNodes: The list containg all the joints of the chain

```
>>> self.chain3bones = pri.add2DChain2(
    self.setup,
    self.getName("chain3bones%s_jnt"),
    self.guide.apos[0:4],
    self.normal,
    False)
```

mgear.core.primitive.**addIkHandle**(*parent*, *name*, *chn*, *solver='ikRPsolver'*, *poleV=None*)

Creates and connect an IKhandle to a joints chain.

Parameters

- **parent** (*dagNode*) – The parent for the IKhandle.
- **name** (*str*) – The node name.
- **chn** (*list*) – List of joints.
- **solver** (*str*) – the solver to be use for the ikHandel. Default value is "ikRPsolver"
- **poleV** (*dagNode*) – Pole vector for the IKHandle

Returns
The IKHandle

Return type
dagNode

```
>>> self.ikHandleUpvRef = pri.addIkHandle(
    self.root,
    self.getName("ikHandleLegChainUpvRef"),
    self.legChainUpvRef,
    "ikSCsolver")
```

mgear.core.primitive.**addJoint**(*parent*, *name*, *m=pymel.core.datatypes.Matrix*, *vis=True*)

Create a joint dagNode.

Note: I'm not using the joint() comand because this is parenting the newly created joint to current selection which might not be desired

Parameters

- **parent** (*dagNode*) – The parent for the node.
- **name** (*str*) – The node name.
- **m** (*matrix*) – The matrix for the node transformation (optional).
- **vis** (*bool*) – Set the visibility of the new joint.

Returns
The newly created node.

Return type
dagNode

`mgear.core.primitive.`**`addJointFromPos`**(*parent*, *name*, *pos=pymel.core.datatypes.Vector*)

Create a joint dagNode.

Note: I'm not using the joint() comand because this is parenting the newly created joint to current selection which might not be desired

Parameters

- **parent** (*dagNode*) – The parent for the node.
- **name** (*str*) – The node name.
- **pos** (*vector*) – The vector for the node position (optional).
- **vis** (*bool*) – Set the visibility of the new joint.

Returns
The newly created node.

Return type
dagNode

`mgear.core.primitive.`**`addLocator`**(*parent*, *name*, *m=pymel.core.datatypes.Matrix*, *size=1*)

Create a space locator dagNode.

Parameters

- **parent** (*dagNode*) – The parent for the node.
- **name** (*str*) – The Node name.
- **m** (*matrix*) – The matrix for the node transformation (optional).
- **size** (*float*) – The space locator shape size (optional).

Returns
The newly created node.

Return type
dagNode

`mgear.core.primitive.`**`addLocatorFromPos`**(*parent*, *name*, *pos=pymel.core.datatypes.Vector*, *size=1*)

Create a space locator dagNode.

Parameters

- **parent** (*dagNode*) – The parent for the node.

- **name** (*str*) – The Node name.
- **pos** (*vector*) – The vector for the node position (optional).
- **size** (*float*) – The space locator shape size (optional).

Returns
The newly created node.

Return type
dagNode

mgear.core.primitive.**addTransform**(*parent*, *name*, *m=pymel.core.datatypes.Matrix*)

Create a transform dagNode.

Parameters

- **parent** (*dagNode*) – The parent for the node.
- **name** (*str*) – The Node name.
- **m** (*matrix*) – The matrix for the node transformation (optional).

Returns
The newly created node.

Return type
dagNode

mgear.core.primitive.**addTransformFromPos**(*parent*, *name*, *pos=pymel.core.datatypes.Vector*)

Create a transform dagNode.

Parameters

- **parent** (*dagNode*) – The parent for the node.
- **name** (*str*) – The Node name.
- **pos** (*vector*) – The vector for the node position (optional).

Returns
The newly created node.

Return type
dagNode

mgear.core.pyFBX module

mgear.core.pyFBX.**FBXClose**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExport**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportAnimationOnly**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportApplyConstantKeyReducer**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportAudio**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportAxisConversionMethod**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportBakeComplexAnimation**(**args*, ***kwargs*) mgear.core.pyFBX.**FBXExportBakeComplexEnd**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportBakeComplexStart**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportBakeComplexStep**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportBakeResampleAnimation**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportCacheFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportCameras**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportColladaFrameRate**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportColladaSingleMatrix**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportColladaTriangulate**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportConstraints**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportConvert2Tif**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportConvertUnitString**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportDeleteOriginalTakeOnSplitAnimation**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportEmbeddedTextures**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportFileVersion**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportFinestSubdivLevel**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportGenerateLog**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportHardEdges**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportInAscii**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportIncludeChildren**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportInputConnections**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportInstances**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportLights**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportQuaternion**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportQuickSelectSetAsCache**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportReferencedAssetsContent**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportReferencedContainersContent**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportScaleFactor**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportShapeAttributeValues**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportShapeAttributes**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportShapes**(**args*, ***kwargs*) mgear.core.pyFBX.**FBXExportShowUI**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportSkeletonDefinitions**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportSkins**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportSmoothMesh**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportSmoothingGroups**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportSplitAnimationIntoTakes**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportTangents**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportTriangulate**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportUpAxis**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportUseSceneName**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXExportUseTmpFilePeripheral**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeComment**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeCount**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeIndex**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeLocalTimeSpan**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeName**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXGetTakeReferenceTimeSpan**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImport**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportAudio**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportAutoAxisEnable**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportAxisConversionEnable**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportCacheFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportCameras**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportConstraints**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportConvertDeformingNullsToJoint**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportConvertUnitString**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportFillTimeline**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportForcedFileAxis**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportGenerateLog**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportHardEdges**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportLights**(**args*, ***kwargs*) mgear.core.pyFBX.**FBXImportMergeAnimationLayers**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportMergeBackNullPivots**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportMode**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportOCMerge**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportProtectDrivenKeys**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportQuaternion**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportResamplingRateSource**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportScaleFactor**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportSetLockedAttribute**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportSetMayaFrameRate**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportSetTake**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportShapes**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportShowUI**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportSkeletonDefinitionsAs**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportSkins**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportUnlockNormals**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXImportUpAxis**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXLoadExportPresetFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXLoadImportPresetFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXLoadMBExportPresetFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXLoadMBImportPresetFile**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXPopSettings**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXProperties**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXProperty**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXPushSettings**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXRead**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXResamplingRate**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXResetExport**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXResetImport**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXUICallBack**(**args*, ***kwargs*)

mgear.core.pyFBX.**FBXUIShowOptions**(**args*, ***kwargs*) mgear.core.pyFBX.**get_fbx_export_presets**()

Returns all available FBX export preset files

Returns
String paths of the available fbx export preset files

Return type
list

mgear.core.pyFBX.**get_fbx_import_presets**()

Returns all available FBX export preset files

Returns
String paths of the available fbx export preset files

Return type
list

mgear.core.pyFBX.**get_fbx_versions**()

Get available FBX version list

Returns
String names of the available fbx versions

Return type
list

mgear.core.pyflow_widgets module

mgear.core.pyqt module

mgear.core.six module

Utilities for writing code that runs on Python 2 and 3

class mgear.core.six.**Module_six_moves_urllib**(*name*, *doc=None*)

Bases: `module`

Create a six.moves.urllib namespace that resembles the Python 3 namespace

error = <module 'mgear.core.six.moves.urllib.error'>

parse = <module 'mgear.core.six.moves.urllib_parse'>

request = <module 'mgear.core.six.moves.urllib.request'>

response = <module 'mgear.core.six.moves.urllib.response'>

robotparser = <module 'mgear.core.six.moves.urllib.robotparser'>

class mgear.core.six.**Module_six_moves_urllib_error**(*name*)

Bases: `_LazyModule`

Lazy loading of moved objects in six.moves.urllib_error

ContentTooShortError

HTTPError

URLError

class mgear.core.six.**Module_six_moves_urllib_parse**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_parse

ParseResult

SplitResult

parse_qs

parse_qsl

quote

quote_plus

splitquery

splittag

splituser

splitvalue

unquote

unquote_plus

unquote_to_bytes

urldefrag

urlencode

urljoin

urlparse

urlsplit

urlunparse

urlunsplit

uses_fragment

uses_netloc

uses_params

uses_query

uses_relative

class mgear.core.six.**Module_six_moves_urllib_request**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_request

AbstractBasicAuthHandler

AbstractDigestAuthHandler

BaseHandler

CacheFTPHandler

FTPHandler

FancyURLopener

FileHandler

HTTPBasicAuthHandler

HTTPCookieProcessor

HTTPDefaultErrorHandler

HTTPDigestAuthHandler

HTTPErrorProcessor

HTTPHandler

HTTPPasswordMgr

HTTPPasswordMgrWithDefaultRealm

HTTPRedirectHandler

HTTPSHandler

OpenerDirector

ProxyBasicAuthHandler

ProxyDigestAuthHandler

ProxyHandler

Request

URLopener

UnknownHandler

build_opener

getproxies

install_opener

parse_http_list

parse_keqv_list

pathname2url

proxy_bypass

url2pathname

urlcleanup

urlopen

urlretrieve

class mgear.core.six.**Module_six_moves_urllib_response**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_response

addbase

addclosehook

addinfo

addinfourl

class mgear.core.six.**Module_six_moves_urllib_robotparser**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_robotparser

RobotFileParser

class mgear.core.six.**MovedAttribute**(*name*, *old_mod*, *new_mod*, *old_attr=None*, *new_attr=None*)

Bases: _LazyDescr

class mgear.core.six.**MovedModule**(*name*, *old*, *new=None*)

Bases: _LazyDescr

mgear.core.six.**add_metaclass**(*metaclass*)

Class decorator for creating a class with a metaclass.

mgear.core.six.**add_move**(*move*)

Add an item to six.moves.

mgear.core.six.**assertCountEqual**(*self*, **args*, ***kwargs*)

mgear.core.six.**assertNotRegex**(*self*, **args*, ***kwargs*)

mgear.core.six.**assertRaisesRegex**(*self*, **args*, ***kwargs*)

mgear.core.six.**assertRegex**(*self*, **args*, ***kwargs*)

mgear.core.six.**b**(*s*)

Byte literal

mgear.core.six.**create_unbound_method**(*func*, *cls*)

mgear.core.six.**ensure_binary**(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce **s** to six.binary_type.

For Python 2:

- *unicode* -> encoded to *str*
- *str* -> *str*

For Python 3:

- *str* -> encoded to *bytes*
- *bytes* -> *bytes*

`mgear.core.six.`**`ensure_str`**(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce *s* to *str*.

For Python 2:

- *unicode* -> encoded to *str*
- *str* -> *str*

For Python 3:

- *str* -> *str*
- *bytes* -> decoded to *str*

`mgear.core.six.`**`ensure_text`**(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce *s* to six.text_type.

For Python 2:

- *unicode* -> *unicode*
- *str* -> *unicode*

For Python 3:

- *str* -> *str*
- *bytes* -> decoded to *str*

`mgear.core.six.`**`get_unbound_function`**(*unbound*)

Get the function out of a possibly unbound function

`mgear.core.six.`**`int2byte`**()

S.pack(v1, v2, …) -> bytes

Return a bytes object containing values v1, v2, … packed according to the format string S.format. See help(struct) for more on format strings.

`mgear.core.six.`**`iteritems`**(*d*, ***kw*)

Return an iterator over the (key, value) pairs of a dictionary.

`mgear.core.six.`**`iterkeys`**(*d*, ***kw*)

Return an iterator over the keys of a dictionary.

`mgear.core.six.`**`iterlists`**(*d*, ***kw*)

Return an iterator over the (key, [values]) pairs of a dictionary.

`mgear.core.six.`**`itervalues`**(*d*, ***kw*)

Return an iterator over the values of a dictionary.

`mgear.core.six.`**`python_2_unicode_compatible`**(*klass*)

A class decorator that defines __unicode__ and __str__ methods under Python 2. Under Python 3 it does nothing.

To support Python 2 and 3 with a single code base, define a __str__ method returning text and apply this decorator to the class.

`mgear.core.six.`**`raise_from`**(*value*, *from_value*) mgear.core.six.**remove_move**(*name*)

Remove item from six.moves.

mgear.core.six.**reraise**(*tp*, *value*, *tb=None*)

Reraise an exception.

mgear.core.six.**u**(*s*)

Text literal

mgear.core.six.**with_metaclass**(*meta*, **bases*)

Create a base class with a metaclass.

mgear.core.skin module

Functions to work with skinCluster data.

This module is derivated from Chad Vernon's Skin IO.

`Chad Vernon's github

<https://github.com/chadmv/cmt/tree/master/scripts/cmt/deform>`_

mgear.core.skin.**collectBlendWeights**(*skinCls*, *dagPath*, *components*, *dataDic*)

mgear.core.skin.**collectData**(*skinCls*, *dataDic*)

mgear.core.skin.**collectInfluenceWeights**(*skinCls*, *dagPath*, *components*, *dataDic*)

mgear.core.skin.**exportJsonSkinPack**(*packPath=None*, *objs=None*, **args*)

mgear.core.skin.**exportSkin**(*filePath=None*, *objs=None*, **args*)

mgear.core.skin.**exportSkinPack**(*packPath=None*, *objs=None*, *use_json=False*, **args*)

mgear.core.skin.**getCurrentWeights**(*skinCls*, *dagPath*, *components*)

Get the skincluster weights

Parameters

- **skinCls** (*PyNode*) – The skincluster node
- **dagPath** (*MDagPath*) – The skincluster dagpath
- **components** (*MObject*) – The skincluster components

Returns

The skincluster weights

Return type

MDoubleArray

mgear.core.skin.**getGeometryComponents**(*skinCls*)

Get the geometry components from skincluster

Parameters

skinCls (*PyNode*) – The skincluster node

Returns

The dagpath for the components componets: The skincluster componets

Return type

dagPath

`mgear.core.skin.getObjsFromSkinFile`(*filePath=None*, **args*)

`mgear.core.skin.getSkinCluster`(*obj*)

Get the skincluster of a given object

Parameters
`obj` (*dagNode*) – The object to get skincluster

Returns
The skin cluster pynode object

Return type
pyNode

`mgear.core.skin.get_mesh_components_from_tag_expression`(*skinCls*, *tag='*'*)

Get the mesh components from the component tag expression

Thanks to Roy Nieterau a.k.a BigRoyNL from colorBleed for the snippet

Parameters

- `skinCls` (*PyNode*) – Skin cluster node
- `tag` (*str, optional*) – Component tag expression

Returns
The dagpath tho the shpe and the MObject components

Return type
dagPath, MObject

`mgear.core.skin.importSkin`(*filePath=None*, **args*)

`mgear.core.skin.importSkinPack`(*filePath=None*, **args*)

`mgear.core.skin.selectDeformers`(**args*)

`mgear.core.skin.setBlendWeights`(*skinCls*, *dagPath*, *components*, *dataDic*, *compressed*)

`mgear.core.skin.setData`(*skinCls*, *dataDic*, *compressed*)

`mgear.core.skin.setInfluenceWeights`(*skinCls*, *dagPath*, *components*, *dataDic*, *compressed*)

`mgear.core.skin.skinCopy`(*sourceMesh=None*, *targetMesh=None*, **args*)

mgear.core.string module

string management methods

`mgear.core.string.convertRLName`(*name*)

Convert a string with underscore

i.e: "_L", "_L0_", "L_", "_L" to "R". And vice and versa.

Parameters
`name` (*string*) – string to convert

Returns
Tuple of Integer

`mgear.core.string.convertRLName_old(name)`

Convert a string with underscore

i.e: "_L", "_L0_", "L_", "_L" to "R". And vice and versa.

Parameters
`name` (*string*) – string to convert

Returns
Tuple of Integer

`mgear.core.string.normalize(string)`

Replace all invalid characters with "_"

Parameters
`string` (*string*) – A string to normalize.

Return string
Normalized string

`mgear.core.string.normalize2(string)`

Replace all invalid characters with "_". including "-" This ensure that the name is compatible with Maya naming rules

Parameters
`string` (*string*) – A string to normalize.

Return string
Normalized string

`mgear.core.string.normalize_path(string)`

Ensure that string path use always forward slash

Parameters
`string` (*TYPE*) – Description

Returns
Description

Return type
TYPE

`mgear.core.string.normalize_with_padding(string)`

Replace all invalid characters with "_". including "-" This ensure that the name is compatible with Maya naming rules

Also list of # symbol with properly padded index.

ie. count_### > count_001

Parameters
`string` (*string*) – A string to normalize.

Return string
Normalized string

`mgear.core.string.removeInvalidCharacter(string)`

Remove all invalid character.

Parameters
`string` (*string*) – A string to normalize.

Return string
Normalized string.

`mgear.core.string.`**`removeInvalidCharacter2`**`(`*string*`)`

Remove all invalid character. Incluede "_" and "."as valid character.

Parameters
`string` (*`string`*) – A string to normalize.

Return string
Normalized string.

`mgear.core.string.`**`replaceSharpWithPadding`**`(`*string*, *index*`)`

Replace a list of # symbol with properly padded index.

ie. count_### > count_001

Parameters

- **`string`** (*`string`*) – A string to set. Should include '#'
- **`index`** (*`integer`*) – Index to replace.

Return string
Normalized string.

mgear.core.transform module

Functions to work with matrix and transformations

`mgear.core.transform.`**`convert2TransformMatrix`**`(`*tm*`)`

Convert a transformation Matrix

Convert a transformation Matrix or a matrix to a transformation matrix in world space.

Parameters
`tm` (*`matrix`*) – The input matrix.

Returns
The transformation matrix in worldSpace

Return type
matrix

`mgear.core.transform.`**`getChainTransform`**`(`*positions*, *normal*, *negate=False*, *axis='xz'*`)`

Get a tranformation list from a positions list and normal.

Parameters

- **`positions`** (*`list of vector`*) – List with the chain positions.
- **`normal`** (*`vector`*) – Normal direction.
- **`negate`** (*`bool`*) – If true invert the chain orientation.

Returns

The list containing the transformation matrix
for the chain.

Return type
list of matrix

```
>>> tra.getChainTransform(self.guide.apos, self.normal, self.negate)
```

`mgear.core.transform.`**`getChainTransform2`**(*positions*, *normal*, *negate=False*, *axis='xz'*)

Get a tranformation list from a positions list and normal.

Note: getChainTransform2 is using the latest position on the chain

Parameters

- **positions** (*list of vector*) – List with the chain positions.
- **normal** (*vector*) – Normal direction.
- **negate** (*bool*) – If true invert the chain orientation.

Returns

The list containing the transformation matrix for the chain.

Return type

list of matrix

```
>>> tra.getChainTransform2(self.guide.apos,
                           self.normal,
                           self.negate)
```

`mgear.core.transform.`**`getClosestPolygonFromTransform`**(*geo*, *loc*)

Get closest polygon from transform

Parameters

- **geo** (*dagNode*) – Mesh object
- **loc** (*matrix*) – location transform

Returns

Closest Polygon

`mgear.core.transform.`**`getDistance2`**(*obj0*, *obj1*)

Get the distance between 2 objects.

Parameters

- **obj0** (*dagNode*) – Object A
- **obj1** (*dagNode*) – Object B

Returns

Distance length

Return type

float

`mgear.core.transform.`**`getFilteredTransform`**(*m*, *translation=True*, *rotation=True*, *scaling=True*)

Retrieve a transformation filtered.

Parameters

- **m** (*matrix*) – the reference matrix

- **translation** (*bool*) – If true the return matrix will match the translation.
- **rotation** (*bool*) – If true the return matrix will match the rotation.
- **scaling** (*bool*) – If true the return matrix will match the scaling.

Returns
The filtered matrix

Return type
matrix

mgear.core.transform.**getInterpolateTransformMatrix**(*t1*, *t2*, *blend=0.5*)

Interpolate 2 matrix.

Parameters

- **t1** (*matrix*) – Input matrix 1.
- **t2** (*matrix*) – Input matrix 2.
- **blend** (*float*) – The blending value. Default 0.5

Returns
The newly interpolated transformation matrix.

Return type
matrix

```
>>> t = tra.getInterpolateTransformMatrix(self.fk_ctl[0],
                                          self.tws1A_npo,
                                          .3333)
```

mgear.core.transform.**getOffsetPosition**(*node*, *offset=[0, 0, 0]*)

Get an offset position from dagNode

Parameters

- **node** (*dagNode*) – The dagNode with the original position.
- **offset** (*list of float*) – Ofsset values for xyz. exp : [1.2, 4.6, 32.78]

Returns
the new offset position.

Return type
list of float

Example

```
self.root = self.addRoot()
vTemp = tra.getOffsetPosition( self.root, [0,-3,0.1])
self.knee = self.addLoc("knee", self.root, vTemp)
```

mgear.core.transform.**getPositionFromMatrix**(*in_m*)

Get the position values from matrix

Parameters
in_m (*matrix*) – The input Matrix.

Returns
The position values for xyz.

Return type
list of float

mgear.core.transform.**getRotationFromAxis**(*in_a*, *in_b*, *axis='xy'*, *negate=False*)

Get the matrix rotation from a given axis.

Parameters

- **in_a** (*vector*) – Axis A
- **in_b** (*vector*) – Axis B
- **axis** (*str*) – The axis to use for the orientation. Default: "xy"
- **negate** (*bool*) – negates the axis orientation.

Returns
The newly created matrix.

Return type
matrix

Example

```
x = datatypes.Vector(0,-1,0)
x = x * tra.getTransform(self.eff_loc)
z = datatypes.Vector(self.normal.x,
                     self.normal.y,
                     self.normal.z)
z = z * tra.getTransform(self.eff_loc)
m = tra.getRotationFromAxis(x, z, "xz", self.negate)
```

mgear.core.transform.**getSymmetricalTransform**(*t*, *axis='yz'*, *fNegScale=False*)

Get the symmetrical tranformation

Get the symmetrical tranformation matrix from a define 2 axis mirror plane. exp:"yz".

Parameters

- **t** (*matrix*) – The transformation matrix to mirror.
- **axis** (*str*) – The mirror plane.
- **fNegScale** (*bool*) – This function is not yet implemented.

Returns
The symmetrical tranformation matrix.

Return type
matrix

mgear.core.transform.**getTransform**(*node*)

Return the transformation matrix of the dagNode in worldSpace.

Parameters
node (*dagNode*) – The dagNode to get the translation

Returns
The transformation matrix

Return type
matrix

`mgear.core.transform.`**`getTransformFromPos`**(*pos*)

Create a transformation Matrix from a given position.

Parameters

pos (*vector*) – Position for the transformation matrix

Returns

The newly created transformation matrix

Return type

matrix

```
>>>  t = tra.getTransformFromPos(self.guide.pos["root"])
```

`mgear.core.transform.`**`getTransformLookingAt`**(*pos*, *lookat*, *normal*, *axis='xy'*, *negate=False*)

Return a transformation mstrix using vector positions.

Return the transformation matrix of the dagNode oriented looking to an specific point.

Parameters

- **pos** (*vector*) – The position for the transformation
- **lookat** (*vector*) – The aiming position to stablish the orientation
- **normal** (*vector*) – The normal control the transformation roll.
- **axis** (*str*) – The 2 axis used for lookat and normal. Default "xy"
- **negate** (*bool*) – If true, invert the aiming direction.

Returns

The transformation matrix

Return type

matrix

```
>>>  t = tra.getTransformLookingAt(self.guide.pos["heel"],
                                    self.guide.apos[-4],
                                    self.normal,
                                    "xz",
                                    self.negate)
```

`mgear.core.transform.`**`getTranslation`**(*node*)

Return the position of the dagNode in worldSpace.

Parameters

node (*dagNode*) – The dagNode to get the translation

Returns

The transformation matrix

Return type

matrix

`mgear.core.transform.`**`get_closes_transform`**(*target_transform*, *source_transforms*)

Summary

Parameters

- **target_transform** (*dagNode*) – target transform
- **source_transforms** (*[dagNode]*) – objects to check distance

Returns
ordered transform list

Return type
list

`mgear.core.transform.`**`get_orientation_from_polygon`**(*face*)

Summary

Parameters

- **`face`** (*`TYPE`*) – Description
- **`loc`** (*`TYPE`*) – Description

Returns
Description

Return type
TYPE

`mgear.core.transform.`**`get_raycast_translation_from_mouse_click`**(*mesh*, *mpx*, *mpy*)

get the raycasted translation of the mouse position

Parameters

- **`mesh`** (*`str`*) – mesh name
- **`mpx`** (*`int`*) – mouse position x
- **`mpy`** (*`int`*) – mouse position x

Returns
XYZ position

Return type
list

`mgear.core.transform.`**`interpolate_rotation`**(*obj*, *targets*, *blends*)

`mgear.core.transform.`**`interpolate_scale`**(*obj*, *targets*, *blends*)

`mgear.core.transform.`**`matchWorldTransform`**(*source*, *target*)

Match 2 dagNode transformations in world space.

Parameters

- **`source`** (*`dagNode`*) – The source dagNode
- **`target`** (*`dagNode`*) – The target dagNode

Returns
None

`mgear.core.transform.`**`quaternionDotProd`**(*q1*, *q2*)

Get the dot product of 2 quaternion.

Parameters

- **`q1`** (*`quaternion`*) – Input quaternion 1.
- **`q2`** (*`quaternion`*) – Input quaternion 2.

Returns
The dot proct quaternion.

Return type
quaternion

`mgear.core.transform.`**`quaternionSlerp`**`(`*q1*, *q2*, *blend*`)`

Get an interpolate quaternion based in slerp function.

Parameters

- **q1** (*quaternion*) – Input quaternion 1.
- **q2** (*quaternion*) – Input quaternion 2.
- **blend** (*float*) – Blending value.

Returns
The interpolated quaternion.

Return type
quaternion

Example

```
q = quaternionSlerp(datatypes.Quaternion(
                    t1.getRotationQuaternion()),
                    datatypes.Quaternion(
                        t2.getRotationQuaternion()), blend)
```

`mgear.core.transform.`**`resetTransform`**`(`*node*, *t=True*, *r=True*, *s=True*`)`

Reset the scale, rotation and translation for a given dagNode.

Parameters

- **node** (*dagNode*) – The object to reset the transforms.
- **t** (*bool*) – If true translation will be reseted.
- **r** (*bool*) – If true rotation will be reseted.
- **s** (*bool*) – If true scale will be reseted.

Returns
None

`mgear.core.transform.`**`setMatrixPosition`**`(`*in_m*, *pos*`)`

Set the position for a given matrix

Parameters

- **in_m** (*matrix*) – The input Matrix.
- **pos** (*list of float*) – The position values for xyz

Returns
The matrix with the new position

Return type
matrix

```
>>> tnpo = tra.setMatrixPosition(tOld, tra.getPositionFromMatrix(t))
```

```
>>> t = tra.setMatrixPosition(t, self.guide.apos[-1])
```

mgear.core.transform.**setMatrixRotation**(*m*, *rot*)

Set the rotation for a given matrix

Parameters

- **in_m** (*matrix*) – The input Matrix.
- **rot** (*list of float*) – The rotation values for xyz

Returns
The matrix with the new rotation

Return type
matrix

mgear.core.transform.**setMatrixScale**(*m*, *scl=[1, 1, 1]*)

Set the scale for a given matrix

Parameters

- **in_m** (*matrix*) – The input Matrix.
- **scl** (*list of float*) – The scale values for xyz

Returns
The matrix with the new scale

Return type
matrix

mgear.core.utils module

Utilitie functions

mgear.core.utils.**as_pynode**(*obj*)

Check and convert a given string to Pynode

If the object is not str or unicode or PyNode will raise type error

Parameters
obj (*str, unicode, PyNode*) – Object to check and/or convert to PyNode

Returns
the pynode object

Return type
PyNode

mgear.core.utils.**filter_nurbs_curve_selection**(*func*)

mgear.core.utils.**gatherCustomModuleDirectories**(*envvarkey*, *defaultModulePath*, *component=False*)

returns component directory

Parameters

- **envvarkey** – The environment variable key name, that is searched
- **defaultModulePath** – The default module path for search in.

Returns
[]string}

Return type
Dict{string

`mgear.core.utils.getModuleBasePath(directories, moduleName)`

search component path

`mgear.core.utils.get_dag_path(name)`

Gets the dag path for the specified object name.

Parameters
`name` (*str*) – Name of the object in the Maya Scene.

Returns
The dag path to the specified name, else None.

Return type
OpenMaya.MDagPath

`mgear.core.utils.get_frame_rate()`

Returns the current scene's fps.

Returns
The fps for the current scene's timeline.

Return type
int

`mgear.core.utils.get_maya_path()`

Gets the path to the folder where Maya binary lives

Note: Only works from inside Maya, as Maya adds the path on startup.

Returns
Absolute path to the binary folder that contains maya executable

Return type
str

`mgear.core.utils.get_os()`

Gets the OS that Maya is running in.

Returns
Current OS

Return type
str

`mgear.core.utils.importFromStandardOrCustomDirectories(directories, defaultFormatter, customFormatter, moduleName)`

Return imported module

Parameters

- `directories` – the directories for search in. this is got by gatherCustomModuleDirectories
- `defaultFormatter` – this represents module structure for default module. for example "mgear.core.shifter.component.{}"
- `customFormatter` – this represents module structure for custom module. for example "{0}.{1}"

Returns
imported module

Return type
module

`mgear.core.utils.is_odd(num)`

Check if the number is odd.

Arguments: num (int): the number

Returns
True or False

Return type
bool

`mgear.core.utils.one_undo(func)`

Decorator - guarantee close chunk.

type: (function) -> function

`mgear.core.utils.set_frame_rate(fps)`

Set Maya Scene's frame rate(fps).

Parameters
fps (*int*) – frames per a second for playback.

`mgear.core.utils.timeFunc(func)`

Use as a property to time any desired function

`mgear.core.utils.viewport_off(func)`

Decorator - Turn off Maya display while func is running.

if func will fail, the error will be raised after.

type: (function) -> function

mgear.core.vector module

Functions to work with vectors

`class mgear.core.vector.Blade(t=pymel.core.datatypes.Matrix)`

Bases: `object`

The Blade object for shifter guides

`mgear.core.vector.calculatePoleVector(p1, p2, p3, poleDistance=1, time=1)`

This function takes 3 PyMEL PyNodes as inputs. Creates a pole vector position at a "nice" distance away from a triangle of positions. Normalizes the bone lengths relative to the knee to calculate straight ahead without shifting up and down if the bone lengths are different. Returns a pymel.core.datatypes.Vector

Parameters

- **p1** (*dagNode*) – Object A
- **p2** (*dagNode*) – Object B
- **p3** (*dagNode*) – Object C
- **poleDistance** (*float*) – distance of the pole vector from the mid point

Returns
The transposed vector.

Return type
vector

`mgear.core.vector.getDistance(v0, v1)`

Get the distance between 2 vectors

Parameters

- **v0** (*vector*) – vector A.
- **v1** (*vector*) – vector B.

Returns
Distance length.

Return type
float

`mgear.core.vector.getDistance2(obj0, obj1)`

Get the distance between 2 objects.

Parameters

- **obj0** (*dagNode*) – Object A
- **obj1** (*dagNode*) – Object B

Returns
Distance length

Return type
float

`mgear.core.vector.getPlaneBiNormal(v0, v1, v2)`

Get the binormal vector of a plane (Defined by 3 positions).

Parameters

- **v0** (*vector*) – First position on the plane.
- **v1** (*vector*) – Second position on the plane.
- **v2** (*vector*) – Third position on the plane.

Returns
The binormal.

Return type
vector

`mgear.core.vector.getPlaneNormal(v0, v1, v2)`

Get the normal vector of a plane (Defined by 3 positions).

Parameters

- **v0** (*vector*) – First position on the plane.
- **v1** (*vector*) – Second position on the plane.
- **v2** (*vector*) – Third position on the plane.

Returns
The normal.

Return type
vector

mgear.core.vector.**getTransposedVector**(*v*, *position0*, *position1*, *inverse=False*)

Get the transposed vector.

Parameters

- **v** (*vector*) – Input Vector.
- **position0** (*vector*) – Position A.
- **position1** (*vector*) – Position B.
- **inverse** (*bool*) – Invert the rotation.

Returns
The transposed vector.

Return type
vector

```
>>> normal = vec.getTransposedVector(self.normal,
                                     [self.guide.apos[0],
                                      self.guide.apos[1]],
                                     [self.guide.apos[-2],
                                      self.guide.apos[-1]])
```

mgear.core.vector.**linearlyInterpolate**(*v0*, *v1*, *blend=0.5*)

Get the vector interpolated between 2 vectors.

Parameters

- **v0** (*vector*) – vector A.
- **v1** (*vector*) – vector B.
- **blend** (*float*) – Blending value.

Returns
The interpolated vector.

Return type
vector

mgear.core.vector.**rotateAlongAxis**(*v*, *axis*, *a*)

Rotate a vector around a given axis defined by other vector.

Parameters

- **v** (*vector*) – The vector to rotate.
- **axis** (*vector*) – The axis to rotate around.
- **a** (*float*) – The rotation angle in radians.

mgear.core.version module

mgear.core.widgets module

mgear.core.wmap module

mgear.core.wmap.**export_weights**(*deformer*, *filePath*)

Export the wmap to a json file

Parameters

- **deformer** (*PyNode or str*) – Name or pynode of a deformer with weight map
- **filePath** (*str*) – Path to save the file

mgear.core.wmap.**export_weights_selected**(*filePath=None*, **args*)

Export the wmap to a json file from selected objet

Parameters
filePath (*str*) – Path to save the file. If None wil pop up file browser

mgear.core.wmap.**file_browser**(*mode=1*)

open file browser

Parameters
mode (*int, optional*) – 0 save mode, 1 load mode

Returns
file path

Return type
str

mgear.core.wmap.**get_weights**(*deformer*)

Get the weight map from a given deformers

It supports multiple objects/weight maps for one sigle deformer

Parameters
deformer (*PyNode or str*) – Name or pynode of a deformer with weight map

Returns
The weights dictionary

Return type
dict

mgear.core.wmap.**import_weights**(*deformer*, *filePath*)

Import the wmap from a json file

Parameters

- **deformer** (*PyNode or str*) – Name or pynode of a deformer to assign the wmap
- **filePath** (*str*) – Path to load the file

mgear.core.wmap.**import_weights_selected**(*filePath=None*, **args*)

Import the wmap to from json file from selected objet

Parameters
filePath (*str*) – Path to load the file. If None wil pop up file browser

`mgear.core.wmap.`**`set_weights`**`(`*deformer*, *dataWeights*`)`

Set the weight map from a given deformers

It supports multiple objects/weight maps for one sigle deformer

Parameters

`deformer` (*`PyNode or str`*) – Name or pynode of a deformer with weight map

Module contents

`mgear.core.`**`aboutMgear`**`(`**args*`)`

About mgear

`mgear.core.`**`getMayaVer`**`()`

Get Maya version

Returns

Maya version

mgear.crank package

Submodules

mgear.crank.crank_tool module

mgear.crank.crank_ui module

`class` `mgear.crank.crank_ui.`**`Ui_Form`**

Bases: `object`

`retranslateUi``(`*Form*`)`

`setupUi``(`*Form*`)`

mgear.crank.menu module

`mgear.crank.menu.`**`install`**`()`

Install Crank submenu

mgear.crank.version module

Module contents

mgear.flex package

Submodules

mgear.flex.analyze module

flex.analyze

flex.analyze module contains functions which allows you analyze the shapes you want to update with Flex

module
flex.analyze

mgear.flex.analyze_widget module

flex.analyze_widget

Contains the Flex Analyze interface

module
flex.analyze_widget

class `mgear.flex.analyze_widget.`**`FlexAnalyzeDialog`**(**args: Any*, ***kwargs: Any*)

Bases: `QDialog`

The Flex analyze widgets

Flex analyze is a side by side list widget style that will allow you to check which shapes matches.

Creates all the user interface widgets

Parameters
parent (*PySide2.QtWidgets*) – the parent widget for the Flex dialog widget

`add_item`(*source*, *target*, *match*, *count*, *bbox*)

Handles adding items to the table widget

Parameters

- **source** (*string*) – the source shape element
- **target** (*string*) – the target corresponding shape element matching source
- **match** (*bool*) – whether the type matches

mgear.flex.attributes module

flex.attributes

A simple module listing all attributes classifications inside Maya or a simplified verison of long attributes in maya

module
flex.attributes

mgear.flex.colors module

flex.colors

Simple collection of QtColors

module
flex.colors

mgear.flex.decorators module

flex.decorators

flex.decorators module contains utility functions that you can use as functions decorators.

module
flex.decorators

`mgear.flex.decorators.`**`finished_running`**(*function*)

Displays a end of process viewport message

Parameters
`function` (*`function`*) – your decorated function

Returns
your decorated function

Return type
function

`mgear.flex.decorators.`**`hold_selection`**(*function*)

Holds the current Maya selection after running the function

Parameters
`function` (*`function`*) – your decorated function

Returns
your decorated function

Return type
function

`mgear.flex.decorators.`**`isolate_view`**(*function*)

Isolates the view panels while function is running

Parameters
`function` (*`function`*) – your decorated function

Returns
your decorated function

Return type
function

`mgear.flex.decorators.`**`set_focus`**(*function*)

Set focus on Flex window

Sets focus on Flex UI.

Parameters
`function` (*`function`*) – your decorated function

Returns
your decorated function

Return type
function

`mgear.flex.decorators.show_view(function)`

Shows the isolated views panels after function runs

Parameters
function (*function*) – your decorated function

Returns
your decorated function

Return type
function

`mgear.flex.decorators.timer(function)`

Function timer

Simple timer function decorator that you can use on Flex to time your code execution

Parameters
function (*function*) – your decorated function

Returns
your decorated function

Return type
function

mgear.flex.flex module

mgear.flex.flex_widget module

flex.ui

Contains the Flex user interface

module
flex.ui

`class mgear.flex.flex_widget.FlexDialog(*args: Any, **kwargs: Any)`

Bases: `QDialog`

The Flex UI widgets

Flex UI contains several options you can tweak to customise your rig update

Creates all the user interface widgets

Parameters
parent (*PySide2.QtWidgets*) – the parent widget for the Flex dialog widget

`deformed_widgets()`

Creates the deformed options widgets

`layout_widgets()`

Creates the general UI layouts

models_groups_widgets()
Creates the source and target widgets area

options_widgets()
Create the options widget area

run_widgets()
Creates the run widgets area

transformed_widgets()
Creates the transformed options widgets

mgear.flex.menu module

flex.menu

Flex menu handles adding the Flex menu item inside the Maya mGear menu.

module
flex.menu

mgear.flex.menu.**install**()
Installs Flex sub-menu

mgear.flex.query module

flex.query

flex.query module contains a collection of functions useful for the analyze and update functions of Flex

module
flex.query

mgear.flex.query.**get_clean_matching_shapes**(*source*, *target*)
Returns the prefix-less found shapes under the given groups

Parameters

- **source** (*string*) – source group containing shapes in Maya
- **target** (*string*) – target group containing shapes in Maya

Returns
The matching target shapes names without prefix

Return type
dict, dict

mgear.flex.query.**get_deformers**(*shape*)
Returns a dict with each deformer found on the given shape

Parameters
shape (*str*) – the shape node name

Returns
the deformers found on shape sorted by type

Return type
dict

mgear.flex.query.**get_dependency_node**(*element*)

Returns a Maya MFnDependencyNode from the given element

Parameters
`element` (*string*) – Maya node to return a dependency node class object

Returns
the element in a Maya MFnDependencyNode object

Return type
MFnDependencyNode

mgear.flex.query.**get_matching_shapes**(*source_shapes*, *target_shapes*)

Returns the matching shapes

This Function will return a dict that contains the target matching shape name from the source.

Parameters

- `source_shapes` (*dict*) – sources dictionary containing prefix-less shapes
- `target` (*dict*) – targets dictionary containing prefix-less shapes

Returns
The matching target shapes names

Return type
dict

Note: This function is the core idea of how Flex finds matching shapes from a source group to the target. Because Flex is not part of a specific studio pipeline this matching is **shapes name based**. Because some studios might bring the source scene into the rig scene as a reference or as an import we cover those two cases.

Using this dict is the fastest way (so far found) to deal with a huge amount of names. Finding the matching names on a scene with more than 2000 shapes takes 0.0009… seconds.

---

mgear.flex.query.**get_matching_shapes_from_group**(*source*, *target*)

Returns the matching shapes on the given groups

Parameters

- `source` (*string*) – source group containing shapes in Maya
- `target` (*string*) – target group containing shapes in Maya

Returns
The matching target shapes names

Return type
dict

mgear.flex.query.**get_missing_shapes**(*source_shapes*, *target_shapes*)

Returns the missing shapes

This Function will return a dict that contains the missing shape found on the target.

Parameters

- `source_shapes` (*dict*) – sources dictionary containing prefix-less shapes
- `target` (*dict*) – targets dictionary containing prefix-less shapes

Returns
The missing target shapes names

Return type
dict

`mgear.flex.query.get_missing_shapes_from_group(source, target)`

Returns the missing shapes from the given source and target group

Parameters

- **source** (*string*) – source group containing shapes in Maya
- **target** (*string*) – source group containing shapes in Maya

Returns
The missing target shapes names

Return type
dict

`mgear.flex.query.get_parent(element)`

Returns the first parent found for the given element

Parameters
element (*string*) – A Maya dag node

`mgear.flex.query.get_prefix_less_dict(elements)`

Returns a dict containing each element with a stripped prefix

This Function will return a dict that contains each element resulting on the element without the found prefix

Parameters
elements (*list*) – List of all your shapes

Returns
The matching prefix-less elements

Return type
dict

Note: Because Flex is not part of a specific studio pipeline we cover two different ways to bring the source shapes inside your rig. You can either import the source group with the meshes or use a Maya reference. This function will strip the prefix whether your object is part of a namespace or a double name getting a full path naming.

`mgear.flex.query.get_prefix_less_name(element)`

Returns a prefix-less name

Parameters
elements (*str*) – element top use on the search

Returns
The prefix-less name

Return type
str

`mgear.flex.query.get_resources_path()`

Gets the directory path to the resources files `mgear.flex.query.get_shape_orig(shape)`

Finds the orig (intermediate shape) on the given shape

Parameters
: **shape** (*str*) – maya shape node

Returns
: the found orig shape

Return type
: str

Note: There are several ways of searching for the orig shape in Maya. Here we query it by first getting the given shape history on the component type attribute (inMesh, create..) then filtering on the result the same shape type. There might be more optimised and stable ways of doing this.

`mgear.flex.query.get_shape_type_attributes(shape)`

Returns a dict with the attributes names depending on the shape type

This function returns the points, output, input and axes attributes for the corresponding shape type. Mesh type of nodes will be set as default but nurbs surfaces and nurbs curves are supported too.

on mesh nodes: points = pnts
: output = outMesh input = inMesh p_axes = (pntx, pnty, pntz)

on nurbs nodes: points = controlPoints
: output = local input = create p_axes = (xValue, yValue, zValue)

Parameters
: **shape** (*str*) – maya shape node

Returns
: corresponding attributes names

Return type
: dict

`mgear.flex.query.get_shapes_from_group(group)`

Gets all object shapes existing inside the given group

Parameters
: **group** (*str*) – maya transform node

Returns
: list of shapes objects

Return type
: list str

`mgear.flex.query.get_temp_folder()`

Returns the user temporary folder in a Maya friendly matter

Returns
: temp folder path

Return type
: str

`mgear.flex.query.get_transform_selection()`

Gets the current dag object selection

Returns the first selected dag object on a current selection that is a transform node

Returns
the first element of the current maya selection

Return type
str

`mgear.flex.query.get_vertice_count(`*shape*`)`

Returns the number of vertices for the given shape

Parameters
shape (*string*) – The maya shape node

Returns
The number of vertices found on shape

Return type
int

`mgear.flex.query.is_lock_attribute(`*element*, *attribute*`)`

Returns if the given attribute on the element is locked

Parameters

- **element** (*string*) – Maya node name
- **attribute** (*string*) – Maya attribute name. Must exist

Returns
if attribute is locked

Return type
bool

`mgear.flex.query.is_matching_bouding_box(`*source*, *target*, *tolerance=0.05*`)`

Checks if the source and target shape have the same bounding box

Parameters

- **source** (*string*) – source shape node
- **target** (*string*) – target shape node
- **tolerance** (*float*) – difference tolerance allowed. Default 0.001

Returns
If source and target matches their bounding box

Return type
bool

`mgear.flex.query.is_matching_count(`*source*, *target*`)`

Checks if the source and target shape have the same amount of vertices

Parameters

- **source** (*string*) – source shape node
- **target** (*string*) – target shape node

Returns
If source and target matches vertices count or not

Return type
bool

`mgear.flex.query.is_matching_type(source, target)`

Checks if the source and target shape type matches

Parameters

- **source** (*string*) – source shape node
- **target** (*string*) – target shape node

Returns
If source and target matches or not

Return type
bool

`mgear.flex.query.is_maya_batch()`

Returns if the current session is a Maya batch session or not

Returns
if Maya is on batch mode or not

Return type
bool

`mgear.flex.query.is_valid_group(group)`

Checks if group is valid

Simply checks if the given group exists in the current Maya session and if it is a valid transform group.

Parameters
group (*str*) – a maya transform node

Returns
If the group is valid

Return type
bool

`mgear.flex.query.lock_unlock_attribute(element, attribute, state)`

Unlocks the given attribute on the given element

Parameters

- **element** (*string*) – Maya node name
- **attribute** (*string*) – Maya attribute name. Must exist
- **state** (*bool*) – If we should lock or unlock

Returns
If the setting was successful or not

Return type
bool

mgear.flex.update module

flex.update

flex.update module handles the updating rig process

module
flex.update

`mgear.flex.update.`**`update_attribute`**(*source*, *target*, *attribute_name*)

Updates the given attribute value

..note:: This in a generic method to setAttr all type of attributes
inside Maya. Using the getSetAttrCmds from the MPLug class allows avoiding to create one method for each type of attribute inside Maya as the setAttr command will differ depending on the attribute type and data.

This method is faster than using PyMel attribute set property.

Parameters

- **`source`** (*`str`*) – the maya source node
- **`target`** (*`str`*) – the maya target node
- **`attribute_name`** (*`str`*) – the attribute name to set in the given target

`mgear.flex.update.`**`update_blendshapes_nodes`**(*source_nodes*, *target_nodes*)

Update all target shapes with the given source shapes

Parameters

- **`source_nodes`** (*`list(str)`*) – source blendshape nodes
- **`target_nodes`** (*`list(str)`*) – target blendshape nodes

`mgear.flex.update.`**`update_clusters_nodes`**(*shape*, *weight_files*)

Updates the given shape cluster weights using the given files

Parameters

- **`shape`** (*`str`*) – the shape node name containing the cluster deformers
- **`weight_files`** (*`list(str)`*) – weight files names for each cluster deformer

`mgear.flex.update.`**`update_deformed_shape`**(*source*, *target*, *mismatching_topology=True*)

Updates the target shape with the given source shape content

Parameters

- **`source`** (*`str`*) – maya shape node
- **`target`** (*`str`*) – maya shape node
- **`mismatching_topology`** (*`bool`*) – ignore or not mismatching topologies

`mgear.flex.update.`**`update_maya_attributes`**(*source*, *target*, *attributes*)

Updates all maya attributes from the given source to the target

Parameters

- **`source`** (*`str`*) – maya shape node
- **`target`** (*`str`*) – maya shape node
- **`attributes`** (*`list`*) – list of Maya attributes to be updated

`mgear.flex.update.update_plugin_attributes(source, target)`

Updates all maya plugin defined attributes

Parameters

- **source** (*str*) – maya shape node
- **target** (*str*) – maya shape node

`mgear.flex.update.update_skincluster_node(source_skin, target_skin)`

Updates the skin weights on the given target skin from the source skin

Parameters

- **source_skin** (*str*) – the source skin cluster node name
- **target_skin** (*str*) – the target skin cluster node name

`mgear.flex.update.update_transform(source, target)`

Updates the transform node on target

This method creates a duplicate of the transform node on source and uses is as the new parent transform for the target shape

Parameters

- **source** (*str*) – maya shape node
- **target** (*str*) – maya shape node

`mgear.flex.update.update_transformed_shape(source, target, hold_transform)`

Updates the target shape with the given source shape content

Parameters

- **source** (*str*) – maya shape node
- **target** (*str*) – maya shape node
- **hold_transform** (*bool*) – keeps the transform node position values

`mgear.flex.update.update_user_attributes(source, target)`

Updates the target shape attributes with the given source shape content

Parameters

- **source** (*str*) – maya shape node
- **target** (*str*) – maya shape node

Note: This method loops twice on the use attributes. One time to add the missing attributes and the second to set their value. This allows avoiding issues when dealing with child attributes.

`mgear.flex.update.update_uvs_sets(shape)`

Forces a given mesh shape uvs to update

mgear.flex.update_utils module

flex.update_utils

flex.update_utils module contains some simple methods that work as utilities for the flex update process

module
flex.update_utils

`mgear.flex.update_utils.`**`add_attribute`**`(`*source*, *target*, *attribute_name*`)`

Adds the given attribute to the given object

Note: This is a generic method to **addAttr** all type of attributes inside Maya. Using the getAddAttrCmd from the MFnAttribute class allows avoiding to create one method for each type of attribute inside Maya as the addAttr command will differ depending on the attribute type and data.

Parameters

- **source** (*str*) – the maya source node
- **target** (*str*) – the maya target node
- **attribute_name** (*str*) – the attribute name to add in the given element

`mgear.flex.update_utils.`**`clean_uvs_sets`**`(`*shape*`)`

Deletes all uv sets besides map1

This is used to be able to update target shapes with whatever the source shape has. This is only relevant for mesh shape types.

Parameters
shape (*string*) – The Maya shape node

`mgear.flex.update_utils.`**`copy_blendshape_node`**`(`*node*, *target*`)`

Copies the given blendshape node into the given target shape

Parameters

- **node** (*str*) – blendshape node
- **target** (*str*) – target shape node

Returns
copied blenshape node

Return type
str

`mgear.flex.update_utils.`**`copy_map1_name`**`(`*source*, *target*`)`

Copies the name of the uvSet at index zero (map1) to match it

Parameters

- **source** (*str*) – maya shape node
- **target** (*str*) – maya shape node

`mgear.flex.update_utils.`**`create_clusters_backup`**`(`*shape*, *nodes*`)`

Generates weight files for the given cluster nodes in the given shape

Parameters

- **shape** (*str*) – the shape node name containing the cluster deformers nodes
- **nodes** (*list*) – the cluster nodes

Returns
cluster weight files names

Return type
dict

`mgear.flex.update_utils.`**`create_deformers_backups`**(*source*, *target*, *shape_orig*, *deformers*)

Handles creating the correct backup shapes for the given deformers

Parameters

- **source** (*str*) – the shape containing the new shape
- **target** (*str*) – the shape containing the deformers
- **shape_orig** (*str*) – the intermediate shape from the target shape
- **deformers** (*dict*) – deformers used on target

Returns
deformers backups nodes created

Return type
list, list

`mgear.flex.update_utils.`**`create_duplicate`**(*shape*, *duplicate_name*)

Creates a shape node duplicate

Parameters

- **shape** (*str*) – the shape node to duplicate
- **name** (*str*) – the name for the duplicate

Returns
the duplicated shape node

Return type
str

`mgear.flex.update_utils.`**`create_wrap`**(*source*, *target*, *intermediate=None*)

Creates a wrap deformer on the target by using source as driver

Parameters

- **source** (*str*) – the maya source node
- **target** (*str*) – the maya target node
- **intermediate** (*str*) – the intermediate shape to use on the warp node

Returns
wrap node

Return type
str

`mgear.flex.update_utils.`**`delete_transform_from_nodes`**(*nodes*)

Deletes the dag object transform node found from the given nodes

Parameters
shape (*list*) – nodes names

`mgear.flex.update_utils.`**`filter_shape_orig`**`(`*shape*, *intermediate*`)`

Filters whether the intermediate shape provided should be used or not

if an intermediate isn't provided then

Parameters

- **`shape`** (*`str`*) – the shape node name
- **`intermediate`** (*`str`*) – the intermediate shape name

Returns
the valid intermediate shape

Return type
str

`mgear.flex.update_utils.`**`set_deformer_off`**`(`*deformer*`)`

Set envelope attribute to **0** on the given deformer

Parameters
`deformer` (*`str`*) – deformer node

`mgear.flex.update_utils.`**`set_deformer_on`**`(`*deformer*`)`

Set envelope attribute to **1** on the given deformer

Parameters
`deformer` (*`str`*) – deformer node

`mgear.flex.update_utils.`**`set_deformer_state`**`(`*deformers*, *enable*`)`

Set envelope attribute to one on the given deformers dictionary

Parameters

- **`deformers`** (*`type`*) – dict containing the deformers set by type
- **`enable`** (*`bool`*) – on or off state for the given deformers

`mgear.flex.update_utils.`**`update_shape`**`(`*source*, *target*`)`

Connect the shape output from source to the input shape on target

Parameters

- **`source`** (*`str`*) – maya shape node
- **`target`** (*`str`*) – maya shape node

mgear.flex.version module

Module contents

FLEX

Flex is the mGear models (geometry) update tool inside rigs.

module
flex.__init__

mgear.rigbits package

Subpackages

mgear.rigbits.facial_rigger package

Submodules

mgear.rigbits.facial_rigger.brow_rigger module

mgear.rigbits.facial_rigger.constraints module

mgear.rigbits.facial_rigger.eye_rigger module

mgear.rigbits.facial_rigger.helpers module

mgear.rigbits.facial_rigger.lib module

mgear.rigbits.facial_rigger.lips_rigger module

Module contents

mgear.rigbits.facial_rigger2 package

Submodules

mgear.rigbits.facial_rigger2.eye_rigger module

mgear.rigbits.facial_rigger2.eye_riggerUI module

mgear.rigbits.facial_rigger2.helpers module

mgear.rigbits.facial_rigger2.lib module

Module contents

mgear.rigbits.sdk_manager package

Submodules

mgear.rigbits.sdk_manager.SDK_manager_ui module

mgear.rigbits.sdk_manager.SDK_transfer_ui module

mgear.rigbits.sdk_manager.core module

mgear.rigbits.sdk_manager.core.**ctl_from_list**(*in_list*, *SDK=False*, *animTweak=False*)

Returns either the SDK's or animTweaks from the in_list. If given a SDK, it will find the animTweak pair and vise versa. To qualify as SDK ctl must have "is_SDK" attr, or "is_tweak" attr for animTweak

Parameters

- **in_list** (*list[PyNode]*) – List of PyNodes to sort through
- **SDK** (*bool*) – If you want SDK ctls
- **animTweak** (*bool*) – If you want animTweak ctls

Returns
list [List of either SDK ctls or animTweaks]

mgear.rigbits.sdk_manager.core.**delete_current_value_keys**(*current_driver_val*, *node*, *sourceDriverFilter*)

Parameters

- **()** (*sourceDriverFilter*) –
- **()** –

Returns
n/a

mgear.rigbits.sdk_manager.core.**driver_ctl_from_joint**(*joint*)

Will try find the Driver control given the joint by searching through the mGear nodes.

Parameters
joint (*PyNode*) – joint to search connections on

Returns
Control

Return type
PyNode

mgear.rigbits.sdk_manager.core.**get_current_SDKs**()

If SDK ctls are selected, will return only the SDK nodes Attatched to those in the selection. If nothing is selected, will get all the SDK nodes in the scene and return them.

Returns
SDKs_to_set (list) - list of SDKs as Pynodes

mgear.rigbits.sdk_manager.core.**get_driven_from_attr**(*driverAttr*, *is_SDK=False*)

Returns a list of driven controls given the driver attr

Parameters

- **driverAttr** (*PyNode*) – the driver attr to search
- **is_SDK** (*bool*) – if True, will check if the is_SDK attr is present before
- **list.** (*adding to driven_ctls*) –

Returns
list [List of unicode names]

mgear.rigbits.sdk_manager.core.**get_driver_from_driven**(*drivenCtl*)

Finds the Driver controls for a given driven ctl

Parameters
drivenCtl (*PyNode*) – A Driven Node to query.

Returns
list [All found Driver Nodes]

`mgear.rigbits.sdk_manager.core.`**`get_driver_keys`**(*driverAttr*, *firstKey=None*, *prevKey=None*, *nextKey=None*, *lastKey=None*)

Returns a list of Driver key values for the given driverAttr.

If all optional arguments are None, will return list of all values

Parameters

- **driverAttr** (*PyNode.Attribute*) – Driver Ctl.attr
- **firstKey** (*bool*) –
- **prevKey** (*bool*) –
- **nextKey** (*bool*) –
- **lastKey** (*bool*) –

Returns
List (If all optional None) - List of driver key values float (If one specified) - The float value for the driver on that key.

`mgear.rigbits.sdk_manager.core.`**`get_info`**(*node*)

Given either the SDK box, or Anim ctl, will find other and return it

Parameters
node (*PyNode*) – either the SDK box or Anim ctl

Returns
list [PyNode(SDK box), PyNode(anim ctl)]

`mgear.rigbits.sdk_manager.core.`**`joint_from_driver_ctl`**(*node*)

Will try find the joint given the Driver control by searching through the mGear nodes.

TO DO:
Expand this to be more robust. Check all channels / not rely on translate connections only.

Parameters
node (*PyNode*) – node to search connections on

Returns
joint

Return type
PyNode

`mgear.rigbits.sdk_manager.core.`**`key_at_current_values`**(*drivenCtls*, *keyChannels*, *driver*, *driverAtt*, *inTanType='linear'*, *outTanType='linear'*, *zeroKey=False*)

Helper function to set SDK's at Driven nodes current values :param drivenCtls: List of String names of the Driven Ctls. :type drivenCtls: list :param keyChannels: List of Channels to Key :type keyChannels: list :param driver: Driver Node :type driver: PyNode :param driverAtt: Driver Attr given as a string :type driverAtt: str :param inTanType: Tangent type, by default is linear. :type inTanType: str / optional :param outTanType: Tangent type, by default is linear. :type outTanType: str / optional :param zeroKey: if True, will set a zero key before

setting the key at current value.

Returns
n/a

`mgear.rigbits.sdk_manager.core.`**`mirror_SDK`**`(`*driverCtl*`)`

Takes in a driver control and extrapolates out all the other information needed to mirror it's connected SDK's.

Parameters
`driverCtl` (*PyNode*) –

Returns
None

`mgear.rigbits.sdk_manager.core.`**`next_biggest`**`(`*target*, *in_list*`)`

Returns the next highest number in the in_list. If target is greater the the last number in in_list, will return the last item in the list.

`mgear.rigbits.sdk_manager.core.`**`next_smallest`**`(`*target*, *in_list*`)`

Returns the next Lowest number in the in_list. If target is smaller the the last number in in_list, will return the first item in the list.

`mgear.rigbits.sdk_manager.core.`**`prune_DK_nodes`**`(`*white_list=[]*`)`

Finds all the driven key nodes that have no input or output connected and removes them. Nodes with the word profile in are excluded so that no guide components are broken.

Parameters
`white_list` (*`list / optional`*) – List of nodes to ignore

Returns
list (All the names of the deleted nodes)

`mgear.rigbits.sdk_manager.core.`**`reset_to_default`**`(`*mode*, *clear_sel=False*`)`

Reset All the Rig Driver Ctls or Anim Ctls to Default

Parameters
`mode` (*`str`*) – All Ctl Curves - drv : Driver Ctls - anim : Anim Ctls

Returns
None

`mgear.rigbits.sdk_manager.core.`**`select_all`**`(`*mode*`)`

Select all the Driver Ctls, Anim Ctls Joints or SDK Nodes in the scene.

Parameters
`mode` (*`str`*) – Driver Ctls - anim : Anim Ctls - jnts : Joints - nodes : SDK Nodes

Returns
None

`mgear.rigbits.sdk_manager.core.`**`set_driven_key`**`(`*driverAttr*, *drivenAttr*, *driverVal*, *drivenVal*, *preInfinity=0*, *postInfinity=0*, *inTanType='linear'*, *outTanType='linear'*`)`

Convinience function to aid in setting driven keys.

Parameters

- **`driverAttr`** (*`PyNode.attribute`*) – Driver.attr to drive the SDK
- **`drivenAttr`** (*`PyNode.attribute`*) – Driven.attr to be driven by the SDK
- **`driverVal`** (*`float`*) – Value to use for driver
- **`drivenVal`** (*`float`*) – Value to use for driven

- **preInfinity** (*int*) – IndexKey - constant[0], linear[1], cycle[2], cycleOffset[3], Oscillate[4]
- **postInfinity** (*int*) – IndexKey - constant[0], linear[1], cycle[2], cycleOffset[3], Oscillate[4]
- **inTanType** (*str*) – spline, linear, fast, slow, flat, stepped, step next, fixed, clampedand plateau
- **outTanType** (*str*) – spline, linear, fast, slow, flat, stepped, step next, fixed, clampedand plateau

Returns
new Anim UU node or the Edited one.

TO DO:
fix the return.

`mgear.rigbits.sdk_manager.core.`**`set_limits_from_current`**(*axis*, *controls=None*, *upperLimit=False*, *lowwerLimit=False*)

Sets either the upper or lowwer limits on the provided control and axis

> get current limits > update either the lower or the upper > set limits to Enabled.

There is a lot of duplicate code but its a bit unavoidable with how transformLimits flags are set up.

Aruments:
axis (str): x,y,z axis to use. controls (list): List of PyNodes to iterate over upperLimit (bool): If True will set the upper Limit lowwerLimit (bool): If True will set the lowwer Limit

`mgear.rigbits.sdk_manager.core.`**`set_zero_key`**(*drivenCtls*, *keyChannels*, *driver*, *driverAtt*, *inTanType='linear'*, *outTanType='linear'*)

Takes a Current "state", Sets a ZERO SDK then resets to the "state".

Parameters

- **drivenCtls** (*list*) – List of String names of the Driven Ctls.
- **keyChannels** (*list*) – List of Channels to Key
- **driver** (*PyNode*) – Driver Node
- **driverAtt** (*str*) – Driver Attr given as a string
- **inTanType** (*str / optional*) – Tangent type, by default is linear.
- **outTanType** (*str / optional*) – Tangent type, by default is linear.

Returns
n/a

`mgear.rigbits.sdk_manager.core.`**`toggle_limits`**(*axis*, *controls=None*)

Toggles the controller translate Limits On or Off from their current values, both upper and lower.

Aruments:
axis (str): x,y,z axis to use. controls (list / optional): List of PyNodes to iterate over

If None, use Selection

Module contents

Submodules

mgear.rigbits.blendShapes module

Rigbits blendshapes utilities and tools

`mgear.rigbits.blendShapes.`**`blendshape_foc`**`(`*deformed_obj*`)`

Move existing blendshape node to the front of chain

Parameters

`deformed_obj` (*`PyNode`*) – object with deformation history including a blendshape node

`mgear.rigbits.blendShapes.`**`connectWithBlendshape`**`(`*mesh*, *bst*, *wgt=1.0*, *ffoc=False*`)`

Connect 2 geometries using blendshape node

Parameters

- **`mesh`** (*`PyNode`*) – The Object to apply the blendshape target
- **`bst`** (*`PyNode`*) – The Blendshape target
- **`wgt`** (*`float, optional`*) – Description
- **`ffoc`** (*`bool, optional`*) – Force Front of Chain. will move the blendshape node after creation

Returns

The blenshape node

Return type

PyNode

`mgear.rigbits.blendShapes.`**`connectWithMorph`**`(`*mesh*, *bst*, *wgt=1.0*, *ffoc=True*`)`

Connect 2 geometries using morph node

Parameters

- **`mesh`** (*`PyNode`*) – The Object to apply the blendshape target
- **`bst`** (*`PyNode`*) – The Blendshape target
- **`wgt`** (*`float, optional`*) – envelope weight
- **`ffoc`** (*`bool, optional`*) – Force Front of Chain. will move the morph node after creation

Returns

The blenshape node

Return type

PyNode

`mgear.rigbits.blendShapes.`**`getBlendShape`**`(`*obj*, *lv=2*`)`

Get the blendshape node of an object.

Parameters

- **`obj`** (*`PyNode`*) – The object with the blendshape node
- **`lv`** (*`int, optional`*) – Levels deep to traverse

Returns

The blendshape node

Return type
PyNode

`mgear.rigbits.blendShapes.`**`getDeformerNode`**(*obj*, *lv=2*, *dtype='blendShape'*)

Get the blendshape node of an object.

Parameters
obj – The object with the blendshape node

Returns
The blendshape node

Return type
PyNode

`mgear.rigbits.blendShapes.`**`getMorph`**(*obj*, *lv=2*)

Get the morph node of an object.

Parameters

- **obj** (*PyNode*) – The object with the blendshape node
- **lv** (*int, optional*) – Levels deep to traverse

Returns
The blendshape node

Return type
PyNode

`mgear.rigbits.blendShapes.`**`morph_foc`**(*deformed_obj*, *morph_deformer*)

Move existing morph node to the front of chain

Parameters
deformed_obj (*PyNode*) – object with deformation history including a morph node

mgear.rigbits.channelWrangler module

mgear.rigbits.channelWranglerUI module

mgear.rigbits.cycleTweaks module

Cycle Tweaks module

This module content the tools and procedures to rig tweaks with a benigne cycle

`mgear.rigbits.cycleTweaks.`**`cycleTweak`**(*name*, *edgePair*, *mirrorAxis*, *baseMesh*, *rotMesh*, *transMesh*, *setupParent*, *ctlParent*, *jntOrg=None*, *grp=None*, *iconType='square'*, *size=0.025*, *color=13*, *ro=pymel.core.datatypes.Vector*)

The command to create a cycle tweak.

A cycle tweak is a tweak that cycles to the parent position but doesn't create a cycle of dependency. This type of tweaks are very useful to create facial tweakers.

Parameters

- **name** (*string*) – Name for the cycle tweak
- **edgePair** (*list*) – List of edge pair to attach the cycle tweak
- **mirrorAxis** (*bool*) – If true, will mirror the x axis behaviour.

- **baseMesh** (*Mesh*) – The base mesh for the cycle tweak.
- **rotMesh** (*Mesh*) – The mesh that will support the rotation transformations for the cycle tweak
- **transMesh** (*Mesh*) – The mesh that will support the translation and scale transformations for the cycle tweak
- **setupParent** (*dagNode*) – The parent for the setup objects
- **ctlParent** (*dagNode*) – The parent for the control objects
- **jntOrg** (*None or dagNode, optional*) – The parent for the joints
- **grp** (*None or set, optional*) – The set to add the controls
- **iconType** (*str, optional*) – The controls shape
- **size** (*float, optional*) – The control size
- **color** (*int, optional*) – The control color
- **ro** (*TYPE, optional*) – The control shape rotation offset

Returns
the tweak control and the list of related joints.

Return type
multi

`mgear.rigbits.cycleTweaks.`**`initCycleTweakBase`**(*outMesh*, *baseMesh*, *rotMesh*, *transMesh*, *staticJnt=None*)

Initialice the cycle tweak setup structure

Parameters

- **outMesh** (*Mesh*) – The output mesh after the tweak deformation
- **baseMesh** (*Mesh*) – The base mesh for the cycle tweak.
- **rotMesh** (*Mesh*) – The mesh that will support the rotation transformations for the cycle tweak
- **transMesh** (*Mesh*) – The mesh that will support the translation and scale transformations for the cycle tweak
- **staticJnt** (*None or joint, optional*) – The static joint for the static vertex.

`mgear.rigbits.cycleTweaks.`**`inverseTranslateParent`**(*obj*)

Invert the parent transformation

Parameters
obj (*dagNode*) – The source dagNode to inver parent transformation.

mgear.rigbits.ghost module

Rigbits Ghost module

Helper tools to create layered controls rigs

`mgear.rigbits.ghost.connect_matching_attrs`(*driver*, *driven*, *attr_list=['compRoot', 'uiHost_cnx']*)

Connect matching attributes from driven to driver node if they exist and have inputs.

Parameters

- **driver** (*pm.PyNode*) – The driver PyNode transform.
- **driven** (*pm.PyNode*) – The driven PyNode transform.
- **attr_list** (*list*) – List of attribute names to check and connect. Default is ["compRoot", "uiHost_cnx"].

Returns
None

`mgear.rigbits.ghost.createDoritoGhostCtl`(*ctl*, *parent=None*)

Create a duplicated Ghost control for doritos Create a duplicate of the dorito/tweak and rename the original with _ghost. Later connect the local transforms and the Channels. This is useful to connect local rig controls with the final rig control.

Parameters

- **ctl** (*dagNode*) – Original Control to duplicate
- **parent** (*dagNode*) – Parent for the new created control

`mgear.rigbits.ghost.createGhostCtl`(*ctl*, *parent=None*, *connect=True*)

Create a duplicated Ghost control

Create a duplicate of the control and rename the original with _ghost. Later connect the local transforms and the Channels. This is useful to connect local rig controls with the final rig control.

Parameters

- **ctl** (*dagNode*) – Original Control to duplicate
- **parent** (*dagNode*) – Parent for the new created control

Returns
The new created control

Return type
pyNode

`mgear.rigbits.ghost.ghostSlider`(*ghostControls*, *surface*, *sliderParent*)

Modify the ghost control behaviour to slide on top of a surface

Parameters

- **ghostControls** (*dagNode*) – The ghost control
- **surface** (*Surface*) – The NURBS surface
- **sliderParent** (*dagNode*) – The parent for the slider.

mgear.rigbits.menu module

mgear.rigbits.menu.**cCtl_sub**(*parent_menu_id*)

Create control as child of selected elements

Parameters

parent_menu_id (*stro*) – Parent menu. i.e: "MayaWindow|mGear|menuItem355"

mgear.rigbits.menu.**connect_submenu**(*parent_menu_id*)

Create the connect local Scale, rotation and translation submenu

Parameters

parent_menu_id (*str*) – Parent menu. i.e: "MayaWindow|mGear|menuItem355"

mgear.rigbits.menu.**gimmick_submenu**(*parent_menu_id*)

Create the gimmick joint submenu

Parameters

parent_menu_id (*str*) – Parent menu. i.e: "MayaWindow|mGear|menuItem355"

mgear.rigbits.menu.**install**()

Install Rigbits submenu

mgear.rigbits.menu.**install_utils_menu**(*m*)

Install rigbit utils submenu

mgear.rigbits.menu.**pCtl_sub**(*parent_menu_id*)

Create control as parent of selected elements

Parameters

parent_menu_id (*stro*) – Parent menu. i.e: "MayaWindow|mGear|menuItem355"

mgear.rigbits.mirror_controls module

class mgear.rigbits.mirror_controls.**MirrorController**

Bases: object

static copy_and_prepare_source(*source*)

static get_opposite_control(*node*)

static get_specific_side_controls(*side='L'*)

mirror_left_to_right()

mirror_pairs(*pairs*)

mirror_right_to_left()

mirror_selection()

class mgear.rigbits.mirror_controls.**MirrorControlsUi**(**args: Any*, ***kwargs: Any*)

Bases: MayaQWidgetDockableMixin, QDialog

mirror_button_pressed()

mgear.rigbits.mirror_controls.**show**(**args*)

mgear.rigbits.postSpring module

Post Spring tool

creates a spring dynamic rig on top of a pre-existing FK chain rig.

`mgear.rigbits.postSpring.`**`bake_spring`**`(`**args*`)`

Shortcut fro the Maya's Bake Simulation Options

`mgear.rigbits.postSpring.`**`build_spring`**`(`**args*`)`

`mgear.rigbits.postSpring.`**`postSpring`**`(`*dist=5*, *hostUI=False*, *hostUI2=False*, *invertX=False*`)`

Create the dynamic spring rig.

This spring system use the mgear_spring node And transfer the position spring to rotation spring using an aim constraint.

Note: The selected chain of object should be align with the X axis.

Parameters

- **dist** (*float*) – The distance of the position spring.
- **hostUI** (*dagNode*) – The spring active and intensity channel host.
- **hostUI2** (*dagNode*) – The daping and stiffness channel host for each object in the chain.
- **invertX** (*bool*) – reverse the direction of the x axis.

`mgear.rigbits.postSpring.`**`spring_UI`**`(`**args*`)`

Creates the post tool UI

mgear.rigbits.proxySlicer module

Rigbits proxy mesh slicer

`mgear.rigbits.proxySlicer.`**`slice`**`(`*parent=False*, *oSel=False*, **args*`)`

Create a proxy geometry from a skinned object

mgear.rigbits.rbf_io module

Handles the import and exporting of all supported RBF node types

`mgear.rigbits.rbf_io.`**`RBF_FILE_EXTENSION`**

extention of the serialized json data

Type
str

`mgear.rigbits.rbf_io.`**`RBF_MODULES`**

nodeType: module api, normalized to fit the rbfManager

Type
Dict

__author__ = "Rafael Villar" __email__ = "rav@ravrigs.com" mgear.rigbits.rbf_io.**exportRBFs**(*nodes*, *filePath*)

exports the desired rbf nodes to the filepath provided

Parameters

- **nodes** (*list*) – of rbfnodes
- **filePath** (*str*) – filepath to json

mgear.rigbits.rbf_io.**fileDialog**(*startDir=None*, *mode=0*)

prompt dialog for either import/export from a UI

Parameters

- **startDir** (*str*) – A directory to start from
- **mode** (*int, optional*) – import or export, 0/1

Returns
path selected by user

Return type
str

mgear.rigbits.rbf_io.**importRBFs**(*filePath*)

import rbfs from file, using the assoiciated module type to recreate

Parameters
filePath (*str*) – filepath to json

Returns
n/a

Return type
n/a

mgear.rigbits.rbf_manager_ui module

mgear.rigbits.rbf_node module

mgear.rigbits.rivet module

Rigbits rivet creator

class mgear.rigbits.rivet.**rivet**

Bases: object

Create a rivet

Thanks to http://jinglezzz.tumblr.com for the tutorial :)

create(*mesh*, *edge1*, *edge2*, *parent*, *name=None*)

createConnections(**args*)

createNodes(**args*)

setAttributes()

mgear.rigbits.rope module

Rigbits Rope rig creator

`mgear.rigbits.rope.`**`build_rope`**(**args*)

`mgear.rigbits.rope.`**`rope`**(*DEF_nb=10*, *ropeName='rope'*, *keepRatio=False*, *lvlType='transform'*, *oSel=None*)

Create rope rig based in 2 parallel curves.

Parameters

- **DEF_nb** (*int*) – Number of deformer joints.
- **ropeName** (*str*) – Name for the rope rig.
- **keepRatio** (*bool*) – If True, the deformers will keep the length position when the curve is stretched.

`mgear.rigbits.rope.`**`rope_UI`**(**args*)

Rope tool UI

mgear.rigbits.sdk_io module

Rigbits, SDK i/o

exportSDKs([“drivenNodeA”, “drivenNodeB”], “path/to/desired/output.json”) importSDKs(path/to/desired/output.json)

MIRRORING ——- # copy from source, say left, to target, right copySDKsToNode(“jacketFlap_L1_fk0_sdk”,

“neck_C0_0_jnt”, “jacketFlap_R1_fk0_sdk”)

invert/mirror the attributes necessary for the other side, # in this case it is the following attributes mirrorSDKkeys(“jacketFlap_R1_fk0_sdk”,

attributes=[“rotateZ”], invertDriver=True, invertDriven=False)

mirrorSDKkeys(“jacketFlap_R1_fk0_sdk”,
attributes=[“translateX”, “translateY”], invertDriver=True, invertDriven=True)

in this other instance, it was the same copySDKsToNode(“jacketFlap_L0_fk0_sdk”,

“neck_C0_0_jnt”, “jacketFlap_R0_fk0_sdk”)

`mgear.rigbits.sdk_io.`**`SDK_ANIMCURVES_TYPE`**

sdk anim curves to support

Type
list

`mgear.rigbits.sdk_io.`**`copySDKsToNode`**(*sourceDriven*, *targetDriver*, *targetDriven*, *sourceAttributes=[]*, *sourceDriverFilter=None*)

Duplicates sdk nodes from the source drive, to any designated target driver/driven

Parameters

- **sourceDriven** (*pynode*) – source to copy from
- **targetDriver** (*pynode*) – to drive the new sdk node
- **targetDriven** (*pynode*) – node to be driven
- **sourceAttributes** (*list, optional*) – of attrs to copy, if none provided

- **all** (*assume*) –
- **sourceDriverFilter** (*list, pynode*) – Driver transforms to filter by,
- **returned.** (*if the connected SDK is not driven by this node it will not be*) –

Returns
n/a

Return type
TYPE

`mgear.rigbits.sdk_io.`**createSDKFromDict**(*sdkInfo_dict*)

Create a sdk node from the provided info dict

Parameters
sdkInfo_dict (*dict*) – dict of node information to create

Returns
created sdk node

Return type
PyNode

`mgear.rigbits.sdk_io.`**exportSDKs**(*nodes*, *filePath*)

exports the sdk information based on the provided nodes to a json file

Parameters

- **nodes** (*list*) – of nodes to export
- **filePath** (*string*) – full filepath to export jsons to

`mgear.rigbits.sdk_io.`**getAllSDKInfoFromNode**(*node*)

returns a dict for all of the connected sdk/animCurve on the provided node

Parameters
node (*pynode*) – name of node to the be searched

Returns
of all of the sdk nodes

Return type
dict

`mgear.rigbits.sdk_io.`**getBlendNodes**(*attrPlug*)

Check the attrPlug (node.attr) provided for any existing connections if blendWeighted exists, return the appropriate input[#], if sdk, create a blendweighted and connect sdk, return input[#]

Parameters
attrPlug (*string*) – node.attr

Returns
node.attr of the blendweighted node that was just created or existing

Return type
string

`mgear.rigbits.sdk_io.`**getConnectedSDKs**(*driven*, *curvesOfType=[]*, *sourceDriverFilter=None*)

get all the sdk, animcurve, nodes/plugs connected to the provided node.

Parameters

- **node** (*str, pynode*) – name of node, or pynode
- **curvesOfType** (*list, optional*) – animCurve nodes of type if none provided
- **set.** (*will fall back on module defined supported*) –
- **sourceDriverFilter** (*list, pynode*) – Driver transforms to filter by,
- **returned.** (*if the connected SDK is not driven by this node it will not be*) –

Returns
of sdk nodes, paired with the node/attr they effect

Return type
list

mgear.rigbits.sdk_io.**getMultiDriverSDKs**(*driven*, *sourceDriverFilter=None*)

get the sdk nodes that are added through a blendweighted node

Parameters

- **driven** (*string*) – name of the driven node
- **sourceDriverFilter** (*list, pynode*) – Driver transforms to filter by,
- **returned.** (*if the connected SDK is not driven by this node it will not be*) –

Returns
of sdk nodes

Return type
list

mgear.rigbits.sdk_io.**getPynodes**(*nodes*)

Conevenience function to allow uses to pass in strings, but convert to pynodes if not already.

Parameters
nodes (*list*) – string names

Returns
of pynodes

Return type
list

mgear.rigbits.sdk_io.**getSDKDestination**(*animNodeOutputPlug*)

Get the final destination of the sdk node, skips blendweighted and conversion node to get the transform node. TODO: Open this up to provided type destination

Parameters
animNodeOutputPlug (*string*) – animationNode.output

Returns
name of the node, and attr

Return type
list

mgear.rigbits.sdk_io.**getSDKInfo**(*animNode*)

get all the information from an sdk/animCurve in a dictioanry for exporting.

Parameters
animNode (*pynode*) – name of node, pynode

Returns
dictionary of all the attrs to be exported

Return type
dict

`mgear.rigbits.sdk_io.`**`importSDKs`**`(`*filePath*`)`

create sdk nodes from json file, connected to drivers and driven

Parameters
filePath (*string*) – path to json file

`mgear.rigbits.sdk_io.`**`invertKeyValues`**`(`*newKeyNode*, *invertDriver=True*, *invertDriven=True*`)`

Mirror keyframe node procedure, in case you need to flip your SDK's.

Parameters

- **newKeyNode** (*PyNode*) – sdk node to invert values on
- **invertDriver** (*bool, optional*) – should the drivers values be inverted
- **invertDriven** (*bool, optional*) – should the drivens values be inverted

`mgear.rigbits.sdk_io.`**`mirrorSDKkeys`**`(`*node*, *attributes=[]*, *invertDriver=True*, *invertDriven=True*`)`

mirror/invert the values on the specified node and attrs, get the sdks and invert those values

Parameters

- **node** (*pynode*) – node being driven to have its sdk values inverted
- **attributes** (*list, optional*) – attrs to be inverted
- **invertDriver** (*bool, optional*) – should the driver, "time" values
- **inverted** (*be*) –
- **invertDriven** (*bool, optional*) – should the driven, "value" values
- **inverted** –

`mgear.rigbits.sdk_io.`**`removeSDKs`**`(`*node*, *attributes=[]*, *sourceDriverFilter=None*`)`

Convenience function to remove, delete, all sdk nodes associated with the provided node

Parameters

- **node** (*pynode*) – name of the node
- **attributes** (*list, optional*) – list of attributes to remove sdks from
- **all** (*if none provided, assume*) –
- **sourceDriverFilter** (*list, pynode*) – Driver transforms to filter by,
- **returned.** (*if the connected SDK is not driven by this node it will not be*) –

`mgear.rigbits.sdk_io.`**`stripKeys`**`(`*animNode*`)`

remove animation keys from the provided sdk node

Parameters
animNode (*pynode*) – sdk/anim node

mgear.rigbits.six module

Utilities for writing code that runs on Python 2 and 3

class mgear.rigbits.six.**Module_six_moves_urllib**(*name*, *doc=None*)

Bases: `module`

Create a six.moves.urllib namespace that resembles the Python 3 namespace

error = <module 'mgear.rigbits.six.moves.urllib.error'>

parse = <module 'mgear.rigbits.six.moves.urllib_parse'>

request = <module 'mgear.rigbits.six.moves.urllib.request'>

response = <module 'mgear.rigbits.six.moves.urllib.response'>

robotparser = <module 'mgear.rigbits.six.moves.urllib.robotparser'>

class mgear.rigbits.six.**Module_six_moves_urllib_error**(*name*)

Bases: `_LazyModule`

Lazy loading of moved objects in six.moves.urllib_error

ContentTooShortError

HTTPError

URLError

class mgear.rigbits.six.**Module_six_moves_urllib_parse**(*name*)

Bases: `_LazyModule`

Lazy loading of moved objects in six.moves.urllib_parse

ParseResult

SplitResult

parse_qs

parse_qsl

quote

quote_plus

splitquery

splittag

splituser

splitvalue

unquote

unquote_plus

unquote_to_bytes

urldefrag

urlencode

urljoin

urlparse

urlsplit

urlunparse

urlunsplit

uses_fragment

uses_netloc

uses_params

uses_query

uses_relative

class mgear.rigbits.six.**Module_six_moves_urllib_request**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_request

AbstractBasicAuthHandler

AbstractDigestAuthHandler

BaseHandler

CacheFTPHandler

FTPHandler

FancyURLopener

FileHandler

HTTPBasicAuthHandler

HTTPCookieProcessor

HTTPDefaultErrorHandler

HTTPDigestAuthHandler

HTTPErrorProcessor

HTTPHandler

HTTPPasswordMgr

HTTPPasswordMgrWithDefaultRealm

HTTPRedirectHandler

HTTPSHandler

OpenerDirector

ProxyBasicAuthHandler

ProxyDigestAuthHandler

ProxyHandler

Request

URLopener

UnknownHandler

build_opener

getproxies

install_opener

parse_http_list

parse_keqv_list

pathname2url

proxy_bypass

url2pathname

urlcleanup

urlopen

urlretrieve

class mgear.rigbits.six.**Module_six_moves_urllib_response**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_response

addbase

addclosehook

addinfo

addinfourl

class mgear.rigbits.six.**Module_six_moves_urllib_robotparser**(*name*)

Bases: _LazyModule

Lazy loading of moved objects in six.moves.urllib_robotparser

RobotFileParser

class mgear.rigbits.six.**MovedAttribute**(*name*, *old_mod*, *new_mod*, *old_attr=None*, *new_attr=None*)

Bases: _LazyDescr

class `mgear.rigbits.six.MovedModule`(*name*, *old*, *new=None*)

Bases: `_LazyDescr`

`mgear.rigbits.six.add_metaclass`(*metaclass*)

Class decorator for creating a class with a metaclass.

`mgear.rigbits.six.add_move`(*move*)

Add an item to six.moves.

`mgear.rigbits.six.assertCountEqual`(*self*, **args*, ***kwargs*)

`mgear.rigbits.six.assertNotRegex`(*self*, **args*, ***kwargs*)

`mgear.rigbits.six.assertRaisesRegex`(*self*, **args*, ***kwargs*)

`mgear.rigbits.six.assertRegex`(*self*, **args*, ***kwargs*)

`mgear.rigbits.six.b`(*s*)

Byte literal

`mgear.rigbits.six.create_unbound_method`(*func*, *cls*)

`mgear.rigbits.six.ensure_binary`(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce **s** to six.binary_type.

For Python 2:

- *unicode* -> encoded to *str*
- *str* -> *str*

For Python 3:

- *str* -> encoded to *bytes*
- *bytes* -> *bytes*

`mgear.rigbits.six.ensure_str`(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce *s* to *str*.

For Python 2:

- *unicode* -> encoded to *str*
- *str* -> *str*

For Python 3:

- *str* -> *str*
- *bytes* -> decoded to *str*

`mgear.rigbits.six.ensure_text`(*s*, *encoding='utf-8'*, *errors='strict'*)

Coerce *s* to six.text_type.

For Python 2:

- *unicode* -> *unicode*
- *str* -> *unicode*

For Python 3:

- *str* -> *str*
- *bytes* -> decoded to *str*

`mgear.rigbits.six.get_unbound_function(unbound)`

Get the function out of a possibly unbound function

`mgear.rigbits.six.int2byte()`

S.pack(v1, v2, …) -> bytes

Return a bytes object containing values v1, v2, … packed according to the format string S.format. See help(struct) for more on format strings.

`mgear.rigbits.six.iteritems(d, **kw)`

Return an iterator over the (key, value) pairs of a dictionary.

`mgear.rigbits.six.iterkeys(d, **kw)`

Return an iterator over the keys of a dictionary.

`mgear.rigbits.six.iterlists(d, **kw)`

Return an iterator over the (key, [values]) pairs of a dictionary.

`mgear.rigbits.six.itervalues(d, **kw)`

Return an iterator over the values of a dictionary.

`mgear.rigbits.six.python_2_unicode_compatible(klass)`

A class decorator that defines __unicode__ and __str__ methods under Python 2. Under Python 3 it does nothing.

To support Python 2 and 3 with a single code base, define a __str__ method returning text and apply this decorator to the class.

`mgear.rigbits.six.raise_from(value, from_value)`

`mgear.rigbits.six.remove_move(name)`

Remove item from six.moves.

`mgear.rigbits.six.reraise(tp, value, tb=None)`

Reraise an exception.

`mgear.rigbits.six.u(s)`

Text literal

`mgear.rigbits.six.with_metaclass(meta, *bases)`

Create a base class with a metaclass.

mgear.rigbits.tweaks module

Rigbits tweaks rig module

`mgear.rigbits.tweaks.createJntTweak(mesh, jntParent, ctlParent)`

Create a joint tweak

Parameters

- **mesh** (*mesh*) – The object to deform with the tweak
- **jntParent** (*dagNode*) – The parent for the new joint
- **ctlParent** (*dagNode*) – The parent for the control.

mgear.rigbits.tweaks.**createMirrorRivetTweak**(*mesh*, *edgePair*, *name*, *parent=None*, *ctlParent=None*, *jntParent=None*, *color=[0, 0, 0]*, *size=0.04*, *defSet=None*, *ctlSet=None*, *side=None*, *gearMulMatrix=True*, *attach_rot=False*, *inputMesh=None*, *ctlShape='sphere'*)

Create a tweak joint attached to the mesh using a rivet. The edge pair will be used to find the mirror position on the mesh

Parameters

- **mesh** (*mesh*) – The object to add the tweak
- **edgePair** (*pair list*) – The edge pair to create the rivet
- **name** (*str*) – The name for the tweak
- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls
- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node

Returns
The tweak control

Return type
PyNode

mgear.rigbits.tweaks.**createRivetTweak**(*mesh*, *edgePair*, *name*, *parent=None*, *ctlParent=None*, *jntParent=None*, *color=[0, 0, 0]*, *size=0.04*, *defSet=None*, *ctlSet=None*, *side=None*, *gearMulMatrix=True*, *attach_rot=False*, *inputMesh=None*, *ctlShape='sphere'*)

Create a tweak joint attached to the mesh using a rivet

Parameters

- **mesh** (*mesh*) – The object to add the tweak
- **edgePair** (*pair list*) – The edge pair to create the rivet
- **name** (*str*) – The name for the tweak
- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls

- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node

Returns
The tweak control

Return type
PyNode

`mgear.rigbits.tweaks.createRivetTweakFromList(mesh, edgePairList, name, parent=None, ctlParent=None, jntParent=None, color=[0, 0, 0], size=0.04, defSet=None, ctlSet=None, side=None, mirror=False, mParent=None, mCtlParent=None, mjntParent=None, mColor=None, gearMulMatrix=True, attach_rot=False, inputMesh=None, ctlShape='sphere')`

Create multiple rivet tweaks from a list of edge pairs

Parameters

- **mesh** (*mesh*) – The object to add the tweak
- **edgePairList** (*list of list*) – The edge pair list of list
- **name** (*str*) – The name for the tweak
- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls
- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **mirror** (*bool, optional*) – Create the mirror tweak on X axis symmetry
- **mParent** (*None, optional*) – Mirror tweak parent, if None will use parent arg
- **mjntParent** (*None, optional*) – Mirror parent joint, if None will use jntParent arg
- **mCtlParent** (*None, optional*) – Mirror ctl parent, if None will use ctlParent arg
- **mColor** (*None, optional*) – Mirror controls color, if None will color arg
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node

Returns
Description

Return type
TYPE

mgear.rigbits.tweaks.**createRivetTweakLayer**(*layerMesh*, *bst*, *edgePairList*, *name*, *parent=None*, *ctlParent=None*, *jntParent=None*, *color=[0, 0, 0]*, *size=0.04*, *defSet=None*, *ctlSet=None*, *side=None*, *mirror=False*, *mParent=None*, *mCtlParent=None*, *mjntParent=None*, *mColor=None*, *gearMulMatrix=True*, *static_jnt=None*, *attach_rot=False*, *inputMesh=None*, *ctlShape='sphere'*)

Create a rivet tweak layer setup

Parameters

- **layerMesh** (*mesh*) – The tweak layer mesh
- **bst** (*mesh*) – The mesh blendshape target
- **edgePairList** (*list of list*) – The edge pair list of list
- **name** (*str*) – The name for the tweak
- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls
- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **mirror** (*bool, optional*) – Create the mirror tweak on X axis symmetry
- **mParent** (*None, optional*) – Mirror tweak parent, if None will use parent arg
- **mjntParent** (*None, optional*) – Mirror parent joint, if None will use jntParent arg
- **mCtlParent** (*None, optional*) – Mirror ctl parent, if None will use ctlParent arg
- **mColor** (*None, optional*) – Mirror controls color, if None will color arg
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node
- **static_jnt** (*dagNode, optional*) – Static joint for the setup

mgear.rigbits.tweaks.**create_mirror_proximity_tweak**(*mesh*, *edgePair*, *name*, *parent=None*, *ctlParent=None*, *jntParent=None*, *color=[0, 0, 0]*, *size=0.04*, *defSet=None*, *ctlSet=None*, *side=None*, *gearMulMatrix=True*, *attach_rot=False*, *inputMesh=None*, *ctlShape='sphere'*, *existing_pin=None*)

Create a tweak joint attached to the mesh using a proximity pin. The edge pair will be used to find the mirror position on the mesh

Parameters

- **mesh** (*mesh*) – The object to add the tweak
- **edgePair** (*pair list*) – The edge pair to create the rivet
- **name** (*str*) – The name for the tweak

- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls
- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node
- **attach_rot** (*bool, optional*) – Description
- **inputMesh** (*None, optional*) – Description
- **ctlShape** (*str, optional*) – Description
- **existing_pin** (*None, optional*) – Description

Returns
The tweak control and pin node

Return type
PyNode

mgear.rigbits.tweaks.**create_proximity_tweak**(*mesh*, *edgePair*, *name*, *parent=None*, *ctlParent=None*, *jntParent=None*, *color=[0, 0, 0]*, *size=0.04*, *defSet=None*, *ctlSet=None*, *side=None*, *gearMulMatrix=True*, *attach_rot=False*, *inputMesh=None*, *ctlShape='sphere'*, *existing_pin=None*)

Create a tweak joint attached to the mesh using a proximity pin

Parameters

- **mesh** (*mesh*) – The object to add the tweak
- **edgePair** (*pair list*) – The edge pair to create the proximity pin
- **name** (*str*) – The name for the tweak
- **parent** (*None or dagNode, optional*) – The parent for the tweak
- **ctlParent** (*None or dagNode, optional*) – The parent for the tweak control
- **jntParent** (*None or dagNode, optional*) – The parent for the joints
- **color** (*list, optional*) – The color for the control
- **size** (*float, optional*) – Size of the control
- **defSet** (*None or set, optional*) – Deformer set to add the joints
- **ctlSet** (*None or set, optional*) – the set to add the controls
- **side** (*None, str*) – String to set the side. Valid values are L, R or C. If the side is not set or the value is not valid, the side will be set automatically based on the world position
- **gearMulMatrix** (*bool, optional*) – If False will use Maya default multiply matrix node
- **attach_rot** (*bool, optional*) – Description

- **inputMesh** (*None, optional*) – Description
- **ctlShape** (*str, optional*) – Description
- **existing_pin** (*None, optional*) – Description

Returns
o_icon and pin node

Return type
list

mgear.rigbits.tweaks.**edgePairList**(*log=True*)

Print and return a list of edge pairs to be use with createRivetTweakLayer and createRivetTweakFromList

Returns
list of edge pairs

Return type
list

mgear.rigbits.tweaks.**negateTransformConnection**(*in_rot*, *out_rot*, *neg_axis=[-1, -1, 1]*)

mgear.rigbits.tweaks.**pre_bind_matrix_connect**(*mesh*, *joint*, *jointBase*)

Connect the pre bind matrix of the skin cluseter to the joint parent. This create the offset in the deformation to avoid double transformation

Parameters

- **mesh** (*PyNode*) – Mesh object with the tweak skin cluster
- **joint** (*PyNode*) – Tweak joint
- **jointBase** (*PyNode*) – Tweak joint parent

mgear.rigbits.tweaks.**resetJntLocalSRT**(*jnt*)

Reset the local SRT and jointOrient of a joint

Parameters
jnt (*joint*) – The joint to reset the local SRT

mgear.rigbits.utils module

Rigbits utilitie tools

mgear.rigbits.utils.**createHotkeys**(**args*)

Create mGear custom hotkey functions ready to be use.

This command doesn't set the hotkey binding. Only create the functions.

Parameters
***args** – Maya's dummy

mgear.rigbits.utils.**createRunTimeCommand**(*name*, *rCmd*, *ann=''*)

Create run time commands from raw string.

This function is used to create the mGear hotkeys.

mgear.rigbits.version module

mgear.rigbits.weightNode_io module

mgear.rigbits.widgets module

Module contents

`mgear.rigbits.`**`addBlendedJoint`**(*oSel=None*, *compScale=True*, *blend=0.5*, *name=None*, *select=True*, **args*)

Create and gimmick blended joint

Create a joint that rotate 50% of the selected joint. This operation is done using a pairBlend node.

Parameters

- **oSel** (*None or joint, optional*) – If None will use the selected joints.
- **compScale** (*bool, optional*) – Set the compScale option of the blended joint. Default is True.
- **blend** (*float, optional*) – blend rotation value
- **name** (*None, optional*) – Name for the blended o_node
- ***args** – Maya's dummy

Returns
blended joints list

Return type
list

`mgear.rigbits.`**`addJnt`**(*obj=False*, *parent=False*, *noReplace=False*, *grp=None*, *jntName=None*, **args*)

Create one joint for each selected object.

Parameters

- **obj** (*bool or dagNode, optional*) – The object to drive the new joint. If False will use the current selection.
- **parent** (*bool or dagNode, optional*) – The parent for the joint. If False will try to parent to jnt_org. If jnt_org doesn't exist will parent the joint under the obj
- **noReplace** (*bool, optional*) – If True will add the extension "_jnt" to the new joint name
- **grp** (*pyNode or None, optional*) – The set to add the new joint. If none will use "rig_deformers_grp"
- ***args** – Maya's dummy

Returns
The New created joint.

Return type
pyNode

`mgear.rigbits.`**`addNPO`**(*objs=None*, **args*)

Add a transform node as a neutral pose

Add a transform node as a parent and in the same pose of each of the selected objects. This way neutralize the local transfromation values. NPO stands for "neutral position" terminology from the all mighty Softimage ;)

mgear.rigbits.**addSupportJoint**(*oSel=None*, *select=True*, **args*)

Add an extra joint to the blended joint.

This is meant to be use with SDK for game style deformation.

Parameters

- **oSel** (*None or blended joint, optional*) – If None will use the current selection.
- ***args** – Mays's dummy

Returns
blended joints list

Return type
list

mgear.rigbits.**alignToPointsLoop**(*points=None*, *loc=None*, *name=None*, **args*)

Create space locator align to the plain define by at less 3 vertex

Parameters

- **points** (*None or vertex list, optional*) – The reference vertex to align the ref locator
- **loc** (*None or dagNode, optional*) – If none will create a new locator
- **name** (*None or string, optional*) – Name of the new locator
- ***args** – Description

Returns
Description

Return type
TYPE

mgear.rigbits.**connectInvertSRT**(*source*, *target*, *srt='srt'*, *axis='xyz'*)

Connect the locat transformations with inverted values.

Parameters

- **source** (*dagNode*) – The source driver dagNode
- **target** (*dagNode*) – The target driven dagNode
- **srt** (*string, optional*) – String value for the scale(s), rotate(r), translation(t). Default value is "srt". Posible values "s", "r", "t" or any combination
- **axis** (*string, optional*) – String value for the axis. Default value is "xyz". Posible values "x", "y", "z" or any combination

mgear.rigbits.**connectLocalTransform**(*objects=None*, *s=True*, *r=True*, *t=True*, **args*)

Connect scale, rotatio and translation.

Parameters

- **objects** (*None or list of dagNode, optional*) – If None will use the current selection.
- **s** (*bool, optional*) – If True will connect the local scale
- **r** (*bool, optional*) – If True will connect the local rotation
- **t** (*bool, optional*) – If True will connect the local translation

- **`*args`** – Maya's dummy

`mgear.rigbits.`**`connectUserDefinedChannels`**`(`*source*, *targets*`)`

Connects the user defined channels

Connects the user defined channels between 2 objects with the same channels. Usually a copy of the same object.

Parameters

- **`source`** (*`dagNode`*) – The dagNode with the source user defined channels
- **`targets`** (*`list of dagNode`*) – The list of dagNodes with the same user defined channels to be connected.

`mgear.rigbits.`**`connectWorldTransform`**`(`*source*, *target*`)`

Connect the source world transform of one object to another object.

Parameters

- **`source`** (*`dagNode`*) – Source dagNode.
- **`target`** (*`dagNode`*) – target dagNode.

`mgear.rigbits.`**`connect_scale_from_world_matrix`**`(`*driver*, *driven*`)`

Set up node connections to make the driven object scale based on the driver object's world scale matrix. Accepts either str or PyNode.

Parameters

- **`driver`** (*`str or pm.nodetypes.Transform`*) – Name or PyNode of the driver object.
- **`driven`** (*`str or pm.nodetypes.Transform`*) – Name or PyNode of the driven object.

Example

```
>>> connect_scale_from_world_matrix('driver_cube', 'driven_cube')
>>> connect_scale_from_world_matrix(pm.PyNode('driver_cube'),
...                                 pm.PyNode('driven_cube'))
```

`mgear.rigbits.`**`createCTL`**`(`*type='square'*, *child=False*, **args*`)`

Create a control for each selected object.

The newly create control can be parent or child of the object.

Parameters

- **`type`** (*`str`*) – The shape of the control.
- **`child`** (*`bool`*) – if True, the control will be created as a child of the object.

`mgear.rigbits.`**`createInterpolateTransform`**`(`*objects=None*, *blend=0.5*, **args*`)`

Create space locator and apply gear_intmatrix_op, to interpolate the his pose between 2 selected objects.

Parameters

- **`objects`** (*`None or list of 2 dagNode, optional`*) – The 2 dagNode to interpolate the transform.
- **`blend`** (*`float, optional`*) – The interpolation blend factor.
- **`*args`** – Maya's dummy

Returns
The new transformation witht the interpolate matrix o_node applied.

Return type
pyNode

`mgear.rigbits.duplicateSym(*args)`

Duplicate one dag hierarchy to/from X/-X renaming "L" to "R"

`mgear.rigbits.matchPosfromBBox(*args)`

Match the position usin bounding box of another object another.

Match the position of one object, using the boundig box center of another object.

`mgear.rigbits.matchWorldXform(*args)`

Align 2 selected objects in world space

`mgear.rigbits.replaceShape(source=None, targets=None, *args)`

Replace the shape of one object by another.

Parameters

- **source** (*None, PyNode*) – Source object with the original shape.
- **targets** (*None, list of pyNode*) – Targets object to apply the source shape.
- ***args** – Maya's dummy

Returns
Return non if nothing is selected or the source and targets are none

Return type
None

`mgear.rigbits.selectDeformers(*args)`

Select the deformers from the object skinCluster

`mgear.rigbits.spaceJump(ref=None, space=None, *args)`

Space Jump gimmick

This function create a local reference space from another space in the hierarchy

Parameters

- **ref** (*None, optional*) – Transform reference
- **space** (*None, optional*) – Space reference
- ***args** – Maya dummy

Returns
Transform

Return type
pyNode

mgear.shifter package

Subpackages

mgear.shifter.component package

Submodules

mgear.shifter.component.chain_guide_initializer module

mgear.shifter.component.chain_guide_initializer_ui module

mgear.shifter.component.guide module

mgear.shifter.component.joint_names_ui module

mgear.shifter.component.main_settings_ui module

Module contents

Submodules

mgear.shifter.afg_tools module

mgear.shifter.afg_tools_ui module

mgear.shifter.custom_step module

mgear.shifter.custom_step_ui module

mgear.shifter.game_tools_disconnect module

mgear.shifter.game_tools_disconnect_ui module

mgear.shifter.game_tools_fbx module

mgear.shifter.game_tools_fbx_utils module

mgear.shifter.game_tools_fbx_widgets module

mgear.shifter.guide module

mgear.shifter.guide_diff_ui module

mgear.shifter.guide_manager module

mgear.shifter.guide_manager_component module

mgear.shifter.guide_manager_component_ui module

mgear.shifter.guide_manager_gui module

mgear.shifter.guide_template module

mgear.shifter.guide_template_explorer module

mgear.shifter.guide_template_explorer_ui module

mgear.shifter.guide_ui module

mgear.shifter.io module

mgear.shifter.menu module

mgear.shifter.mocap_tools module

mgear.shifter.naming module

mgear.shifter.naming_rules_ui module

mgear.shifter.plebes module

mgear.shifter.relative_guide_placement module

mgear.shifter.version module

Module contents

mgear.shifter_classic_components package

Subpackages

mgear.shifter_classic_components.arm_2jnt_01 package

Submodules

mgear.shifter_classic_components.arm_2jnt_01.guide module

mgear.shifter_classic_components.arm_2jnt_01.settingsUI module

Module contents

mgear.shifter_classic_components.arm_2jnt_02 package

Submodules

mgear.shifter_classic_components.arm_2jnt_02.guide module

mgear.shifter_classic_components.arm_2jnt_02.settingsUI module

Module contents

mgear.shifter_classic_components.arm_2jnt_03 package

Submodules

mgear.shifter_classic_components.arm_2jnt_03.guide module

mgear.shifter_classic_components.arm_2jnt_03.settingsUI module

Module contents

mgear.shifter_classic_components.arm_2jnt_04 package

Submodules

mgear.shifter_classic_components.arm_2jnt_04.guide module

mgear.shifter_classic_components.arm_2jnt_04.settingsUI module

Module contents

mgear.shifter_classic_components.arm_2jnt_freeTangents_01 package

Submodules

mgear.shifter_classic_components.arm_2jnt_freeTangents_01.guide module

mgear.shifter_classic_components.arm_2jnt_freeTangents_01.settingsUI module

Module contents

mgear.shifter_classic_components.arm_ms_2jnt_01 package

Submodules

mgear.shifter_classic_components.arm_ms_2jnt_01.guide module

mgear.shifter_classic_components.arm_ms_2jnt_01.settingsUI module

Module contents

mgear.shifter_classic_components.cable_01 package

Submodules

mgear.shifter_classic_components.cable_01.guide module

mgear.shifter_classic_components.cable_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_01 package

Submodules

mgear.shifter_classic_components.chain_01.guide module

mgear.shifter_classic_components.chain_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_FK_spline_01 package

Submodules

mgear.shifter_classic_components.chain_FK_spline_01.guide module

mgear.shifter_classic_components.chain_FK_spline_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_FK_spline_02 package

Submodules

mgear.shifter_classic_components.chain_FK_spline_02.guide module

mgear.shifter_classic_components.chain_FK_spline_02.settingsUI module

Module contents

mgear.shifter_classic_components.chain_FK_spline_variable_IK_01 package

Submodules

mgear.shifter_classic_components.chain_FK_spline_variable_IK_01.guide module

mgear.shifter_classic_components.chain_FK_spline_variable_IK_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_IK_spline_variable_FK_01 package

Submodules

mgear.shifter_classic_components.chain_IK_spline_variable_FK_01.guide module

mgear.shifter_classic_components.chain_IK_spline_variable_FK_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_IK_spline_variable_FK_stack_01 package

Submodules

mgear.shifter_classic_components.chain_IK_spline_variable_FK_stack_01.guide module

mgear.shifter_classic_components.chain_IK_spline_variable_FK_stack_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_loc_ori_01 package

Submodules

mgear.shifter_classic_components.chain_loc_ori_01.guide module

mgear.shifter_classic_components.chain_loc_ori_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_net_01 package

Submodules

mgear.shifter_classic_components.chain_net_01.guide module

mgear.shifter_classic_components.chain_net_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_spring_01 package

Submodules

mgear.shifter_classic_components.chain_spring_01.guide module

Module contents

mgear.shifter_classic_components.chain_spring_lite_stack_master_01 package

Submodules

mgear.shifter_classic_components.chain_spring_lite_stack_master_01.guide module

Module contents

mgear.shifter_classic_components.chain_stack_01 package

Submodules

mgear.shifter_classic_components.chain_stack_01.guide module

mgear.shifter_classic_components.chain_stack_01.settingsUI module

Module contents

mgear.shifter_classic_components.chain_whip_01 package

Submodules

mgear.shifter_classic_components.chain_whip_01.guide module

mgear.shifter_classic_components.chain_whip_01.settingsUI module

Module contents

mgear.shifter_classic_components.control_01 package

Submodules

mgear.shifter_classic_components.control_01.guide module

mgear.shifter_classic_components.control_01.settingsUI module

Module contents

mgear.shifter_classic_components.eye_01 package

Submodules

mgear.shifter_classic_components.eye_01.guide module

mgear.shifter_classic_components.eye_01.settingsUI module

Module contents

mgear.shifter_classic_components.foot_bk_01 package

Submodules

mgear.shifter_classic_components.foot_bk_01.guide module

mgear.shifter_classic_components.foot_bk_01.settingsUI module

Module contents

mgear.shifter_classic_components.hydraulic_01 package

Submodules

mgear.shifter_classic_components.hydraulic_01.guide module

mgear.shifter_classic_components.hydraulic_01.settingsUI module

Module contents

mgear.shifter_classic_components.leg_2jnt_01 package

Submodules

mgear.shifter_classic_components.leg_2jnt_01.guide module

mgear.shifter_classic_components.leg_2jnt_01.settingsUI module

Module contents

mgear.shifter_classic_components.leg_2jnt_02 package

Submodules

mgear.shifter_classic_components.leg_2jnt_02.guide module

mgear.shifter_classic_components.leg_2jnt_02.settingsUI module

Module contents

mgear.shifter_classic_components.leg_2jnt_freeTangents_01 package

Submodules

mgear.shifter_classic_components.leg_2jnt_freeTangents_01.guide module

mgear.shifter_classic_components.leg_2jnt_freeTangents_01.settingsUI module

Module contents

mgear.shifter_classic_components.leg_3jnt_01 package

Submodules

mgear.shifter_classic_components.leg_3jnt_01.guide module

mgear.shifter_classic_components.leg_3jnt_01.settingsUI module

Module contents

mgear.shifter_classic_components.leg_ms_2jnt_01 package

Submodules

mgear.shifter_classic_components.leg_ms_2jnt_01.guide module

mgear.shifter_classic_components.leg_ms_2jnt_01.settingsUI module

Module contents

mgear.shifter_classic_components.lite_chain_01 package

Submodules

mgear.shifter_classic_components.lite_chain_01.guide module

mgear.shifter_classic_components.lite_chain_01.settingsUI module

Module contents

mgear.shifter_classic_components.lite_chain_stack_01 package

Submodules

mgear.shifter_classic_components.lite_chain_stack_01.guide module

mgear.shifter_classic_components.lite_chain_stack_01.settingsUI module

Module contents

mgear.shifter_classic_components.lite_chain_stack_02 package

Submodules

mgear.shifter_classic_components.lite_chain_stack_02.guide module

mgear.shifter_classic_components.lite_chain_stack_02.settingsUI module

Module contents

mgear.shifter_classic_components.meta_01 package

Submodules

mgear.shifter_classic_components.meta_01.guide module

mgear.shifter_classic_components.meta_01.settingsUI module

Module contents

mgear.shifter_classic_components.mouth_01 package

Submodules

mgear.shifter_classic_components.mouth_01.guide module

Module contents

mgear.shifter_classic_components.mouth_02 package

Submodules

mgear.shifter_classic_components.mouth_02.guide module

Module contents

mgear.shifter_classic_components.neck_ik_01 package

Submodules

mgear.shifter_classic_components.neck_ik_01.guide module

mgear.shifter_classic_components.neck_ik_01.settingsUI module

Module contents

mgear.shifter_classic_components.sdk_control_01 package

Submodules

mgear.shifter_classic_components.sdk_control_01.guide module

mgear.shifter_classic_components.sdk_control_01.settingsUI module

Module contents

mgear.shifter_classic_components.shoulder_01 package

Submodules

mgear.shifter_classic_components.shoulder_01.guide module

mgear.shifter_classic_components.shoulder_01.settingsUI module

Module contents

mgear.shifter_classic_components.shoulder_02 package

Submodules

mgear.shifter_classic_components.shoulder_02.guide module

mgear.shifter_classic_components.shoulder_02.settingsUI module

Module contents

mgear.shifter_classic_components.shoulder_ms_01 package

Submodules

mgear.shifter_classic_components.shoulder_ms_01.guide module

Module contents

mgear.shifter_classic_components.spine_FK_01 package

Submodules

mgear.shifter_classic_components.spine_FK_01.guide module

mgear.shifter_classic_components.spine_FK_01.settingsUI module

Module contents

mgear.shifter_classic_components.spine_S_shape_01 package

Submodules

mgear.shifter_classic_components.spine_S_shape_01.guide module

mgear.shifter_classic_components.spine_S_shape_01.settingsUI module

Module contents

mgear.shifter_classic_components.spine_ik_01 package

Submodules

mgear.shifter_classic_components.spine_ik_01.guide module

mgear.shifter_classic_components.spine_ik_01.settingsUI module

Module contents

mgear.shifter_classic_components.spine_ik_02 package

Submodules

mgear.shifter_classic_components.spine_ik_02.guide module

mgear.shifter_classic_components.spine_ik_02.settingsUI module

Module contents

mgear.shifter_classic_components.squash4Sides_01 package

Submodules

mgear.shifter_classic_components.squash4Sides_01.guide module

mgear.shifter_classic_components.squash4Sides_01.settingsUI module

Module contents

mgear.shifter_classic_components.squash_01 package

Submodules

mgear.shifter_classic_components.squash_01.guide module

mgear.shifter_classic_components.squash_01.settingsUI module

Module contents

mgear.shifter_classic_components.tangent_spline_01 package

Submodules

mgear.shifter_classic_components.tangent_spline_01.guide module

mgear.shifter_classic_components.tangent_spline_01.settingsUI module

Module contents

mgear.shifter_classic_components.ui_container_01 package

Submodules

mgear.shifter_classic_components.ui_container_01.guide module

mgear.shifter_classic_components.ui_container_01.settingsUI module

Module contents

mgear.shifter_classic_components.ui_slider_01 package

Submodules

mgear.shifter_classic_components.ui_slider_01.guide module

mgear.shifter_classic_components.ui_slider_01.settingsUI module

Module contents

Module contents

mgear.shifter_epic_components package

Subpackages

mgear.shifter_epic_components.EPIC_arm_01 package

Submodules

mgear.shifter_epic_components.EPIC_arm_01.guide module

mgear.shifter_epic_components.EPIC_arm_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_chain_01 package

Submodules

mgear.shifter_epic_components.EPIC_chain_01.guide module

mgear.shifter_epic_components.EPIC_chain_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_control_01 package

Submodules

mgear.shifter_epic_components.EPIC_control_01.guide module

mgear.shifter_epic_components.EPIC_control_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_foot_01 package

Submodules

mgear.shifter_epic_components.EPIC_foot_01.guide module

mgear.shifter_epic_components.EPIC_foot_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_hydraulic_01 package

Submodules

mgear.shifter_epic_components.EPIC_hydraulic_01.guide module

mgear.shifter_epic_components.EPIC_hydraulic_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_leg_01 package

Submodules

mgear.shifter_epic_components.EPIC_leg_01.guide module

mgear.shifter_epic_components.EPIC_leg_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_mannequin_arm_01 package

Submodules

mgear.shifter_epic_components.EPIC_mannequin_arm_01.guide module

mgear.shifter_epic_components.EPIC_mannequin_arm_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_mannequin_leg_01 package

Submodules

mgear.shifter_epic_components.EPIC_mannequin_leg_01.guide module

mgear.shifter_epic_components.EPIC_mannequin_leg_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_neck_01 package

Submodules

mgear.shifter_epic_components.EPIC_neck_01.guide module

mgear.shifter_epic_components.EPIC_neck_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_shoulder_01 package

Submodules

mgear.shifter_epic_components.EPIC_shoulder_01.guide module

mgear.shifter_epic_components.EPIC_shoulder_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_spine_01 package

Submodules

mgear.shifter_epic_components.EPIC_spine_01.guide module

mgear.shifter_epic_components.EPIC_spine_01.settingsUI module

Module contents

mgear.shifter_epic_components.EPIC_spine_cartoon_01 package

Submodules

mgear.shifter_epic_components.EPIC_spine_cartoon_01.guide module

mgear.shifter_epic_components.EPIC_spine_cartoon_01.settingsUI module

Module contents

Module contents

mgear.simpleRig package

Submodules

mgear.simpleRig.menu module

`mgear.simpleRig.menu.install()`

Install Simple Rig submenu

mgear.simpleRig.simpleRigTool module

mgear.simpleRig.simpleRigUI module

mgear.simpleRig.version module

Module contents

mgear.synoptic package

Subpackages

mgear.synoptic.tabs package

Subpackages

mgear.synoptic.tabs.baker package

Submodules

mgear.synoptic.tabs.baker.widget module

Module contents

mgear.synoptic.tabs.biped package

Submodules

mgear.synoptic.tabs.biped.widget module

Module contents

mgear.synoptic.tabs.control_list package

Submodules

mgear.synoptic.tabs.control_list.searchControlsWidget module

mgear.synoptic.tabs.control_list.widget module

Module contents

mgear.synoptic.tabs.quadruped package

Submodules

mgear.synoptic.tabs.quadruped.widget module

Module contents

mgear.synoptic.tabs.visibility package

Submodules

mgear.synoptic.tabs.visibility.toggleGeoVisibilityWidget module

mgear.synoptic.tabs.visibility.widget module

Module contents

Module contents

Submodules

mgear.synoptic.menu module

mgear.synoptic.utils module

mgear.synoptic.version module

mgear.synoptic.widgets module

Module contents

mgear.vendor package

Submodules

mgear.vendor.Qt module

Minimal Python 2 & 3 shim around all Qt bindings

DOCUMENTATION

Qt.py was born in the film and visual effects industry to address the growing need for the development of software capable of running with more than one flavour of the Qt bindings for Python - PySide, PySide2, PyQt4 and PyQt5.

1. Build for one, run with all
2. Explicit is better than implicit
3. Support co-existence

Default resolution order:

- PySide2
- PyQt5

- PySide
- PyQt4

Usage:
>> import sys >> from Qt import QtWidgets >> app = QtWidgets.QApplication(sys.argv) >> button = QtWidgets.QPushButton("Hello World") >> button.show() >> app.exec_()

All members of PySide2 are mapped from other bindings, should they exist. If no equivalent member exist, it is excluded from Qt.py and inaccessible. The idea is to highlight members that exist across all supported binding, and guarantee that code that runs on one binding runs on all others.

For more details, visit https://github.com/mottosso/Qt.py

LICENSE

See end of file for license (MIT, BSD) information.

mgear.vendor.qjsonmodel module

Python adaptation of https://github.com/dridk/QJsonModel

Supports Python 2 and 3 with PySide, PySide2, PyQt4 or PyQt5. Requires https://github.com/mottosso/Qt.py

Usage:
Use it like you would the C++ version.

```
>>> import qjsonmodel
>>> model = qjsonmodel.QJsonModel()
>>> model.load({"key": "value"})
```

Test:
Run the provided example to sanity check your Python, dependencies and Qt binding.

$ python qjsonmodel.py

Changes:
This module differs from the C++ version in the following ways.

1. Setters and getters are replaced by Python properties
2. Objects are sorted by default, disabled via load(sort=False)
3. load() takes a Python dictionary as opposed to a string or file handle.
   - **To load from a string, use built-in *json.loads()***

   ```
   >>> import json
   >>> document = json.loads("{'key': 'value'}")
   >>> model.load(document)
   ```

   - **To load from a file, use *with open(fname)***

   ```
   >>> import json
   >>> with open("file.json") as f:
   ...     document = json.load(f)
   ...     model.load(document)
   ```

class mgear.vendor.qjsonmodel.**QJsonModel**(**args: Any*, ***kwargs: Any*)
Bases: `QAbstractItemModel`

clear()

columnCount(*parent=PySide2.QtCore.QModelIndex*)

data(*index*, *role*)

flags(*index*)

genJson(*item*)

headerData(*section*, *orientation*, *role*)

index(*row*, *column*, *parent=PySide2.QtCore.QModelIndex*)

json(*root=None*)

Serialise model as JSON-compliant dictionary

Parameters

root (`QJsonTreeItem`, *optional*) – Serialise from here defaults to the the top-level item

Returns

model as dict

load(*document*)

Load from dictionary

Parameters

document (*dict*) – JSON-compatible dictionary

parent(*index*)

rowCount(*parent=PySide2.QtCore.QModelIndex*)

setData(*index*, *value*, *role*)

class mgear.vendor.qjsonmodel.**QJsonTreeItem**(*parent=None*)

Bases: `object`

appendChild(*item*)

child(*row*)

childCount()

property key

classmethod load(*value*, *parent=None*, *sort=True*)

parent()

row()

property type

property value

Module contents

Submodules

mgear.menu module

`mgear.menu.create(menuId='mGear')`

Create mGear main menu

Parameters
menuId (*str, optional*) – Main menu name

Returns
main manu name

Return type
str

`mgear.menu.install(label, commands, parent='mGear', image='')`

Installer Function for sub menus

Parameters

- **label** (*str*) – Name of the sub menu
- **commands** (*list*) – List of commands to install
- **parent** (*str, optional*) – Parent menu for the submenu

`mgear.menu.install_help_menu(menuId='mGear')`

Install help menu section

Parameters
menuId (*str, optional*) – Main menu name

`mgear.menu.install_main_menu()`

Create top level mGear menu

`mgear.menu.install_utils_menu()`

Install Utilities submenu

mgear.version module

Module contents

mGear init module

exception `mgear.FakeException`

Bases: `Exception`

`mgear.getInfos(level)`

Get information from where the method has been fired. Such as module name, method, line number…

Parameters
level (*int*) – Level

Returns
The info

Return type
str

`mgear.getVersion()`

Get mGear version

Returns
mgear version

`mgear.install()`

`mgear.log(message, severity=32, infos=False)`

Log a message using severity and additional info from the file itself.

Severity has been taken from Softimage one:

- 1. Fatal
- 2. Error
- 4. Warning
- 8. Info
- 16. Verbose
- 32. Comment

Parameters

- **messages** (*str*) – The message
- **severity** (*int*) – Severity level.
- **infos** (*bool*) – Add extra infos from the module, class, method and line number.

`mgear.logInfos()`

Log version of Gear

`mgear.reloadModule(name='mgear', *args)`

Reload a module and its sub-modules from a given module name.

Parameters
name (*str*) – Module Name. Default value is "mgear".

`mgear.setDebug(b)`

Set the debug mode to given value.

Parameters
b (*bool*) – boolean

Returns
The previous value of the debug mode

Return type
bool

`mgear.toggleDebug()`

Toggle the debug mode value.

Returns;
bool: The new debug mode value.

`mgear.toggleLog()`

Toggle the log value.

Returns;
bool: The new debug mode value.

4.3 Rigbits

Content:

- *Introduction*
- *Modules*

4.3.1 Introduction

Rigbits is a rigging commont library with tools and functions to help the rigging workflow. This library is meant to be use with custom steps or other rigging tools.

4.3.2 Modules

mgear.rigbits	
mgear.rigbits.blendShapes	Rigbits blendshapes utilities and tools
mgear.rigbits.cycleTweaks	Cycle Tweaks module
mgear.rigbits.ghost	Rigbits Ghost module
mgear.rigbits.menu	
mgear.rigbits.postSpring	Post Spring tool
mgear.rigbits.proxySlicer	Rigbits proxy mesh slicer
mgear.rigbits.rbf_io	Handles the import and exporting of all supported RBF node types
mgear.rigbits.rivet	Rigbits rivet creator
mgear.rigbits.rope	Rigbits Rope rig creator
mgear.rigbits.sdk_io	Rigbits, SDK i/o
mgear.rigbits.tweaks	Rigbits tweaks rig module
mgear.rigbits.utils	Rigbits utilitie tools

mgear.rigbits

Functions

addBlendedJoint([oSel, compScale, blend, ...])	Create and gimmick blended joint
addJnt([obj, parent, noReplace, grp, jntName])	Create one joint for each selected object.
addNPO([objs])	Add a transform node as a neutral pose
addSupportJoint([oSel, select])	Add an extra joint to the blended joint.
alignToPointsLoop([points, loc, name])	Create space locator align to the plain define by at less 3 vertex
connectInvertSRT(source, target[, srt, axis])	Connect the locat transformations with inverted values.
connectLocalTransform([objects, s, r, t])	Connect scale, rotatio and translation.
connectUserDefinedChannels(source, targets)	Connects the user defined channels
connectWorldTransform(source, target)	Connect the source world transform of one object to another object.
connect_scale_from_world_matrix(driver, driven)	Set up node connections to make the driven object scale based on the driver object's world scale matrix.
createCTL([type, child])	Create a control for each selected object.
createInterpolateTransform([objects, blend])	Create space locator and apply gear_intmatrix_op, to interpolate the his pose between 2 selected objects.
duplicateSym(*args)	Duplicate one dag hierarchy to/from X/-X renaming "L" to "R"
matchPosfromBBox(*args)	Match the position usin bounding box of another object another.
matchWorldXform(*args)	Align 2 selected objects in world space
replaceShape([source, targets])	Replace the shape of one object by another.
selectDeformers(*args)	Select the deformers from the object skinCluster
spaceJump([ref, space])	Space Jump gimmick

mgear.rigbits.blendShapes

Rigbits blendshapes utilities and tools

Functions

blendshape_foc(deformed_obj)	Move existing blendshape node to the front of chain
connectWithBlendshape(mesh, bst[, wgt, ffoc])	Connect 2 geometries using blendshape node
connectWithMorph(mesh, bst[, wgt, ffoc])	Connect 2 geometries using morph node
getBlendShape(obj[, lv])	Get the blendshape node of an object.
getDeformerNode(obj[, lv, dtype])	Get the blendshape node of an object.
getMorph(obj[, lv])	Get the morph node of an object.
morph_foc(deformed_obj, morph_deformer)	Move existing morph node to the front of chain

mgear.rigbits.cycleTweaks

Cycle Tweaks module

This module content the tools and procedures to rig tweaks with a benigne cycle

Functions

cycleTweak(name, edgePair, mirrorAxis, ...)	The command to create a cycle tweak.
initCycleTweakBase(outMesh, baseMesh, ...[, ...])	Initialice the cycle tweak setup structure
inverseTranslateParent(obj)	Invert the parent transformation

mgear.rigbits.ghost

Rigbits Ghost module

Helper tools to create layered controls rigs

Functions

connect_matching_attrs(driver, driven[, ...])	Connect matching attributes from driven to driver node if they exist and have inputs.
createDoritoGhostCtl(ctl[, parent])	Create a duplicated Ghost control for doritos Create a duplicate of the dorito/tweak and rename the original with _ghost.
createGhostCtl(ctl[, parent, connect])	Create a duplicated Ghost control
ghostSlider(ghostControls, surface, sliderParent)	Modify the ghost control behaviour to slide on top of a surface

mgear.rigbits.menu

Functions

cCtl_sub(parent_menu_id)	Create control as child of selected elements
connect_submenu(parent_menu_id)	Create the connect local Scale, rotation and translation submenu
gimmick_submenu(parent_menu_id)	Create the gimmick joint submenu
install()	Install Rigbits submenu
install_utils_menu(m)	Install rigbit utils submenu
pCtl_sub(parent_menu_id)	Create control as parent of selected elements

mgear.rigbits.postSpring

Post Spring tool

creates a spring dynamic rig on top of a pre-existing FK chain rig.

Functions

bake_spring(*args)	Shortcut fro the Maya's Bake Simulation Options
build_spring(*args)	
postSpring([dist, hostUI, hostUI2, invertX])	Create the dynamic spring rig.
spring_UI(*args)	Creates the post tool UI

mgear.rigbits.proxySlicer

Rigbits proxy mesh slicer

Functions

slice([parent, oSel])	Create a proxy geometry from a skinned object

mgear.rigbits.rbf_io

Handles the import and exporting of all supported RBF node types

`mgear.rigbits.rbf_io.`**`RBF_FILE_EXTENSION`**

extention of the serialized json data

Type
str

`mgear.rigbits.rbf_io.`**`RBF_MODULES`**

nodeType: module api, normalized to fit the rbfManager

Type
Dict

__author__ = “Rafael Villar” __email__ = “rav@ravrigs.com”

Functions

exportRBFs(nodes, filePath)	exports the desired rbf nodes to the filepath provided
fileDialog([startDir, mode])	prompt dialog for either import/export from a UI
importRBFs(filePath)	import rbfs from file, using the assoiciated module type to recreate

mgear.rigbits.rivet

Rigbits rivet creator

Classes

rivet()	Create a rivet

mgear.rigbits.rope

Rigbits Rope rig creator

Functions

build_rope(*args)	
rope([DEF_nb, ropeName, keepRatio, lvlType, ...])	Create rope rig based in 2 parallel curves.
rope_UI(*args)	Rope tool UI

mgear.rigbits.sdk_io

Rigbits, SDK i/o

exportSDKs([“drivenNodeA”, “drivenNodeB”], “path/to/desired/output.json”) importSDKs(path/to/desired/output.json)

MIRRORING ——- # copy from source, say left, to target, right copySDKsToNode(“jacketFlap_L1_fk0_sdk”,

“neck_C0_0_jnt”, “jacketFlap_R1_fk0_sdk”)

invert/mirror the attributes necessary for the other side, # in this case it is the following attributes mirrorSDKkeys(“jacketFlap_R1_fk0_sdk”,

attributes=[“rotateZ”], invertDriver=True, invertDriven=False)

mirrorSDKkeys(“jacketFlap_R1_fk0_sdk”,
attributes=[“translateX”, “translateY”], invertDriver=True, invertDriven=True)

in this other instance, it was the same copySDKsToNode(“jacketFlap_L0_fk0_sdk”,

“neck_C0_0_jnt”, “jacketFlap_R0_fk0_sdk”)

`mgear.rigbits.sdk_io.`**`SDK_ANIMCURVES_TYPE`**

sdk anim curves to support

Type
list

Functions

`copySDKsToNode`(sourceDriven, targetDriver, ...)	Duplicates sdk nodes from the source drive, to any designated target driver/driven
`createSDKFromDict`(sdkInfo_dict)	Create a sdk node from the provided info dict
`exportSDKs`(nodes, filePath)	exports the sdk information based on the provided nodes to a json file
`getAllSDKInfoFromNode`(node)	returns a dict for all of the connected sdk/animCurve on the provided node
`getBlendNodes`(attrPlug)	Check the attrPlug (node.attr) provided for any existing connections if blendWeighted exists, return the appropriate input[#], if sdk, create a blendweighted and connect sdk, return input[#]
`getConnectedSDKs`(driven[, curvesOfType, ...])	get all the sdk, animcurve, nodes/plugs connected to the provided node.
`getMultiDriverSDKs`(driven[, sourceDriverFilter])	get the sdk nodes that are added through a blendweighted node
`getPynodes`(nodes)	Conevenience function to allow uses to pass in strings, but convert to pynodes if not already.
`getSDKDestination`(animNodeOutputPlug)	Get the final destination of the sdk node, skips blendweighted and conversion node to get the transform node.
`getSDKInfo`(animNode)	get all the information from an sdk/animCurve in a dictioanry for exporting.
`importSDKs`(filePath)	create sdk nodes from json file, connected to drivers and driven
`invertKeyValues`(newKeyNode[, invertDriver, ...])	Mirror keyframe node procedure, in case you need to flip your SDK's.
`mirrorSDKkeys`(node[, attributes, ...])	mirror/invert the values on the specified node and attrs, get the sdks and invert those values
`removeSDKs`(node[, attributes, ...])	Convenience function to remove, delete, all sdk nodes associated with the provided node
`stripKeys`(animNode)	remove animation keys from the provided sdk node

mgear.rigbits.tweaks

Rigbits tweaks rig module

Functions

createJntTweak(mesh, jntParent, ctlParent)	Create a joint tweak
createMirrorRivetTweak(mesh, edgePair, name)	Create a tweak joint attached to the mesh using a rivet.
createRivetTweak(mesh, edgePair, name[, ...])	Create a tweak joint attached to the mesh using a rivet
createRivetTweakFromList(mesh, edgePairList, ...)	Create multiple rivet tweaks from a list of edge pairs
createRivetTweakLayer(layerMesh, bst, ...[, ...])	Create a rivet tweak layer setup
create_mirror_proximity_tweak(mesh, ...[, ...])	Create a tweak joint attached to the mesh using a proximity pin.
create_proximity_tweak(mesh, edgePair, name)	Create a tweak joint attached to the mesh using a proximity pin
edgePairList([log])	Print and return a list of edge pairs to be use with createRivetTweakLayer and createRivetTweakFromList
negateTransformConnection(in_rot, out_rot[, ...])	
pre_bind_matrix_connect(mesh, joint, jointBase)	Connect the pre bind matrix of the skin cluseter to the joint parent.
resetJntLocalSRT(jnt)	Reset the local SRT and jointOrient of a joint

mgear.rigbits.utils

Rigbits utilitie tools

Functions

createHotkeys(*args)	Create mGear custom hotkey functions ready to be use.
createRunTimeCommand(name, rCmd[, ann])	Create run time commands from raw string.

4.4 Shifter Rig Builder

Content:

- *Introduction*
- *Modules*

4.4.1 Introduction

Shifter is the default autorigging modular system included with mGear. This system is provided with the hope to solve the majority of your rigging necesities. From a freelancer or small boutique to a big studio.

Shifter can be use in a wide variety of projects. Animated feature films, video games, VR, VFX, TV series, etc… and can build rigs for any kind of asset. i.e: cartoon characters, creatures, robots, props, vehicles, etc…

Shifter have been build on top of mGear framework and can co-exist with other rigging systems inside mGear framework. In other words, if Shifter's paradigm doesn't fit your rigging needs, you can create another rigging system using the same mGear base modules.

4.4.2 Modules

4.5 Simple Rig

- *Introduction*
- *Modules*

4.5.1 Introduction

Simple rigging system with option to conver to Shifter rigging system.

Simple Rig 2.0 Intro

4.5.2 Modules

mgear.simpleRig
mgear.simpleRig.menu

mgear.simpleRig

mgear.simpleRig.menu

Functions

install()	Install Simple Rig submenu

4.6 Synoptic

Content:

- *Introduction*
- Creating a new synoptic tab
- Adding Synoptic to a Shifter rig
- Adding Synoptic to any rig
- *Modules*

4.6.1 Introduction

Synoptic view is a user interface to help the animators to interact better and faster with the rigs. Synoptic view can be use to select rig parts, save keyframes, mirror poses, switch spaces and many other functions that an animator would need.

4.6.2 Modules

4.7 Modules flat List

Just a complete list of all modules.

4.7.1 mgear

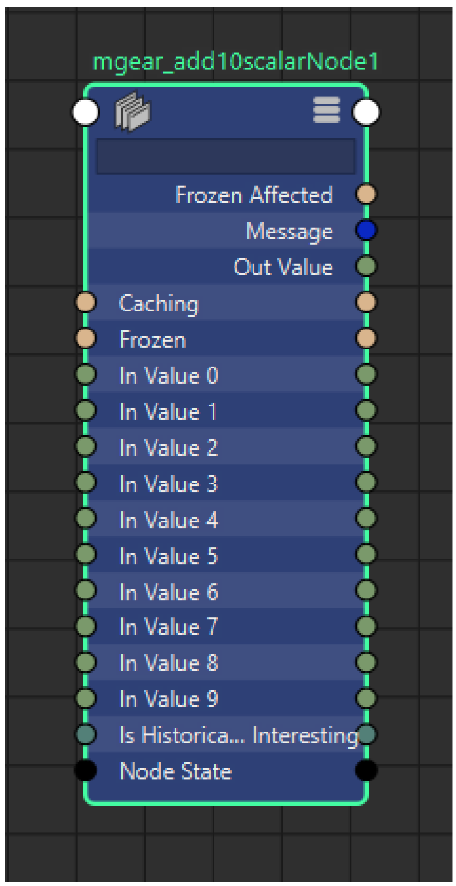
CHAPTER

FIVE

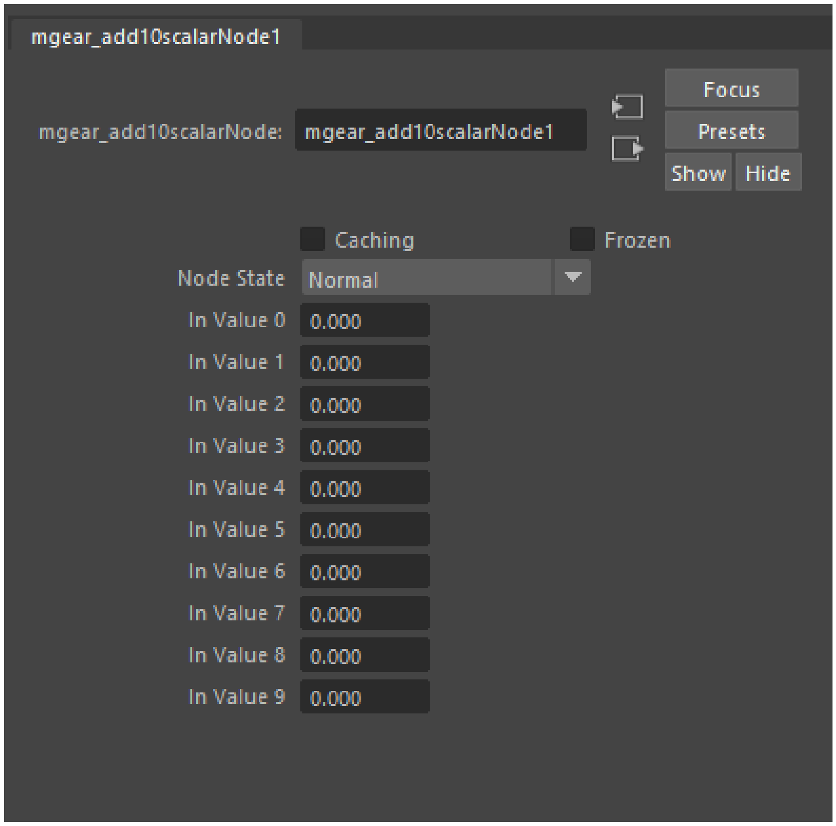
CUSTOM SOLVERS/DEFORMERS

mGear custom C++ Solvers/Deformers Documentation

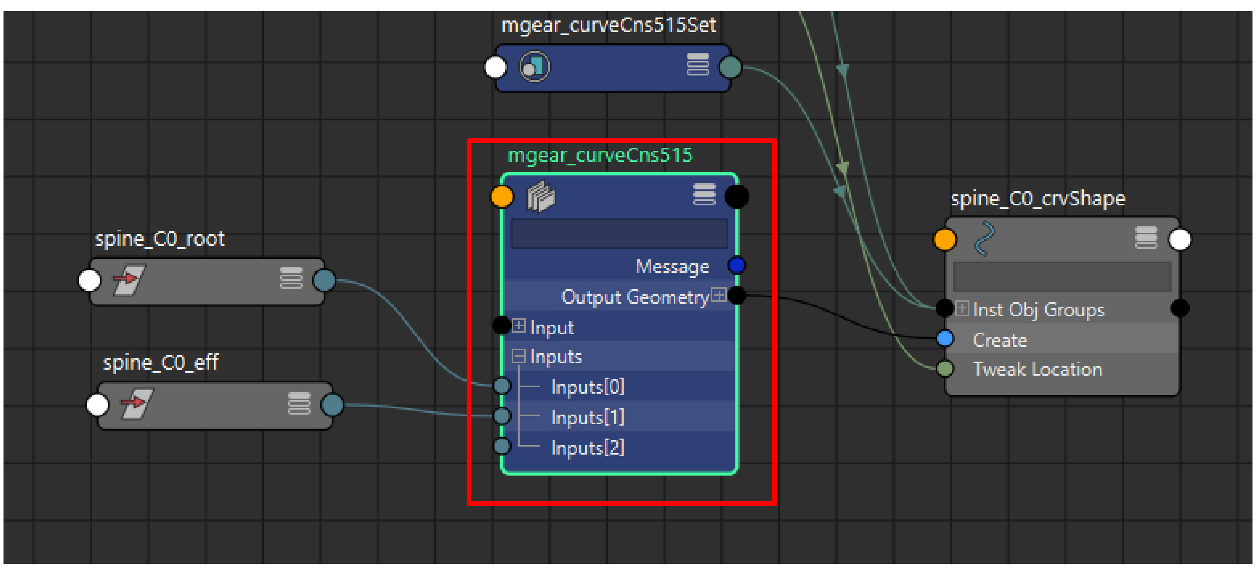
5.1 mgear_add10Scalar



Add 10 scalar values. This node is deprecated. Only kept for backwards compatibility reasons.

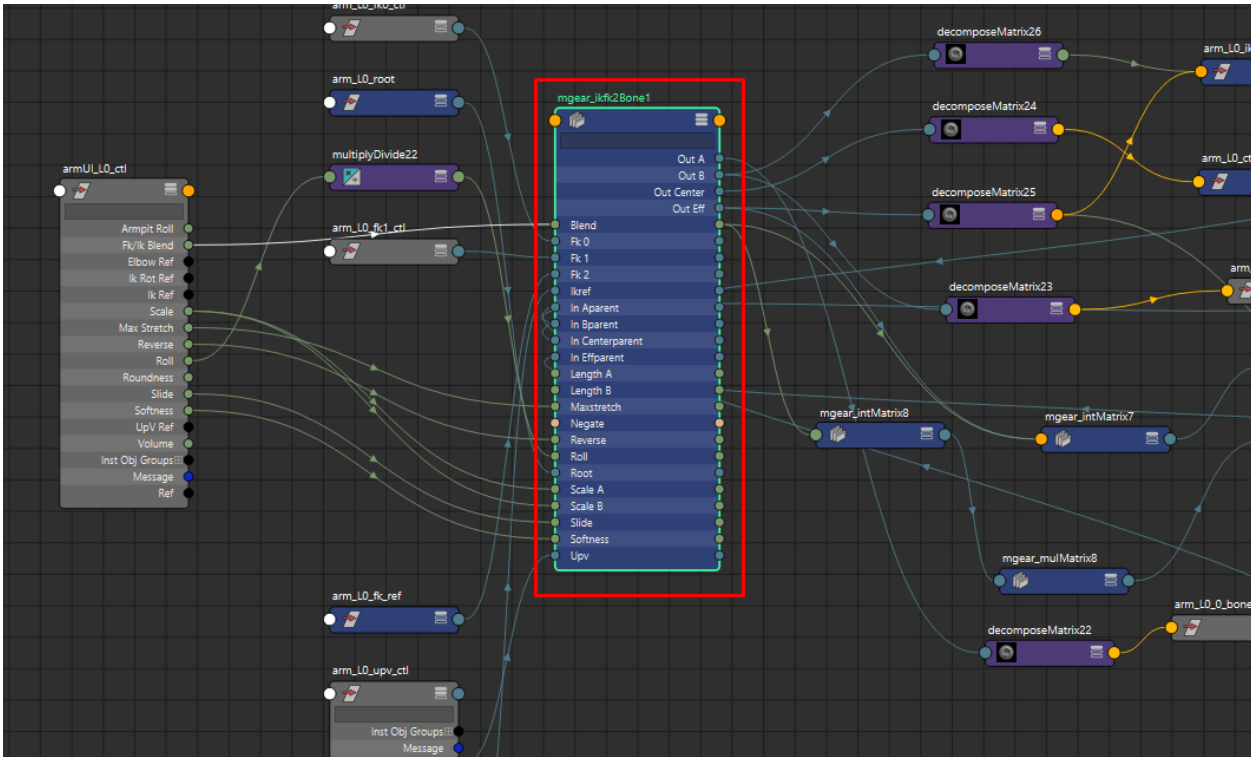


5.2 mgear_curveCns



Generate a curve based on the input positions. This is used in the Shifter guides to create the visual connections with the guide locations.

5.3 mgear_ikfk2Bone

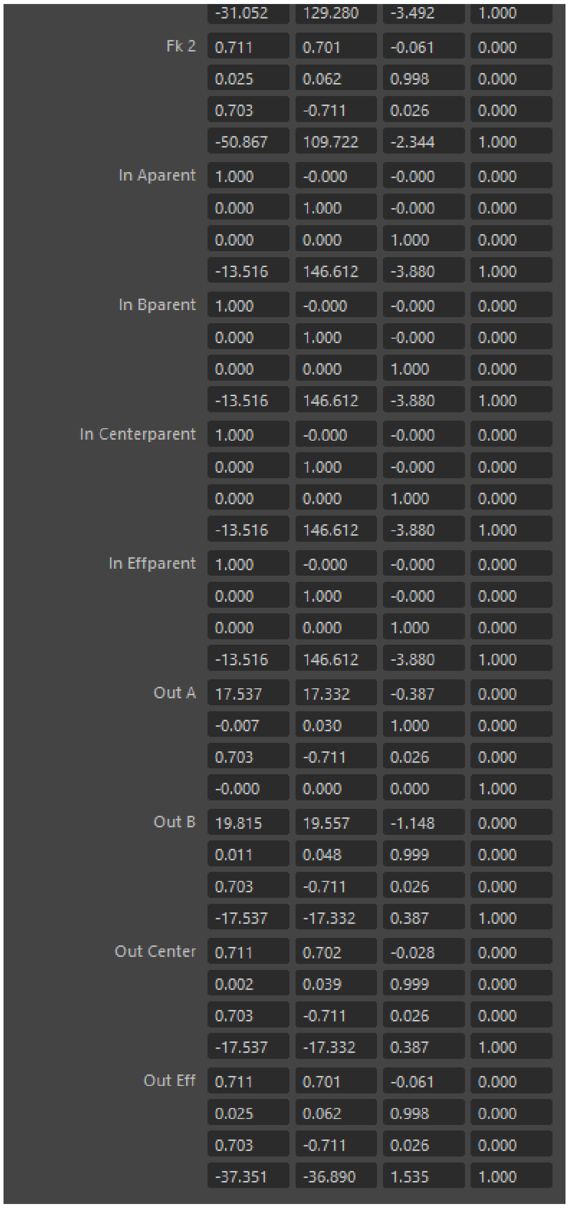


IK FK 2 bones chain solver.

Mainly used for legs and arms. The solver encapsulate many functions that will be very complex and expensive to evaluate using vanilla Maya nodes and other techniques like expressions.

Some of the features are, soft IK, reverse IK, squash, stretch, independent bone length, slide and roll.

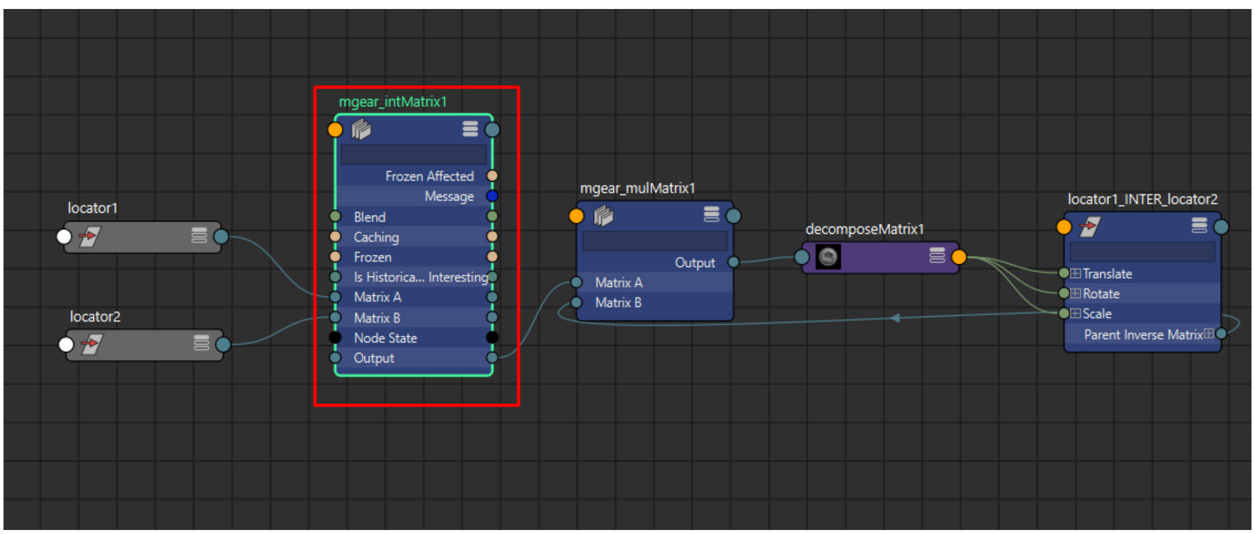

mgear_ikfk2Bone1
multiplyDivide22
decomposeMatrix22
decompo
mgear_ikfk2Bone:
mgear_ikfk2Bone1
Focus
Presets
Show
Hide
Caching
Frozen
Node State
Normal
Blend
0.000
Length A
2.696
Length B
2.799
Negate
Scale A
1.000
Scale B
1.000
Roll
0.000
Maxstretch
1.500
Slide
0.500
Softness
0.000
Reverse
0.000
Root
Ikref
Upv
Fk 0
Fk 1
Fk 2



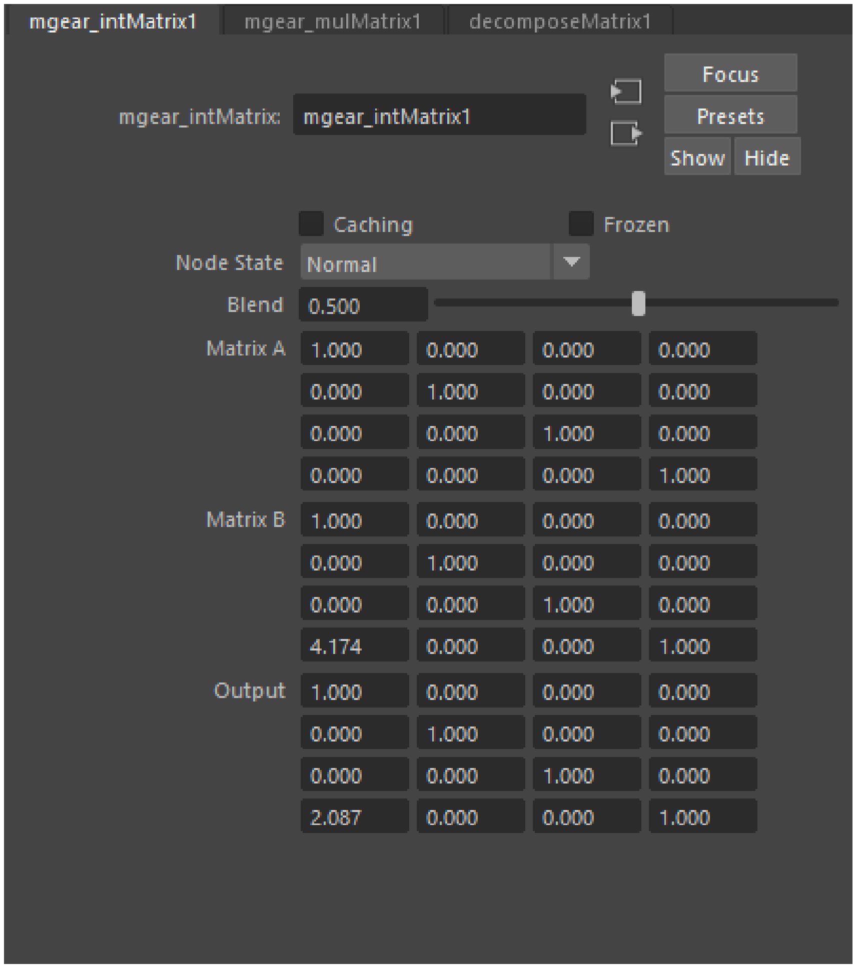
- **Blend:** IK fk blend
- **Length A:** Rest length of the bone A (Arm)
- **Length B:** Rest length of the bone B (Forearm)
- **Negate:** Negate the solver direction (i.e: the right side arm negates the direction)
- **Scale A:** Length multiplier for the arm.
- **Scale B:** Length multiplier for the forearm.
- **Roll:** Roll value. This value is complementarity to the Up Vector control.
- **Max stretch:** Maximum stretching value for the IK behavior. Value 1 will represent the original size and not scale.
- **Slide:** Slide the elbow position between the lengths of the arm and forearm. Value of .5 represents the middle point, whatever the proportions ratio is between the arm and forearm.

- **Reverse:** Reverse the IK solver direction. (i.e: human leg vs Chicken leg)
- **root:** Matrix. Root of the component world matrix
- **ikref:** Matrix. IK control world matrix
- **upv:** Matrix. Up vector control world matrix
- **FK0:** Matrix. FK arm control world matrix
- **FK1:** Matrix. FK forearm control world matrix
- **FK2:** Matrix. FK hand control world matrix
- **in A parent:** Matrix. Output bone A parent matrix (arm)
- **in B parent:** Matrix. Output bone A parent matrix (forearm)
- **in Center parent:** Matrix. Output elbow parent matrix
- **in Eff parent:** Matrix. Output effector parent matrix (hand)
- **out A:** Matrix. Output world matrix for bone A (arm)
- **out B:** Matrix. Output world matrix for bone B (forearm)
- **out Center:** Matrix. Output world matrix for elbow
- **out Eff:** Matrix. Output world matrix for effector (hand)

5.4 mgear_intMatrix

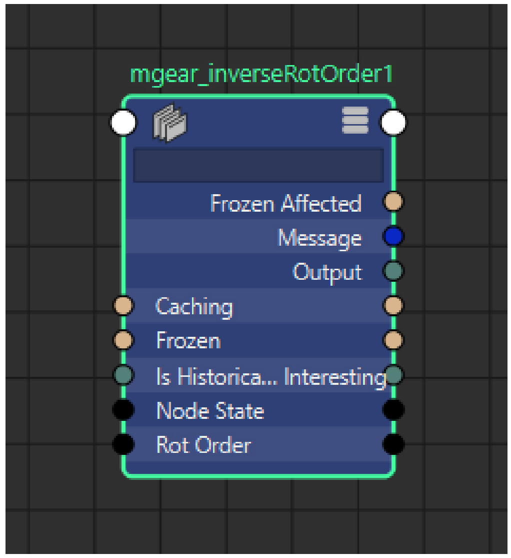


Interpolate between 2 input matrix using a blend value. The rotation is calculated in quaternion.

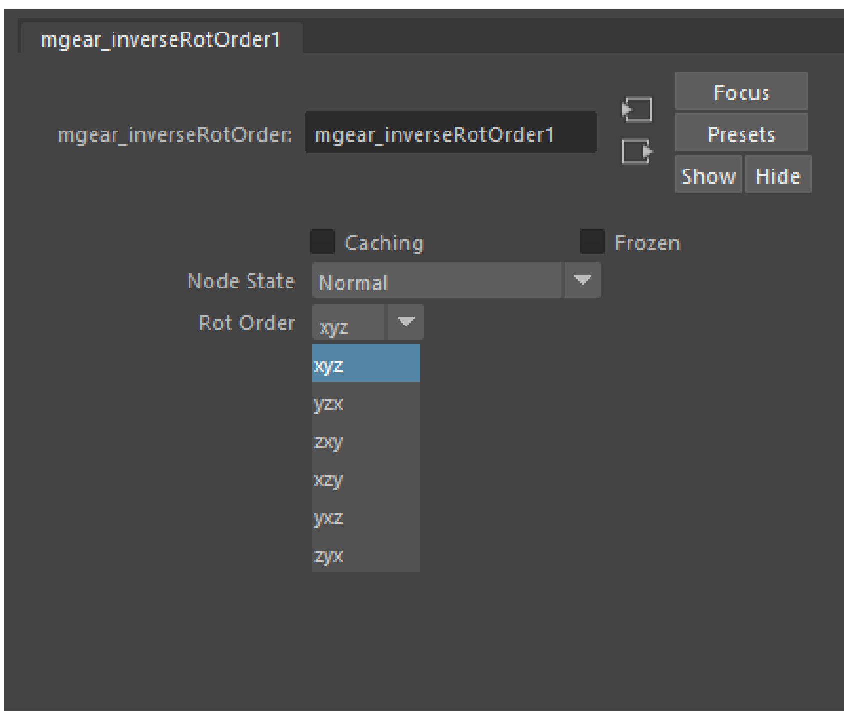


- **Blend:** Blend between the 2 input matrix
- **Matrix A:** Input Matrix
- **Matrix B:** Input Matrix
- **Output:** Output Matrix

5.5 mgear_inverseRotOrder

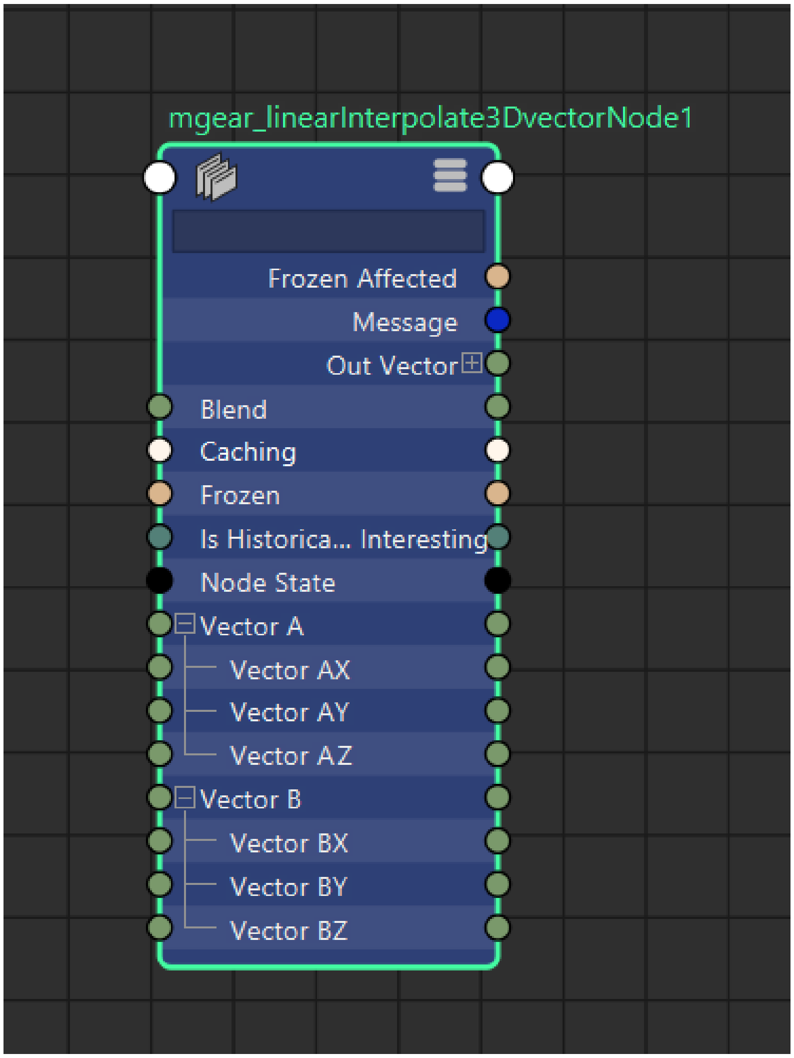


Invert the rotation order. For example and input of "XYZ" will output ZYX. This is very useful when you need to negate an animated rotation order to avoid gimbal.

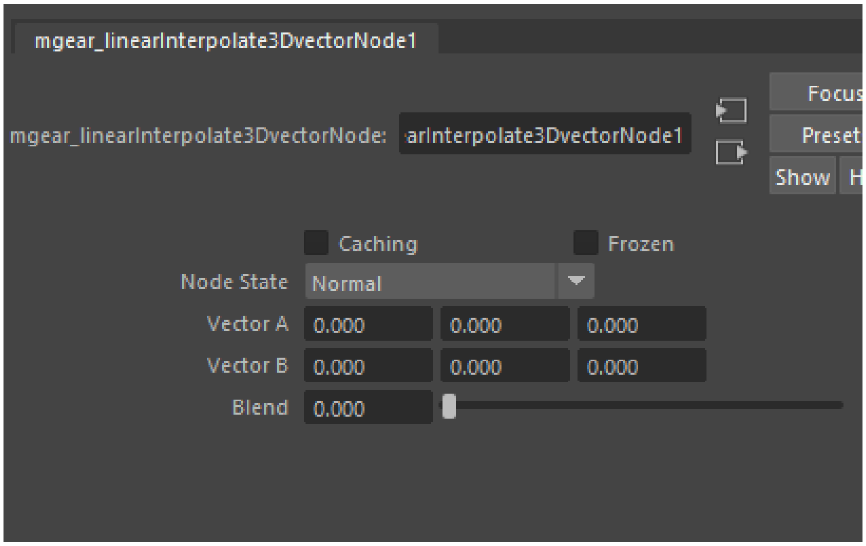


- **Rot Order:** Rotation order to invert

5.6 mgear_linearInterpolate3Dvector

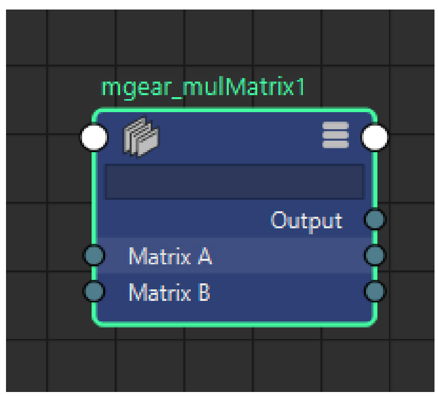


Interpolate between 2 input vector using a blend value. i.e: the XYZ position of 2 transforms.

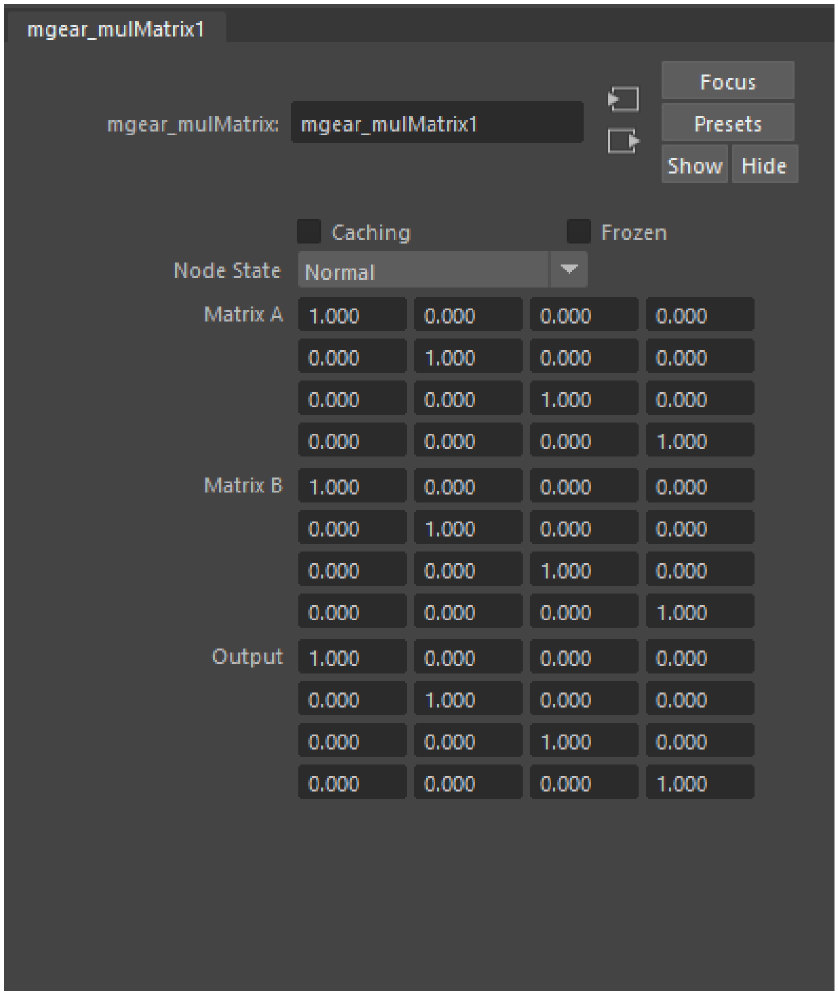


- **Blend:** Blend between the 2 input matrix
- **Vector A:** Input Vector
- **Vector B:** Input Vector
- **Out Vector:** Output Vector

5.7 mgear_mulMatrix

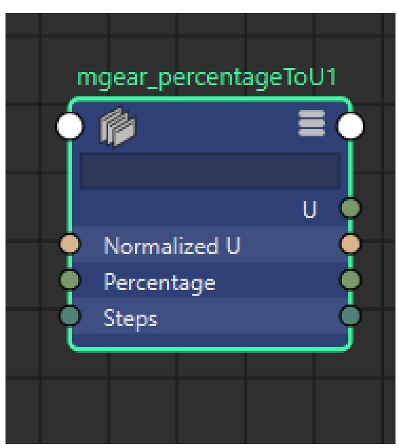


Multiply 2 input matrix. The only advantage between this node and the default one, is that with this you can visualize the values in the attribute editor. With the default Maya's multMatrix node the values are not visible, this make very difficult debugging rigs in some situations. For the rest are exactly the same and interchangeable. In terms of performance there is not noticeable difference.

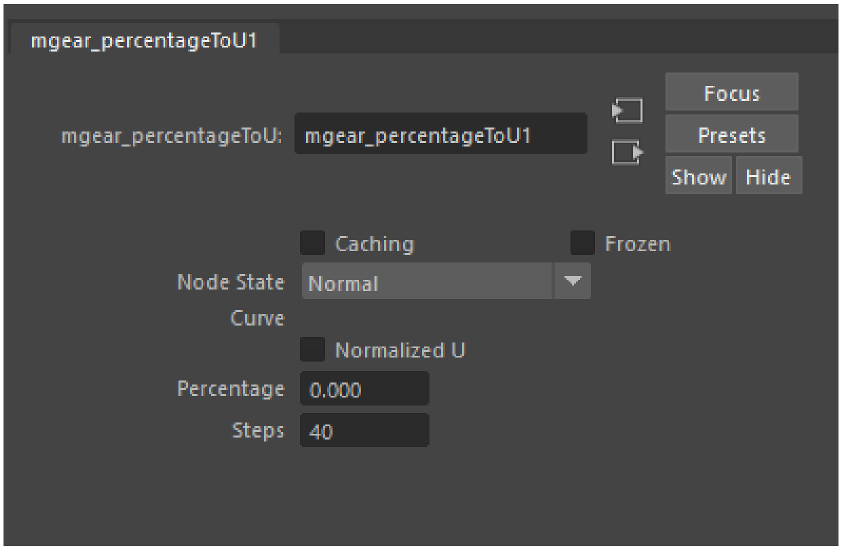


- **Matrix A:** Input Matrix
- **Matrix B:** Input Matrix
- **Output:** Output Matrix

5.8 mgear_percentageToU

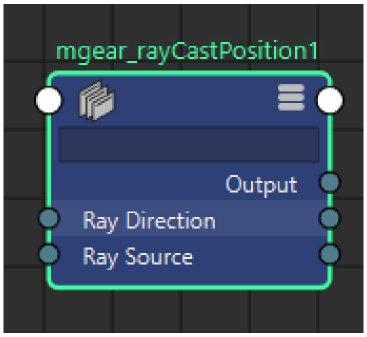


Converts a percentage values to a curve U value.

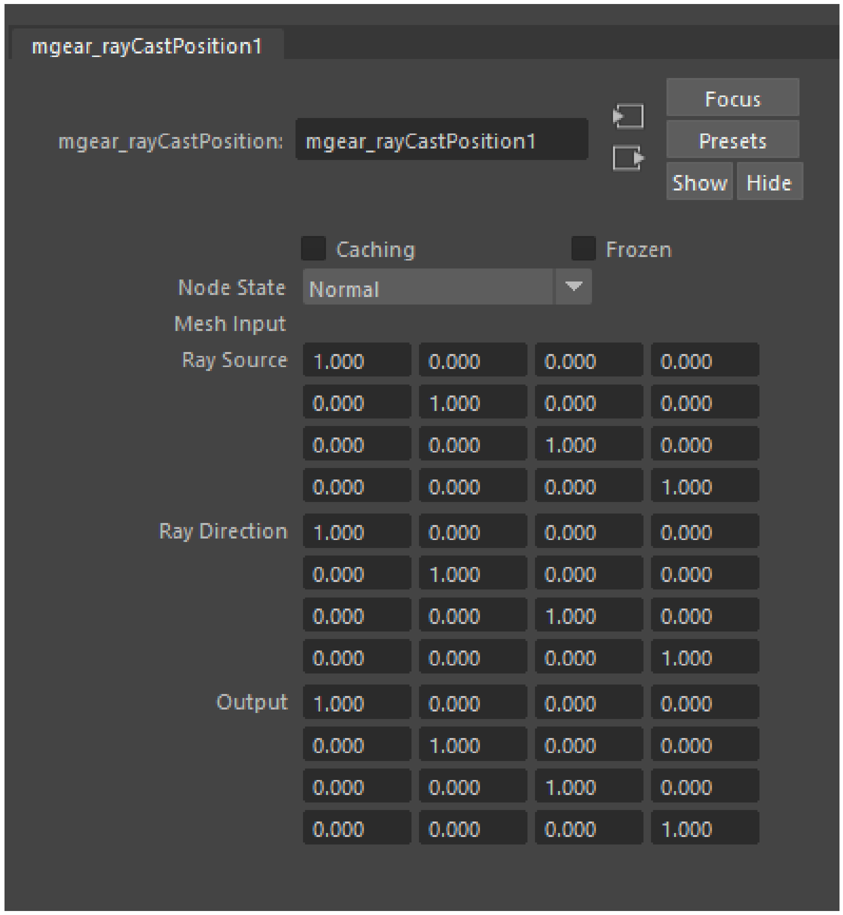


- **Curve:** Input curve.
- **Normalized U:** If active will normalize U value between 0 and 1.
- **Percentage:** Percentage value.
- **Steps:** Interpolation steps.

5.9 mgear_rayCastPosition

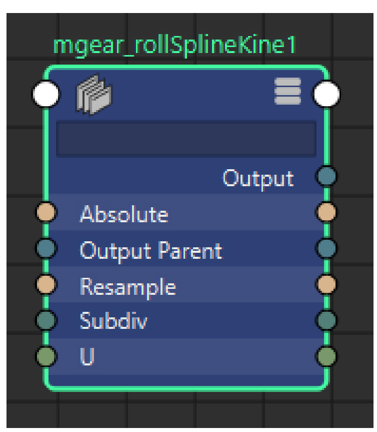


Raycast the contact position using a vector from 2 position inputs The operation is set using Matrix, but usually we will use it only to get the position.



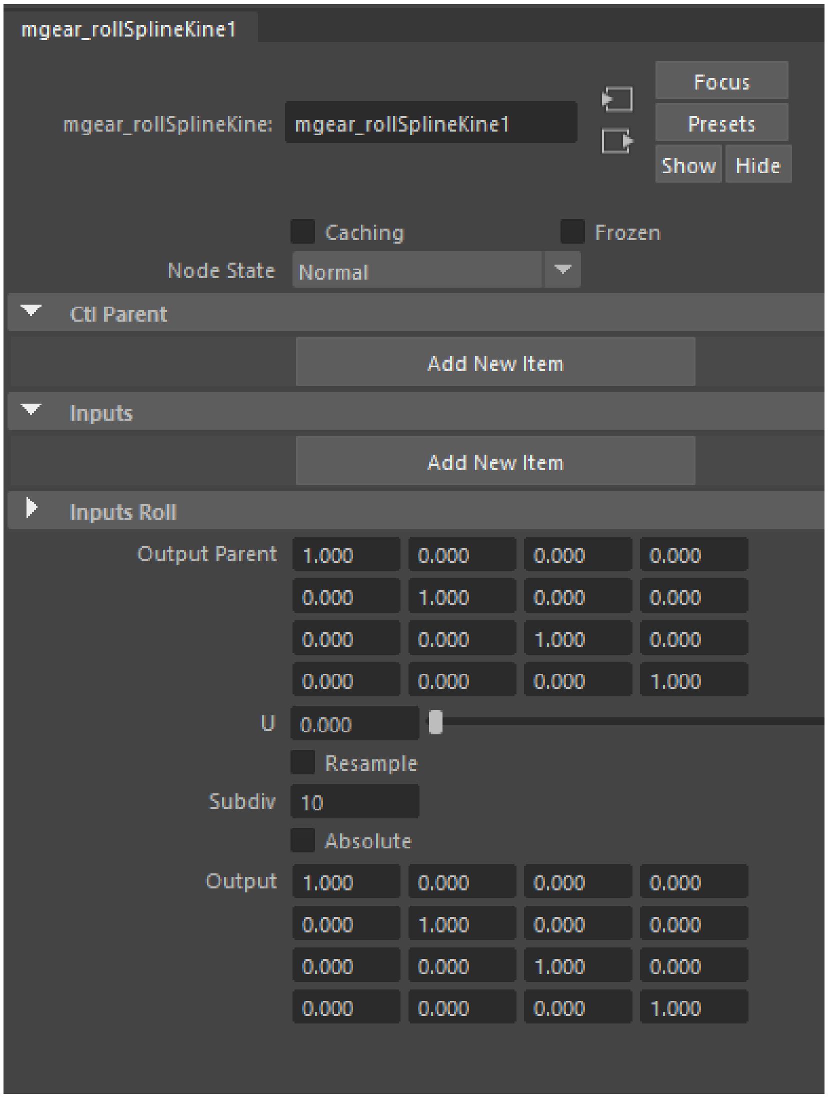
- **Mesh Input:** Contact Mesh.
- **Ray Source:** Matrix. starting position for the vector
- **Ray Direction:** Matrix. Aim position for the vector
- **Output:** Output Matrix with the position on the Contact Mesh

5.10 mgear_rollSplineKine



Roll Spline kinematic is a Bezier curve style with roll support. This solver is used in several Shifter components.

Mainly arms and legs. This will be the equivalent of or similar to a ribbon setup, with the advantage of been much more lightweight at evaluation time. Every input transform (world matrix plug) represents a point in the Bezier curve. And the scale in X axis of each transform represents the length of the Bezier tangents. The main limitation is that the 2 tangents are always of the same length for each point. In order to workaround this you can use 2 transforms in the same position. One representing each tangent, so the scale can be control independently. Each mgear_rollSplineKine node, outputs only one point in the U value of the curve.

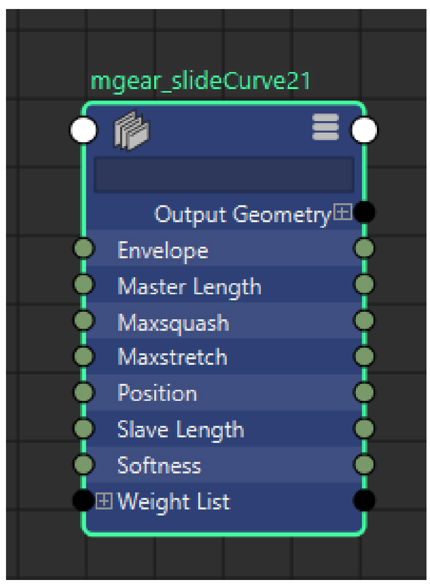


- **Ctl Input:** Array of Matrix. Input control points parent matrix.
- **inputs:** Array of Matrix. Transform controls world matrix
- **inputs Roll:** Array of Rotations. Transform controls rotation.
- **Output Parent:** Output transform parent Matrix.
- **U:** U percentage position represented from 0 to 1. NOTE: Usually the value should be always between 0.0001 and 0.999. The most extreme values are not taking in consideration the tangency for the output transform.
- **Resample:** Resample the output curve.
- **Subdiv:** Number of subdivision in the curve. Higher values will create a smoother curve but slower evaluation. Small values will create a more stepped curve, this can cause artifacts when sliding a transform on the U value.

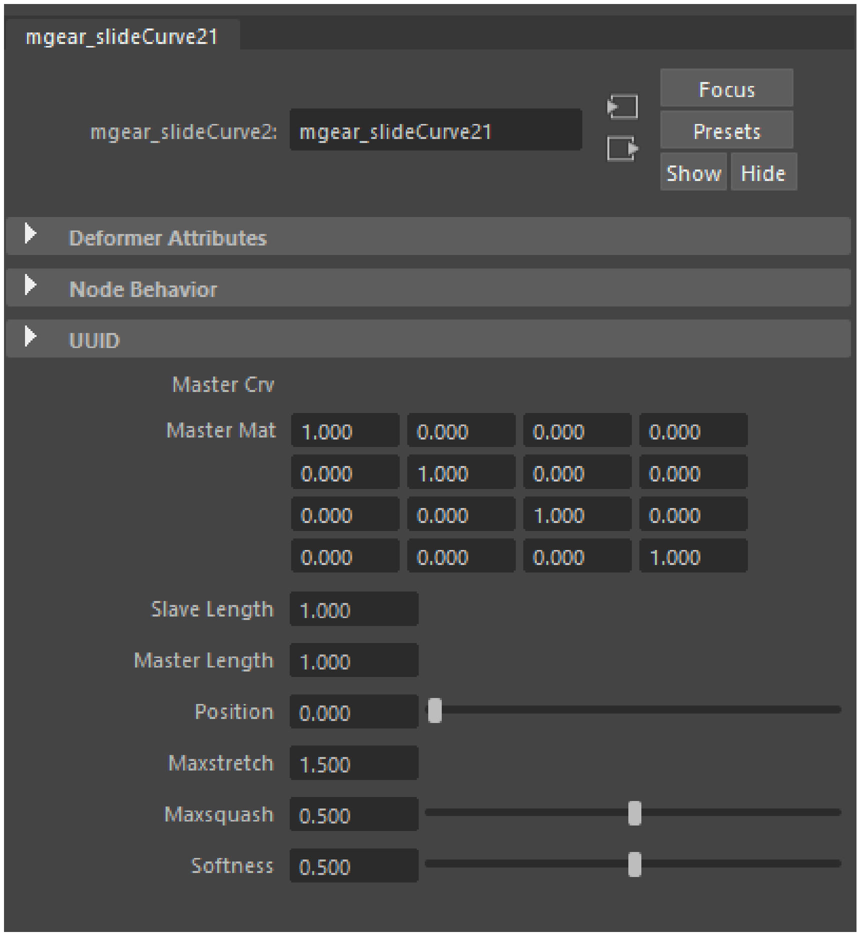
NOTE: Also, can have a little discordance between the same component in the left and right side. Due inversion of the direction. Usually the solution is simple as increase the subdivision.

- **Absolute:** Change the way that the subdivision are distributed in the curve.
- **Output:** Output transform Matrix.

5.11 mgear_slideCurve

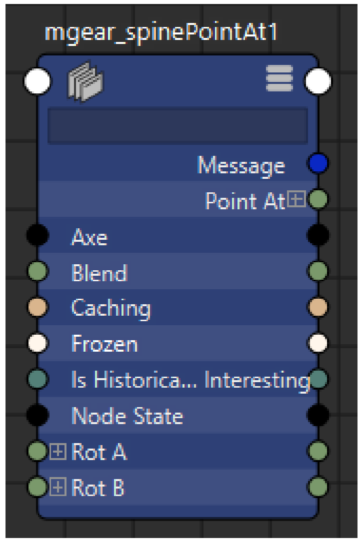


Deform a curve, sliding it on top of other. i.e: It is used in the Shifter spine component. Use this function to apply the deformer: mgear.core.applyop.gear_curveslide2_op

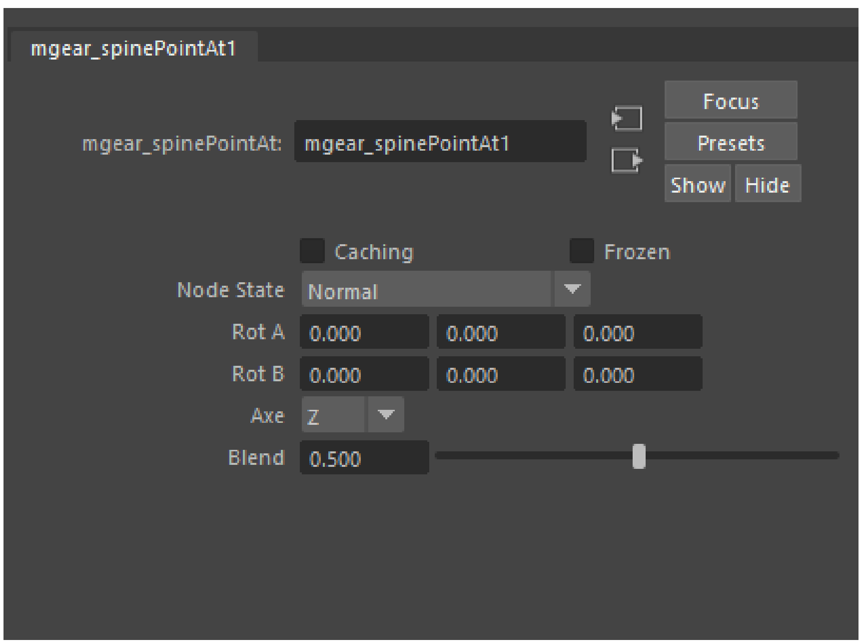


- **Master Crv:** Input Curve.
- **Master Mat:** Master curve matrix.
- **Slave Length:** Slave curve length.
- **Master Length:** Master curve length.
- **Position:** Slave curve position.
- **Max stretch:** Maximum stretch of the slave curve.
- **Max squash:** Maximum squash of the slave curve.
- **Softness:** Soft clamping for squash and stretch.

5.12 mgear_spinePointAt

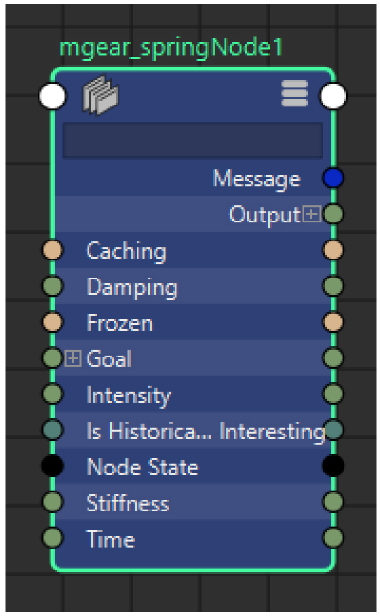


Point at an axis direction base in 2 input rotations. Note: This solver was design to handle the spine twist, but currently is not used in any component.

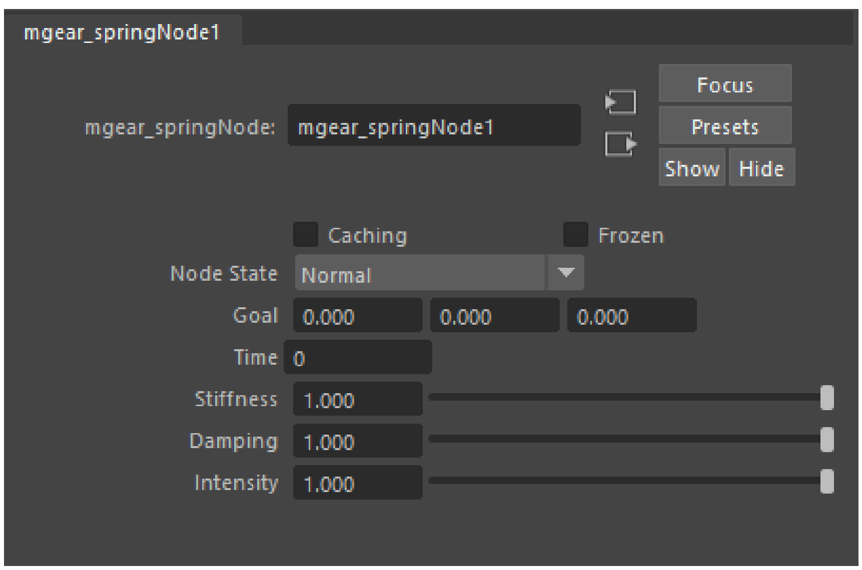


- **Rot A:** Input rotation A.
- **Rot B:** Input rotation B.
- **Axe:** Aim axis.
- **Blend:** Blend value between the 2 rotations

5.13 mgear_springNode

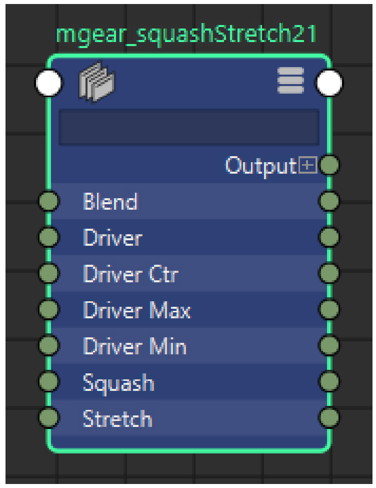


Spring dynamic solver based in goal position.

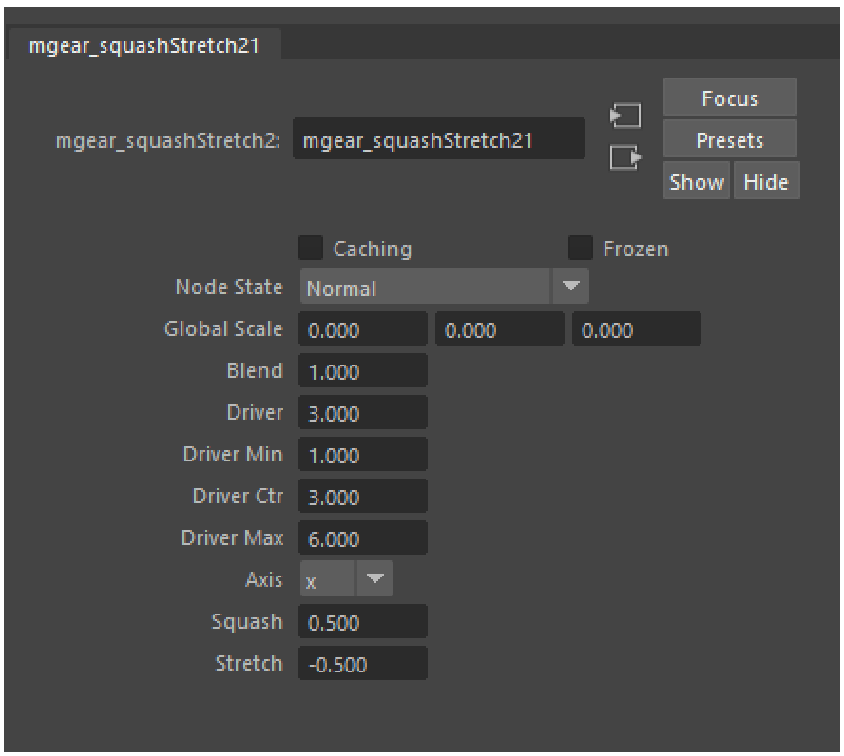


- **Goal:** Position goal.
- **Time:** Current time input.
- **Stiffness:** Stiffness value.
- **Damping:** Damping value.
- **Intensity:** Intensity value.

5.14 mgear_squashStretch_attr



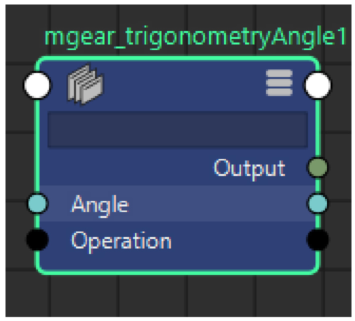
Squash and stretch solver.



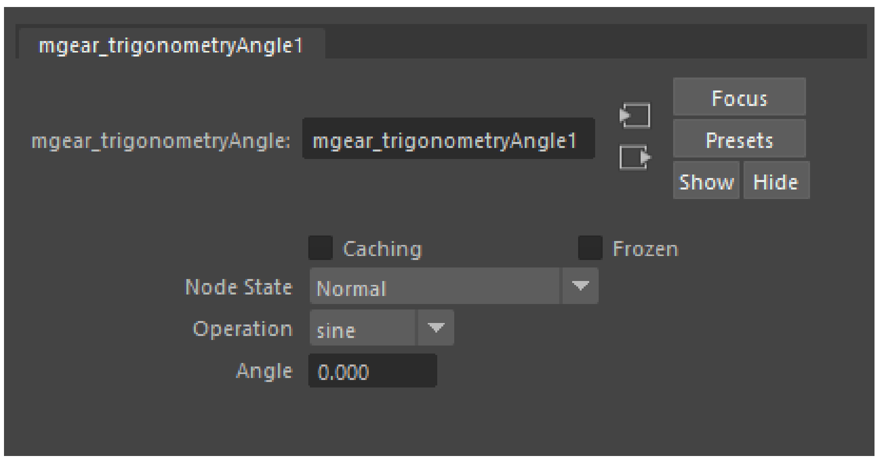
- **Global Scale:** Global scale reference.
- **Blend:** Blend to deal down the squash and stretch effect.
- **Driver:** Driver rest value.
- **Driver Min:** Driver minimum value where the squash and stretch effect will be calculated.
- **Driver ctr:** Driver control value.
- **Driver Max:** Driver maximum value where the squash and stretch effect will be calculated.
- **Axis:** Axis along the squash and stretch value will be calculated.

- **Squash:** Multiplication value for the squash direction.
- **Stretch:** Multiplication value for the stretch direction.

5.15 mgear_trigonometyAngle

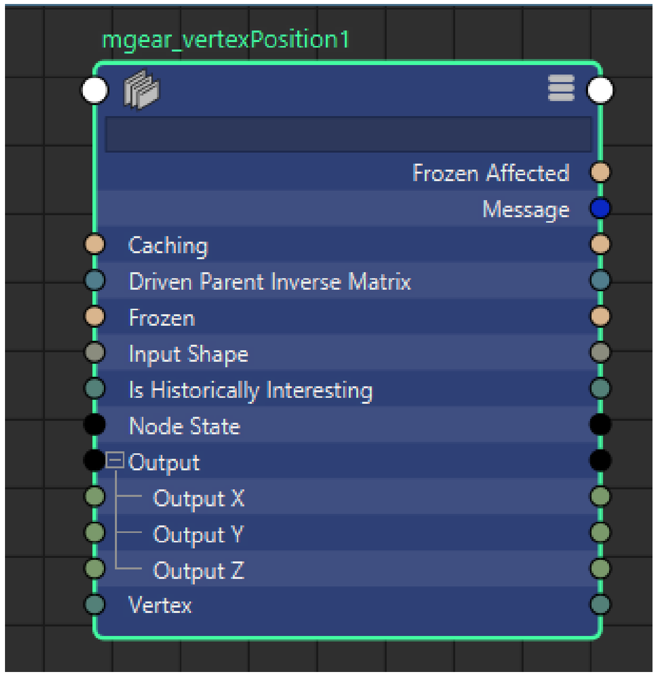


Sine and cosine trigonometry node

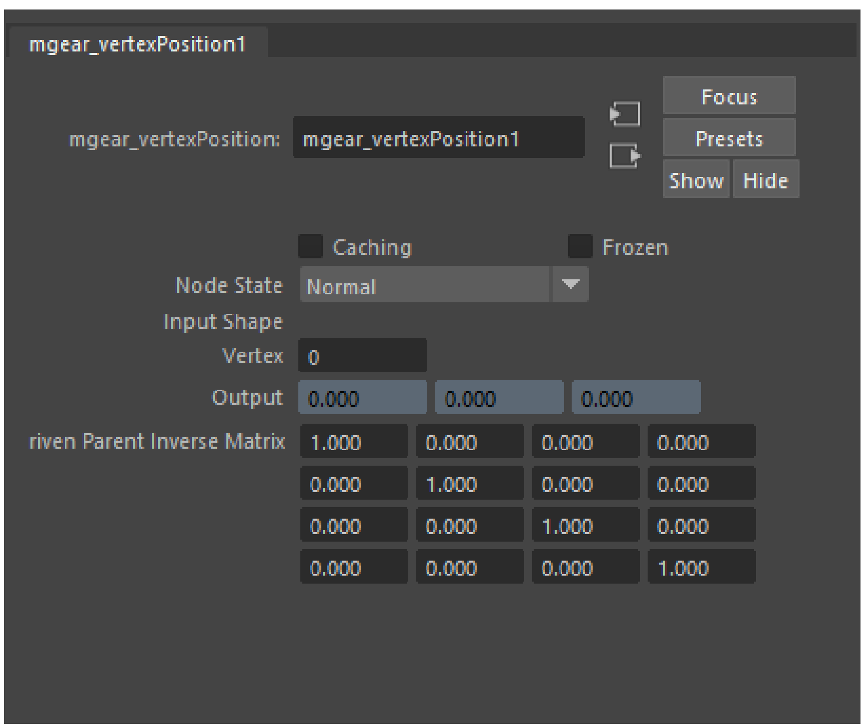


- **Operation:** Sine or Cosine.
- **Angle:** Input angle.

5.16 mgear_vertexPosition



Get the world position of a given vertex



- **Input Shape:** Input mesh shape.
- **Vertex:** Vertex index number to track.
- **Output:** Output position.

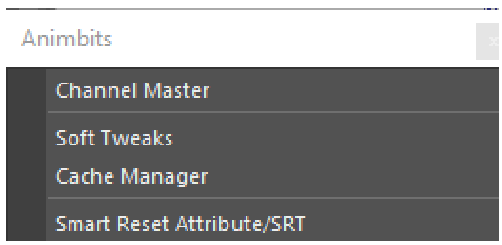
- **Driven parent invert Matrix:** Driven parent invert matrix.

CHAPTER

SIX

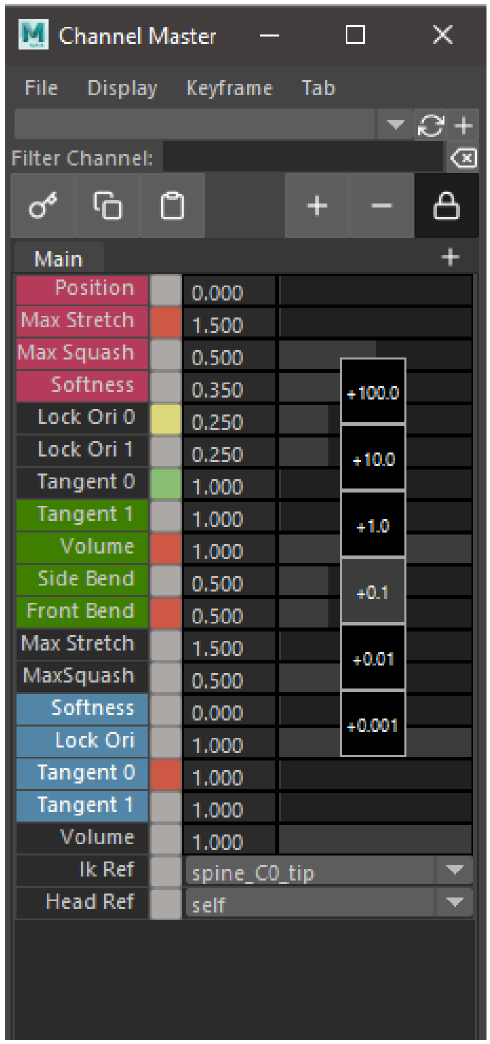
ANIMBITS USER DOCUMENTATION

Tools for animators.

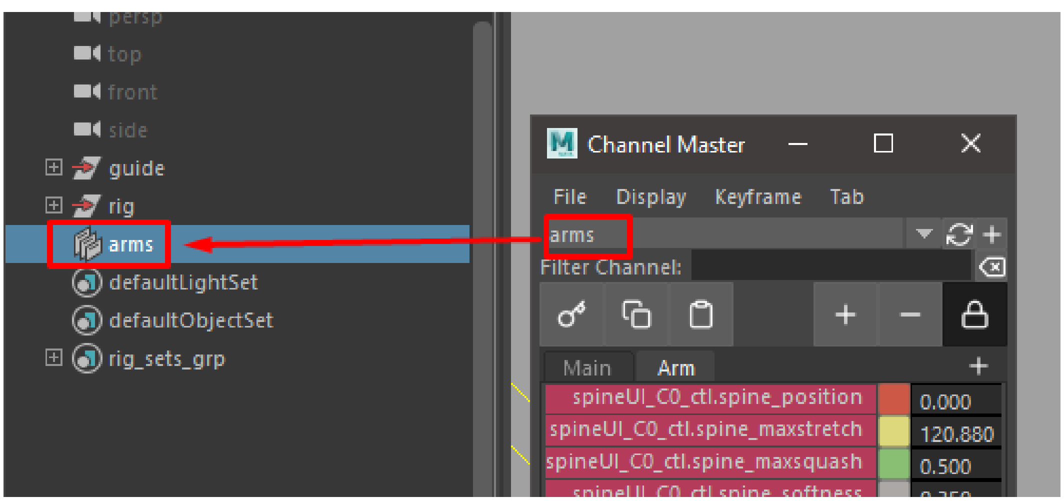


6.1 Channel Master

Channel Master is an alternative ChannelBox, but allowing the user to create custom configurations and store it.



The configuration is stored in a persistent node in the scene. The data can be exported and imported.



The channels can be a mix from different objects. The only rule is that the object that contains the channel and the channel master node should be in the same namespace.

However the internal data is not stored with name spaces, so the same configuration can be re-used in different name spaces.

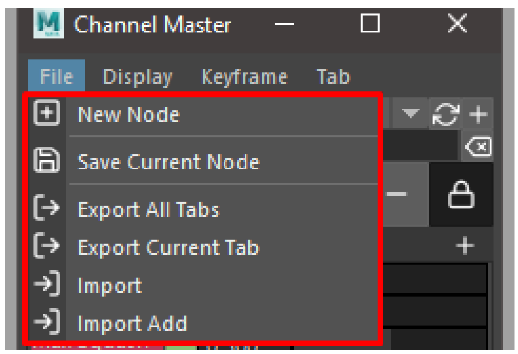
The Main tab works like the regular ChannelBox and the data is rebuild on the flight. The advantage is that is possible to lock the content using the lock icon. Also create a node is not required to use the main tab

To create custom configuration, require a custom node.

> **Warning:** Custom configurations needs to be saved. If to be stored in the node. Missing to save the new configuration will revert the channel configuration to the previous state when the node is refresh or change node in the drop-down menu

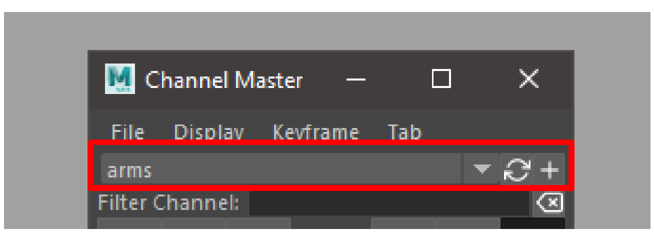
> **Warning:** The tool has been design to avoid bottleneck the evaluation while playback by turning off the refresh and updating on demand in some situations. When scrubbing the timeline still updating the channel values. This will take some FPS away (Only when Scrubbing, not in Playback).
>
> If you need to do scrubbing a lot and need the performance. You can change the Channel Master to empty Main tab, by lock the tab when nothing is selected. Or Just close Channel Master
>
> We planning to add more control over this in the future.

File Menu:



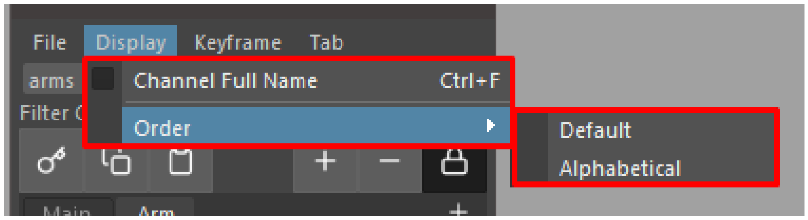
- **New Node:** Creates a new node

Also is possible to add nodes from the node bar in the UI

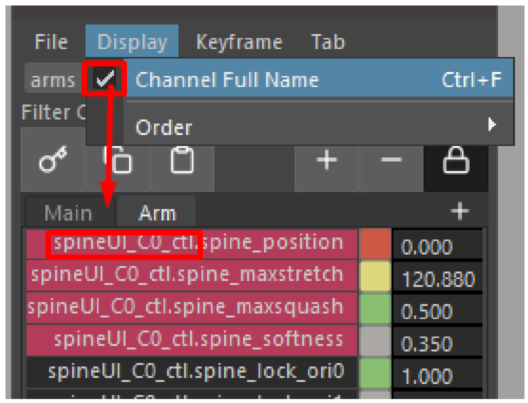


- **Save Current Node:** Save the current node configuration
- **Export All Tabs:** Export the current node configuration to a Json file
- **Export Current Tab:** Export current tab configuration to Json file
- **Import:** Import and create a new node
- **Import Add:** Import and add the configuration to an existing node

Display Menu:

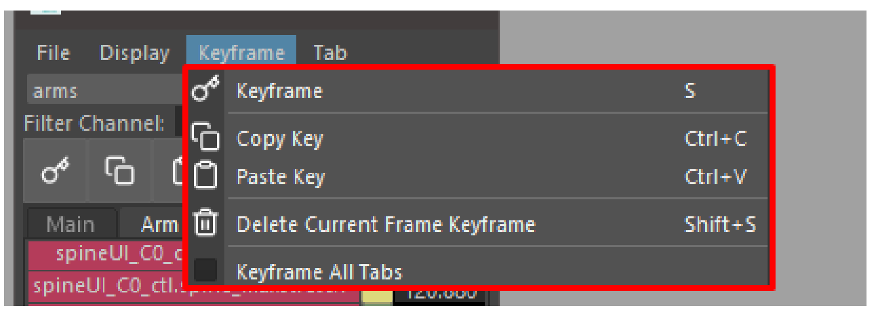


- **Channel Full Name:** Display the full name of the channel



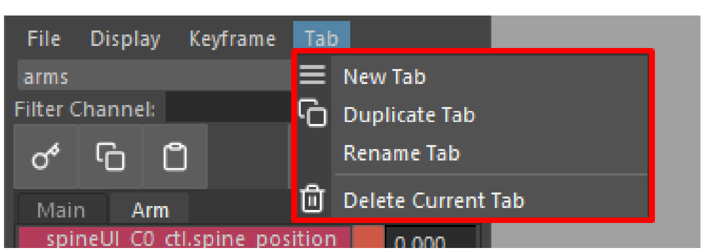
- **Order:** Arrange the channels by Alphabetic or Default order

Keyframe Menu:



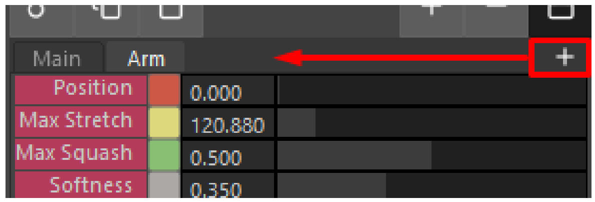
- **Keyframe:** Toggle a keyframe for all the channels in the current tab. If there is a channel that doesn't have keyframe will add keyframe. If all channels have keyframe it will remove the keyframe.
- **Copy key:** Copy current values in the buffer
- **Paste key:** Paste from buffer and set keyframe
- **Delete current keyframe:** Delete the current tab keyframe
- **Keyframe all tabs:** Keyframe command will be applied to all tabs. With this option active the toggle functionality will change to always key.

Tabs Menu:



- **New Tab:** Create new tab.
- **Duplicate Tab:** Duplicate current Tab
- **Rename Tab:** Rename current tab
- **Delete Current Tab:** Delete current Tab

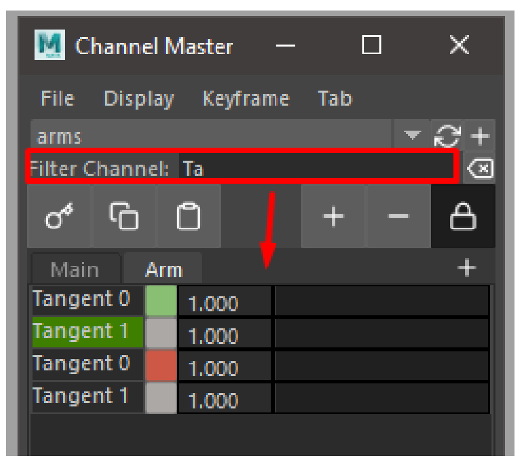
Tabs can be added using the + icon in the tabs row



6.1.1 Channels

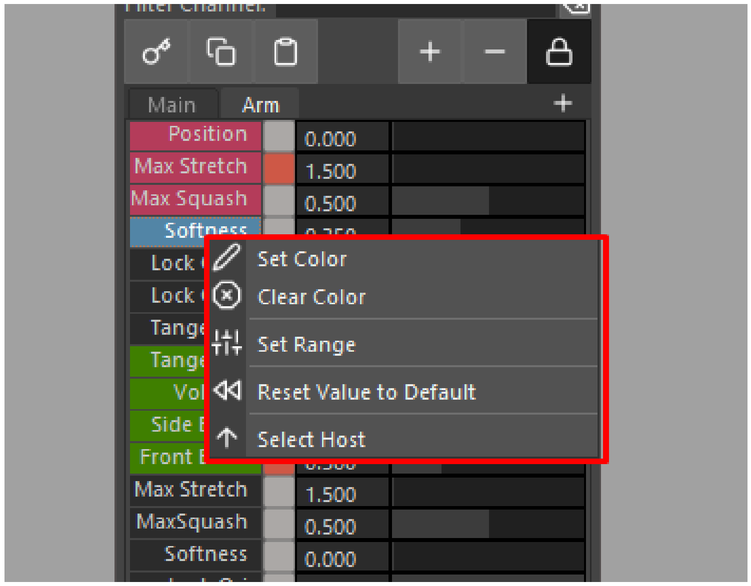
1) **Keyframe button:** Same command as keyframe menu
2) **Copy key button:** Same command as keyframe menu
3) **Copy paste button:** Same command as keyframe menu
4) **Plus Button:** Add selected channels from ChannelBox to Channel Master
5) **Minus Button:** Remove SelecteT channels
6) **Lock refresh:** in Main Tab

Search channel: will filter the channel list



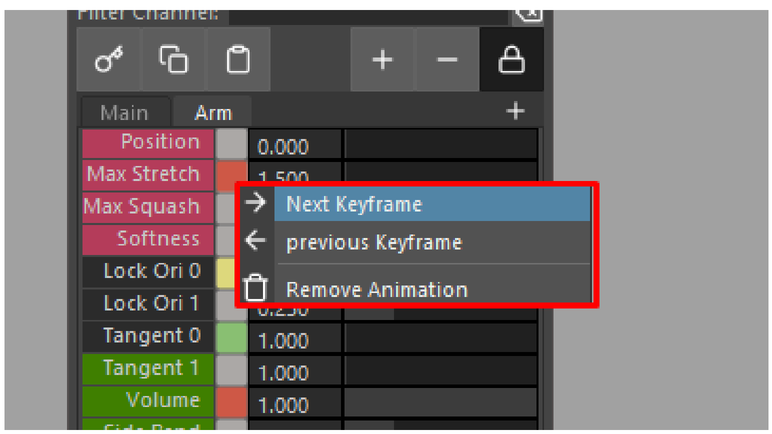
In the channels area we will find 2 context menus and a middle click slider precision widget.

Channel context menu: The actions in this menu can be apply to multiple selection



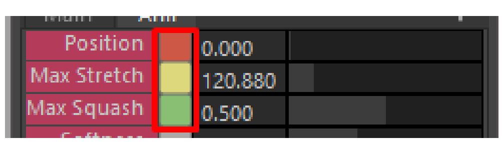
- **Set Color:** Set a custom color for easy identification.
- **Clear Color:** Clear custom color.
- **Set Range:** Set range min and max for the slider.
- **Reset Value to Default:** Reset the channel value.
- **Select Host:** Select the object that owns the channel.

Keyframe channel button context menu.:



- **Next Keyframe:** Move time to the next keyframe.
- **Previous Keyframe:** Move time to the previous keyframe.
- **Remove Animation:** Delete channel animation.

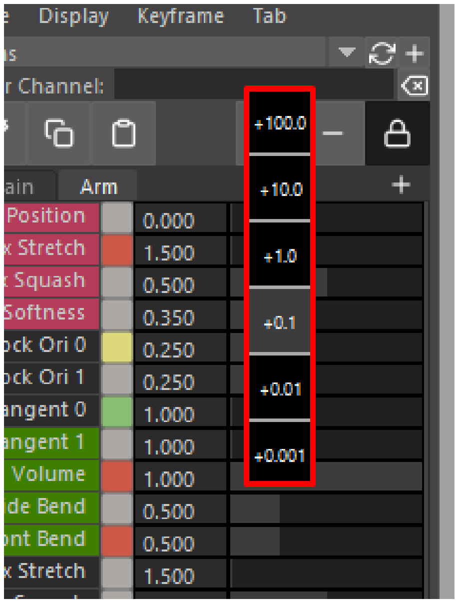
Keyframe button color coding:



- **Red:** Keyframe in the current frame.

- **Green:** Animation but not keyframe in the current frame.
- **Yellow:** Current value changed.

The slider precision widget is similar to the one from Houdini

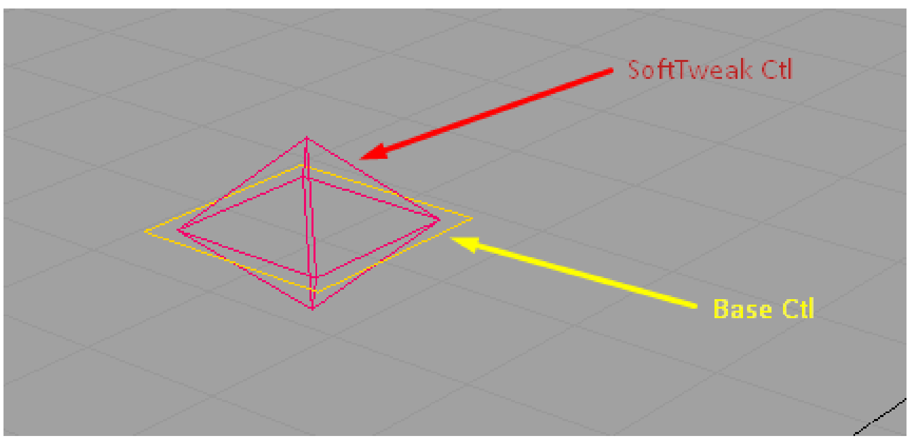


6.2 Soft Tweaks

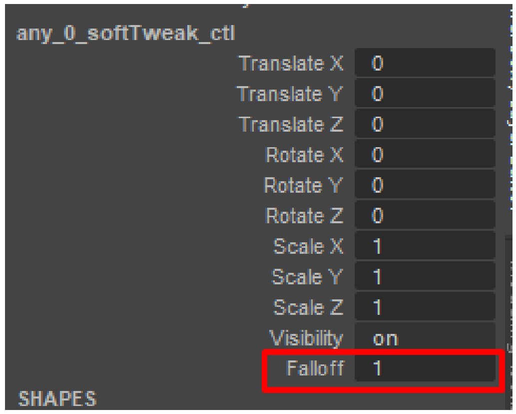
This tool create and manage SoftTweaks (ST for short) and also provide as simple API to import and export configurations, so is possible to integrate with your pipeline and animation publishing system.

SoftTweak is a dynamic position tweak using softmod deformer The credit of the idea goes to Vasil Shotarov , thanks!

The SoftTweak have 2 controls. The Base represents the bind position where the ST will not have any effect. The main control will trigger the deformation when is not in the reset position..

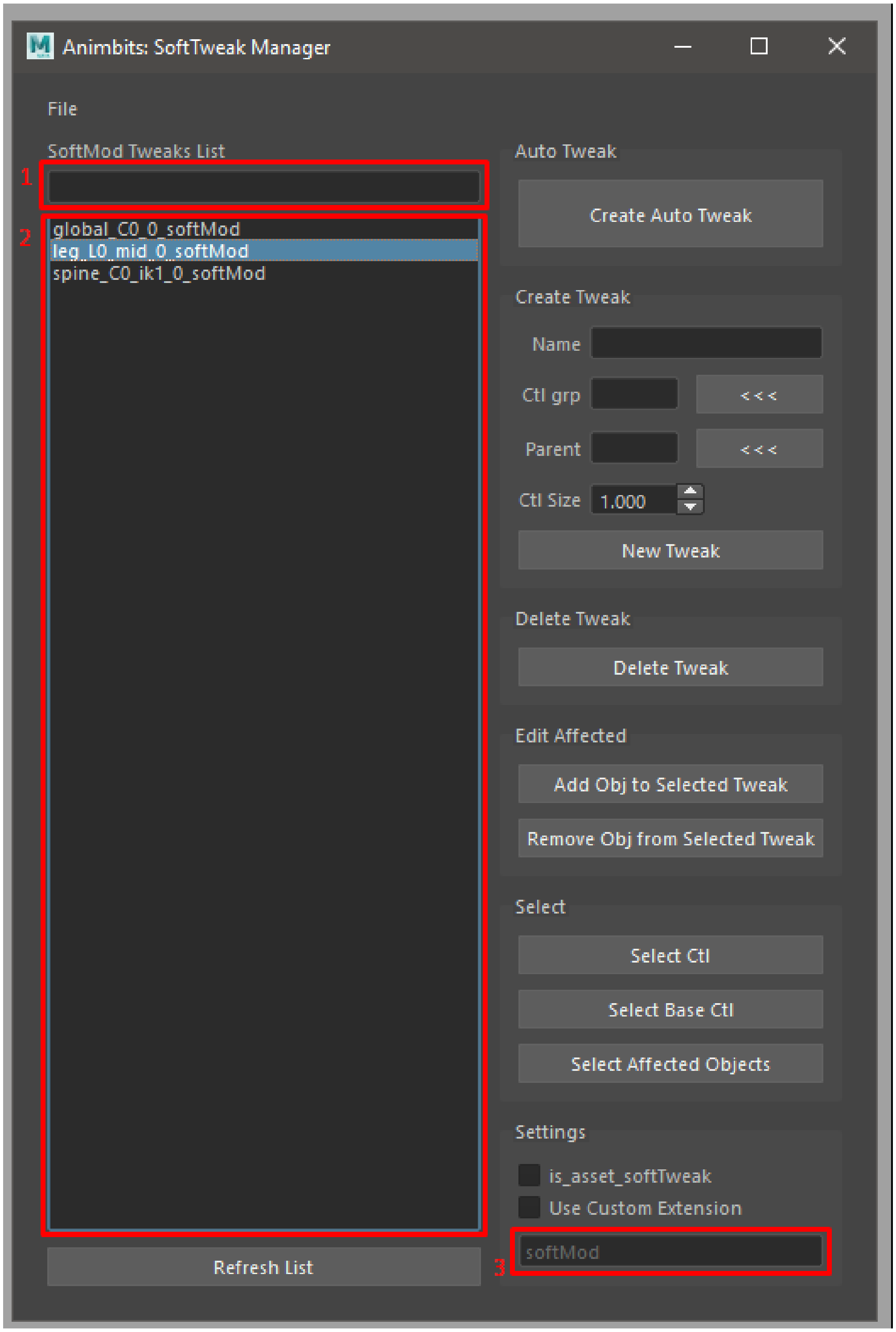


In the ST main control we can find a falloff channel to control the area of deformation



6.2.1 SoftTweak Manager GUI

By default the SoftTweak is design to be used by shot and not part of a rig. However if is needed to publish a rig with some SoftTweaks there is an option to flag (is_asset_softTweak) the SoftTweak as asset tweak. So the tool will make a distinction between the ones that need to be publish with the shot and the ones that are included in the asset.



File Menu:

- **Export Selected:** Exports selected ST from the list
- **Export All:** Exports all the ST from the list. If search filter is used, will not affect the export. But export will make a distinction between regular SoftTweak and asset SoftTweak
- **Import:** Import ST configuration from a file.

GUI:

- **Search Filter:** (1) Quick search filter of the SoftTweak list.
- **SoftTweaks List:** (2) ST selection list.
- **Create Auto Tweak:** Create a tweak based on the current selection.
  1. Select the objects to apply the ST
  2. Last element selected should be a control. This will be the parent of the ST. Also the ST will take the Name and the group(set) from the parent
  3. Click Crete Auto Tweak
- **Name:** Name of the new ST.
- **Ctl grp:** Group of controls to add the new ST controls.
- **Parent:** Parent of the new ST.
- **Ctl Size:** Parent of the new ST.
- **New Tweak:** Create a new ST.
- **Delete Tweak:** Delete the tweaks selected in the ST list.
- **Add Object to Selected Tweak:** Add selected objects to the tweaks selected in the ST list.
- **Remove Object from Selected Tweak:** Remove selected objects from the tweaks selected in the ST list.
- **Select Ctl:** Select the control from the tweaks selected in the ST list.
- **Select Base Ctl:** Select the Base control from the tweaks selected in the ST list.
- **Select Affected Objects:** Select the object affected by the the tweaks selected in the ST list.
- **is_asset_softTweak:** Tag the new created ST as an asset ST. This will also change the ST selection list to show the asset ST.
- **Use Custom Extension:** The new ST will have a custom suffix.
- **Suffix Name:** (3) Suffix name for the new ST.

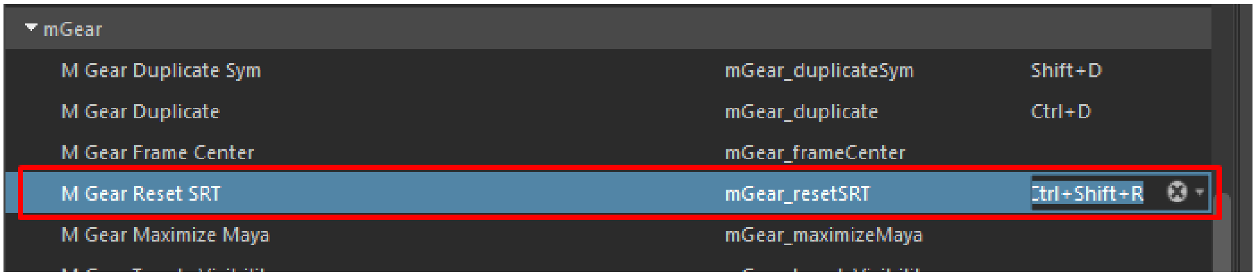
6.2.2 API

```
# To import a softTweak configuration from script editor or Shifter Custom Step:

from mgear.animbits import softTweaks as st
st.importConfigurationFromFile(filePath= path to the .smt configuration file)

# to export configuration
# list the softtweaks in the scene
softtweaks = st._listSoftModTweaks(is_asset=False)
# export
st.exportConfiguration(softtweaks)
```

6.3 Smart reset Attribute/SRT

This command will reset the SRT (Scale, rotation and Translation of any selected object). If an attribute is higlighted in the Channel Box, will reset the channel instead.

TIP: Set a hotkey for this command using mGear Hotkey creator in utilities menu.

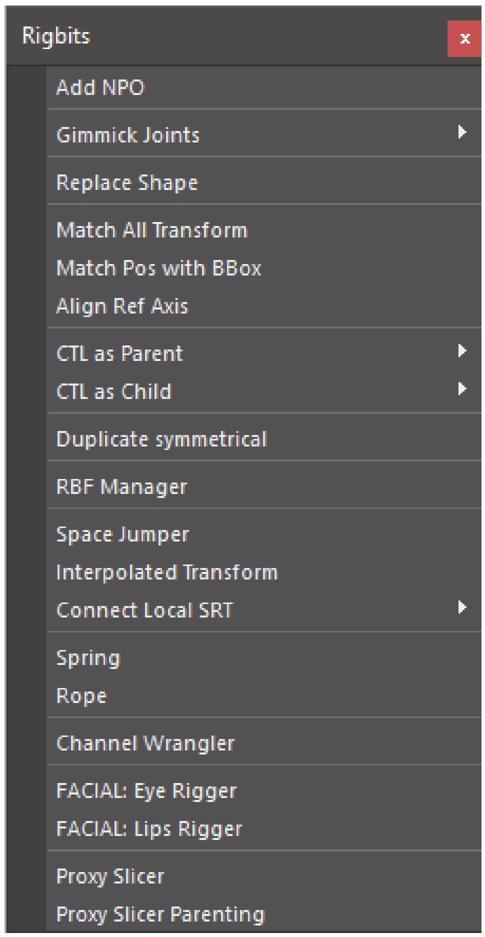


CHAPTER

SEVEN

RIGBITS USER DOCUMENTATION

Rigging tools



7.1 Add NPO

Add a transform as parent of each selected object in order to neutralize the local values to the reset position

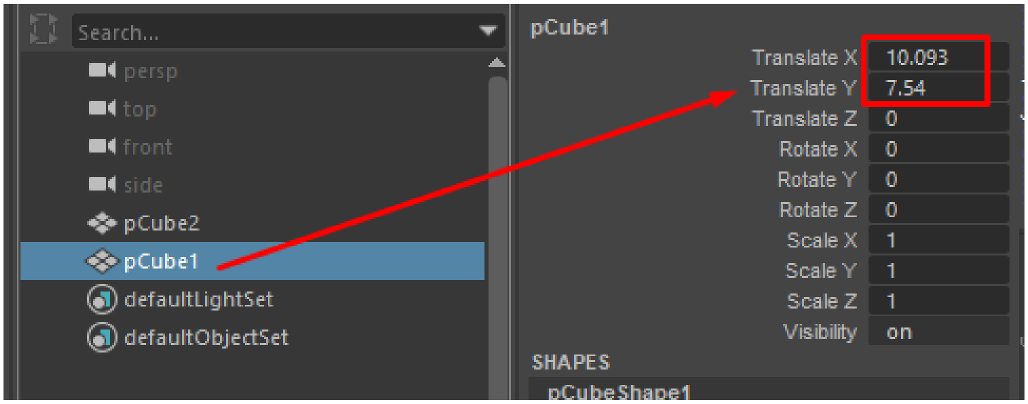


Fig. 1: The tranlate X and Y have some values

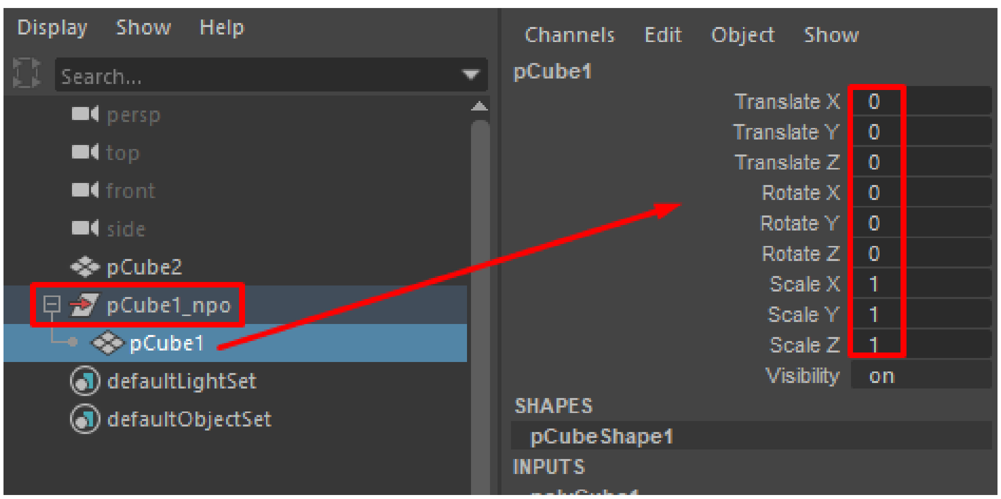


Fig. 2: All the local transform values are reset

7.2 Gimmick Joints

Joint helper tools.

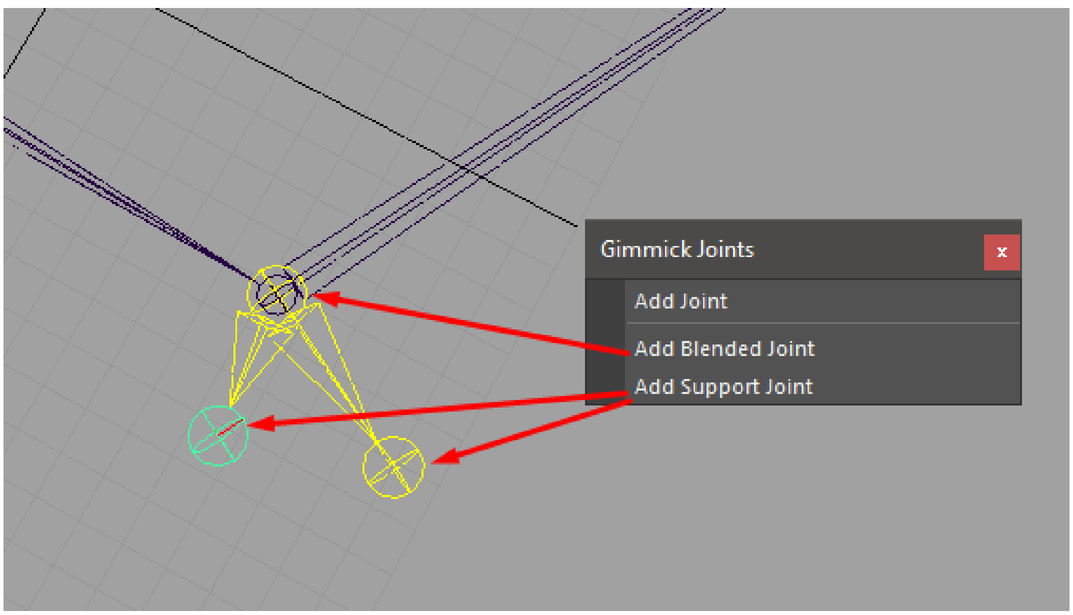
7.2.1 Add Joint

Add a deformer joint to each selected object.

This command will try to add the joint to "rig_deformers_grp" or create it if doesn't exist. Also will parent the joint under "jnt_org" if exist. If doesn't exist will parent the joint under the corresponding object.

7.2.2 Add Blended Joint

Add a blended joint under a chain of joints. This joint will rotat 50% between 2 joints.



7.2.3 Add Support Joint

Support joint are created under a blended joint and are design to help with deformation. Typically this kind of joints will also be driven by a SDK or similar.

7.3 Replace Shape

Replace the shape of one object shape with another

7.4 Match All Transform

Align one object to another object using the world Matrix reference.

7.5 Match Pos with BBox

Center the position of an object in the center of the bounding box of an object.

7.6 Align Ref Axis

Create a reference locator axis based on a point selection. This command needs at less 3 points.

Tip: Very useful to find rotation axis in mechanical rigs if the transformations of the mesh have been freeze.

7.7 CTL as Parent

Create a control of the selected shape as parent of each selected object.

7.8 Ctl as Child

Create a control of the selected shape as child of each selected object.

7.9 Duplicate Symmetrical

Duplicate and mirror the selected object and his children. This is done by negating some axis scaling and inverting some of the values. This will provide a mirror behavior. Also handle some renaming. i.e: from _L to _R

7.10 RBF Manager

A tool to manage a number of RBF type nodes under a user defined setup(name)

Steps -

set Driver set Control for driver(optional, recommended) select attributes to driver RBF nodes Select Node to be driven in scene(Animation control, transform) Name newly created setup select attributes to be driven by the setup add any additional driven nodes position driver(via the control) position the driven node(s) select add pose

Add notes - Please ensure the driver node is NOT in the same position more than once. This will cause the RBFNode to fail while calculating. This can be fixed by deleting any two poses with the same input values.

Edit Notes - Edit a pose by selecting "pose #" in the table. (which recalls recorded pose) reposition any controls involved in the setup select "Edit Pose"

Delete notes - select desired “pose #” select “Delete Pose”

Mirror notes - setups/Controls will succefully mirror if they have had their inverseAttrs configured previously.

7.11 Space Jumper

7.12 Interpolate Transform

7.13 Connect Local SRT

7.14 Spring

7.15 Rope

7.16 Channel Wrangler

7.17 Eye Rigger

7.18 Lips Rigger

7.19 Proxy Slicer

7.20 Proxy Slicer Parenting

CHAPTER

EIGHT

UEGEAR USER DOCUMENTATION

ueGear are a collection of tools that will allow for faster workflows between Maya and Unreal.

8.1 Unreal Setup

In order to utilise ueGear, you will need to install the **Unreal ueGear plugin**, or download the **ueGear repo** and build the plugin.

- Unreal Plugin
- Github Repo

Once the plugin is installed, launch the Unreal project you want to work in and open Maya. You can now use ueGear.

8.2 Menu

The menu can be found under **mGear > ueGear**

As Unreal has multiple locations [**Content Browser**, **Active Level**] for assets, do be aware of where you are sending assets to and from.

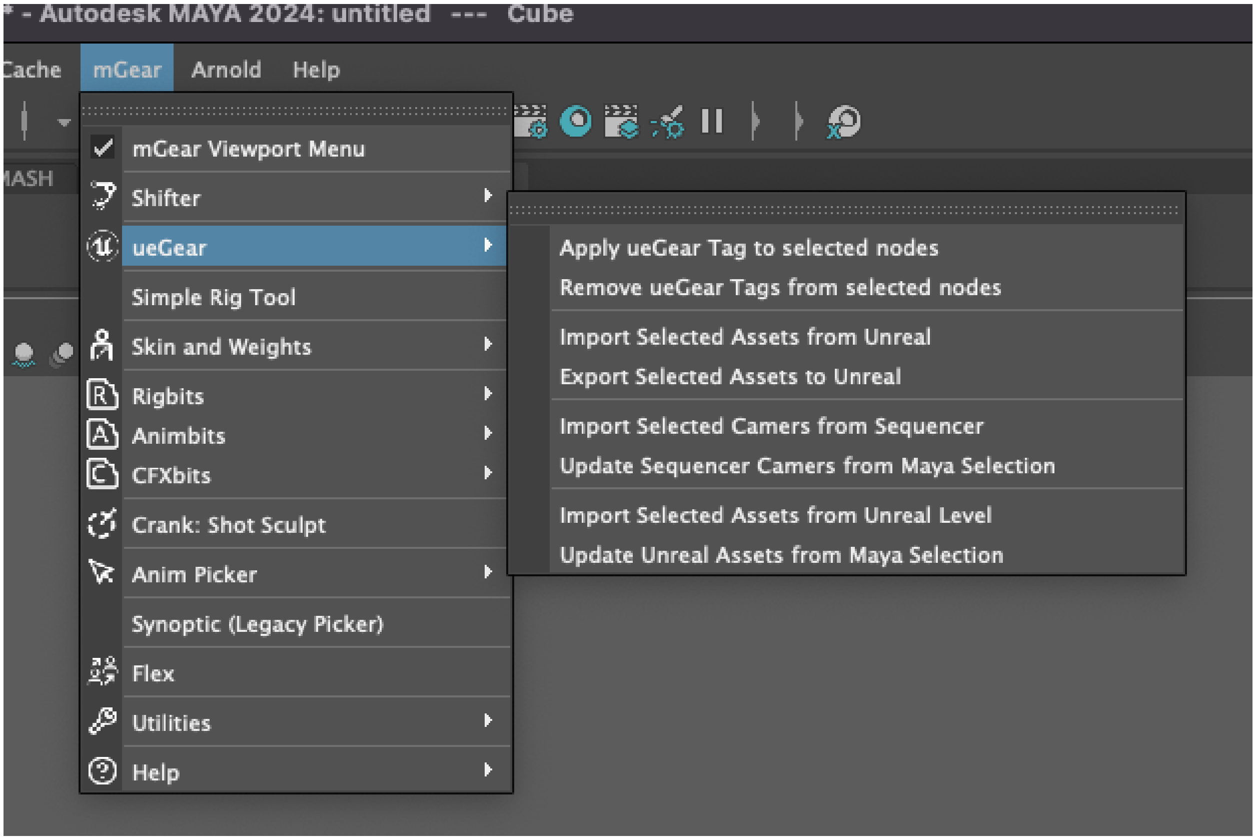
8.3 Unreal Tag

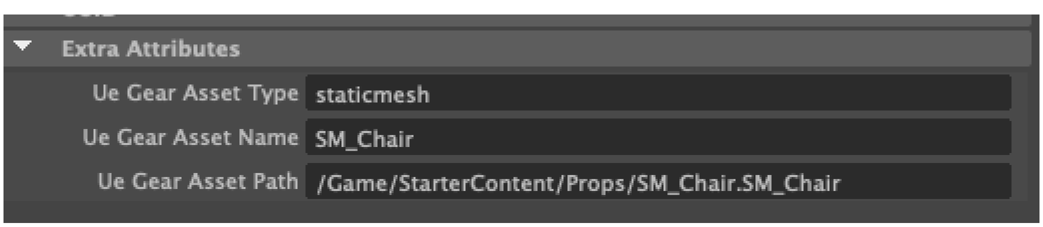
ueGear tags, are extra attributes that are added onto the Maya DAG object as a bunch of extra attributes. These attributes help keep data in sync, with objects that exist in Unreal.

As Unreal allows for multiple objects to be instantiated in a Level and have the same name, we need a way to keep track of which object is associated in Maya, as Maya does not allow for objects to have the same name.

- ue_gear_asset_type: refers to the type of asset this object will be in Unreal.
- ue_gear_asset_name: Name of the asset in Unreal's ContentBrowser
- ue_gear_asset_path: Unreal package path, to where the asset will exist.

Note: For most of the current ueGear tooling this will be **staticmesh**, if you want to export SkeletalMeshes(SKM) then please refer to the **Shifter FBX Exporter**, as that tool has special Unreal intergrations for exporting skeleton and animations into Unreal from Maya.


* - Autodesk MAYA 2024: untitled --- Cube
Cache
mGear
Arnold
Help
MASH
mGear Viewport Menu
Shifter
ueGear
Simple Rig Tool
Skin and Weights
Rigbits
Animbits
CFXbits
Crank: Shot Sculpt
Anim Picker
Synoptic (Legacy Picker)
Flex
Utilities
Help
Apply ueGear Tag to selected nodes
Remove ueGear Tags from selected nodes
Import Selected Assets from Unreal
Export Selected Assets to Unreal
Import Selected Camers from Sequencer
Update Sequencer Camers from Maya Selection
Import Selected Assets from Unreal Level
Update Unreal Assets from Maya Selection


Extra Attributes
Ue Gear Asset Type
staticmesh
Ue Gear Asset Name
SM_Chair
Ue Gear Asset Path
/Game/StarterContent/Props/SM_Chair.SM_Chair

8.4 Assets

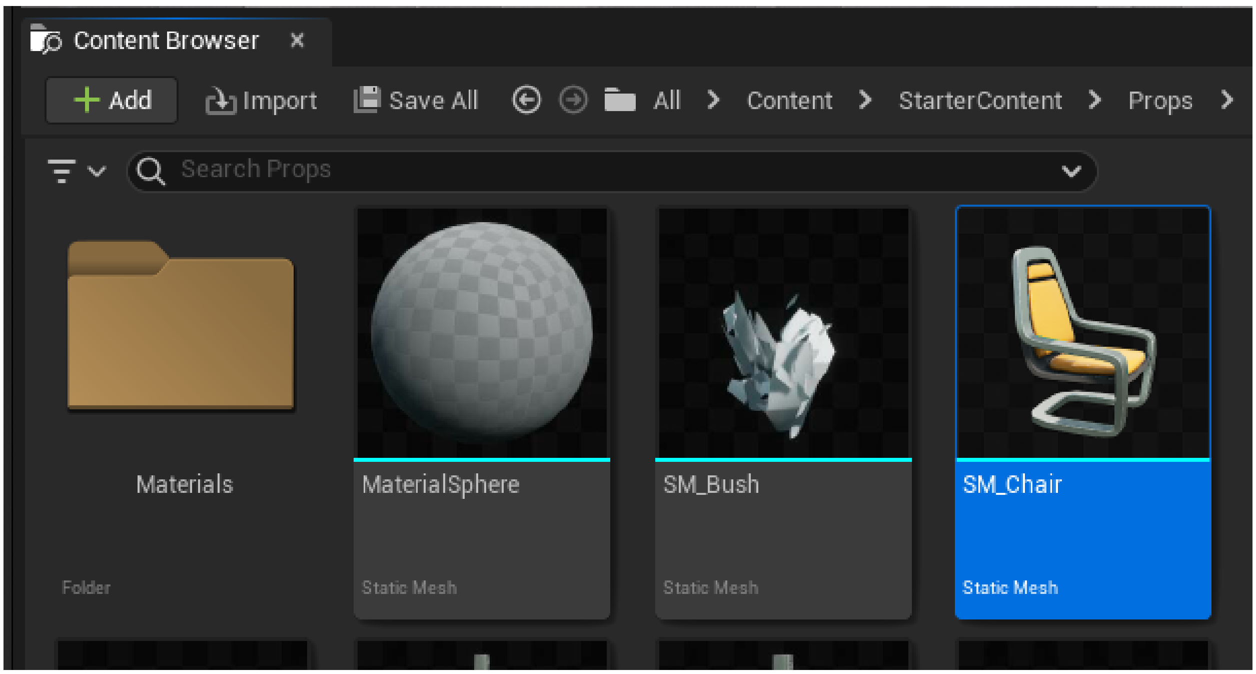
Static Meshes (SM) can be synced between Maya and Unreal.

Note: All assets require a *Unreal Tag* for syncing.

8.4.1 Import

To import an object.

1) Open Unreal



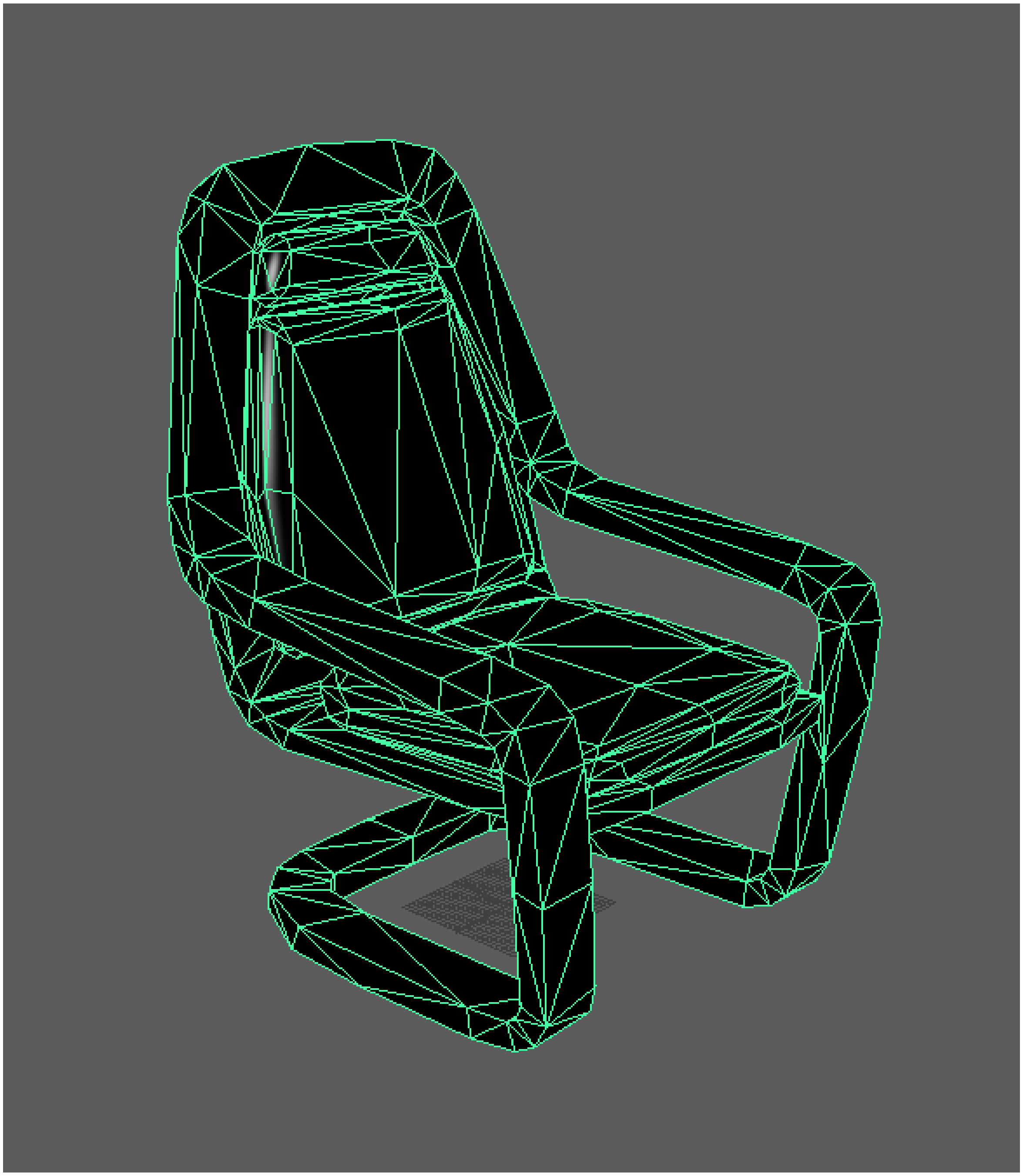
2) Select the object you wish to import in the Content Browser.
3) In Maya click **ueGear > Import Selected Asset from Unreal**
4) The selected asset should now be imported into Maya.

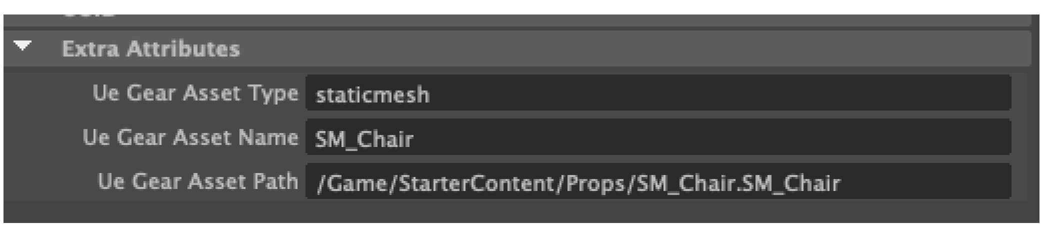
Note: It will not bring in any texture or shading data. It will import the triangulated geo, and setup the tags.

8.4.2 Export

To export an SM from Maya to Unreal

1) Make sure that the DAG has a tag. If one does not exist, create it and populate the *Unreal Tag*
2) Select the asset.
3) In Maya click **ueGear > Export Selected Asset from Unreal**
4) Check Unreals ContentBrowser, the asset should now exist at the *ue_gear_asset_path* location.



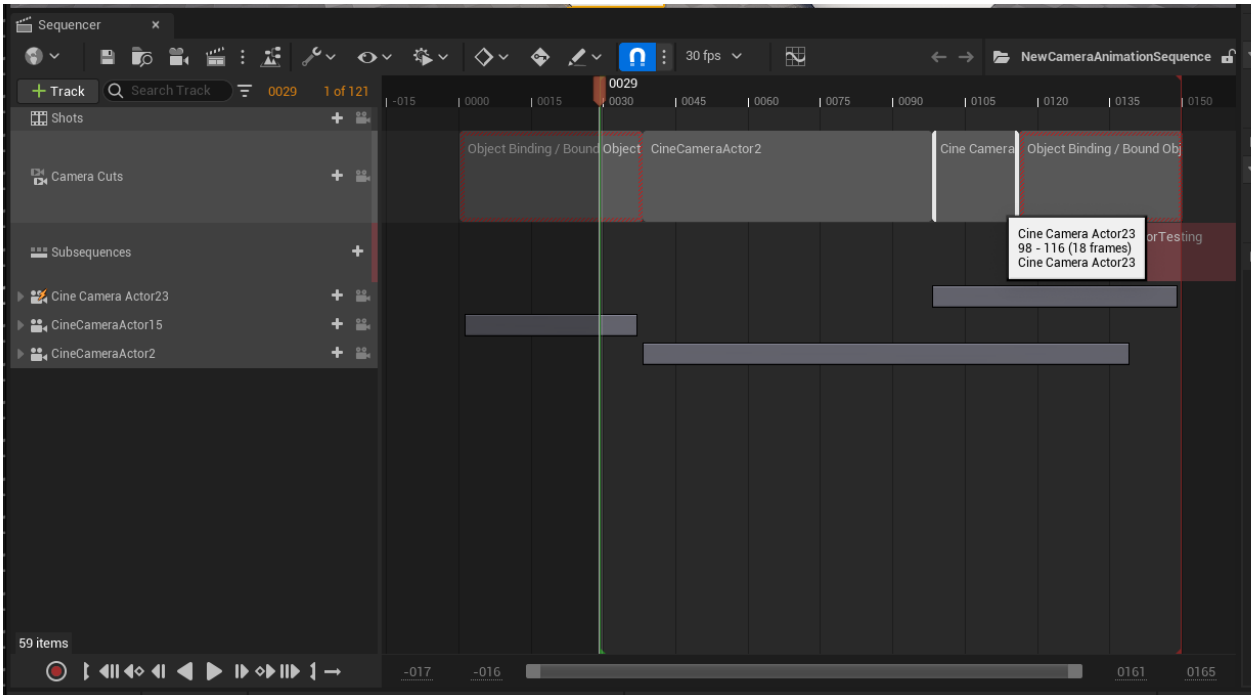

Extra Attributes
Ue Gear Asset Type
staticmesh
Ue Gear Asset Name
SM_Chair
Ue Gear Asset Path
/Game/StarterContent/Props/SM_Chair.SM_Chair

8.5 Cameras

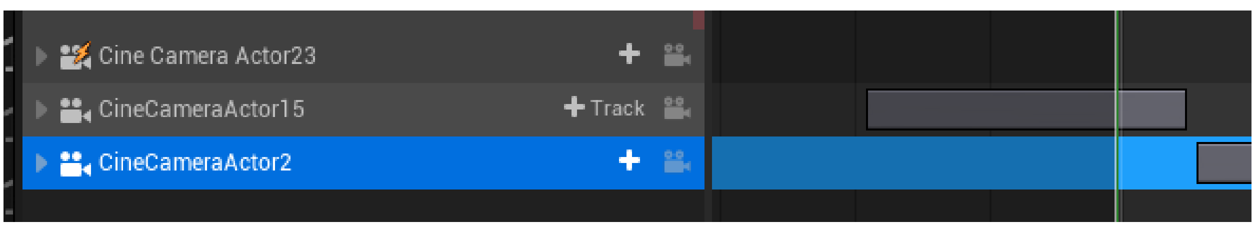
LevelSequence Cameras can be exported into Maya, updated and exported back into Unreal.

8.5.1 Import

To import a Camera into Maya



1) Open up the LevelSequence, which contains the camer you want to export.



2) Select the camera layer. **Note**: Make sure it goes blue and is highlighted, else it will not be detected.
3) In Maya click **ueGear > Import Selected Cameras From Sequencer**
4) The camera is now be imported into your Maya scene.

- Animation on the camera will be baked down per a Frame
- Importing the camera will try and match the FPS of your Unreal Sequencer. If Maya does not match it will warn you and ask if you want to update it Maya's fps.

8.5.2 Export

To update the cameras animation track back to Unreal.

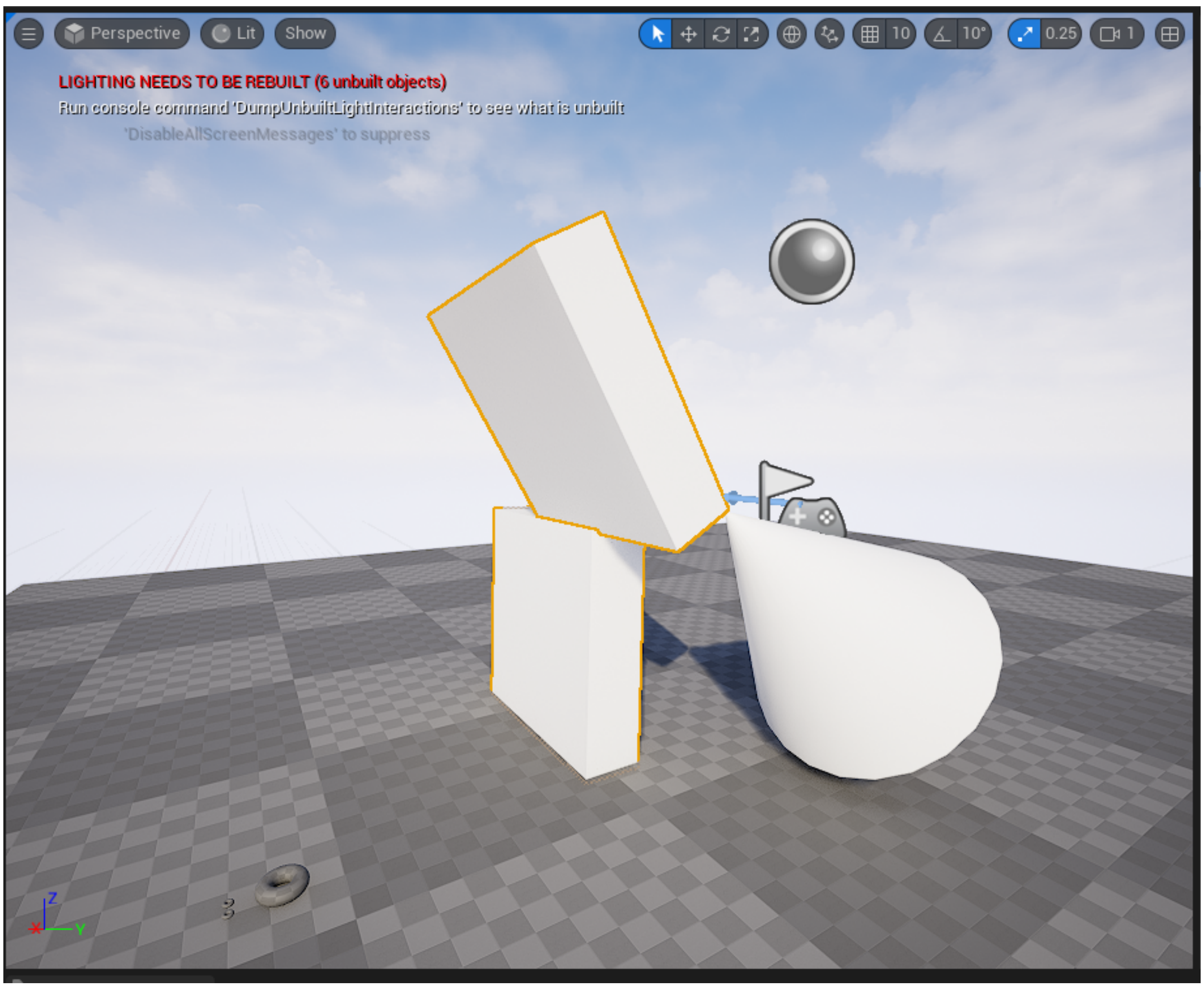
1) Select the Camera in Maya.
2) In Maya click **ueGear > Update Sequencer Cameras From Maya Selection**.
3) The Camera in Sequencer now has the latest keys.

8.6 Layout

ueGear allows you to export assets that are in your Level, directly into Maya, reposition them and push the position changes back to unreal.

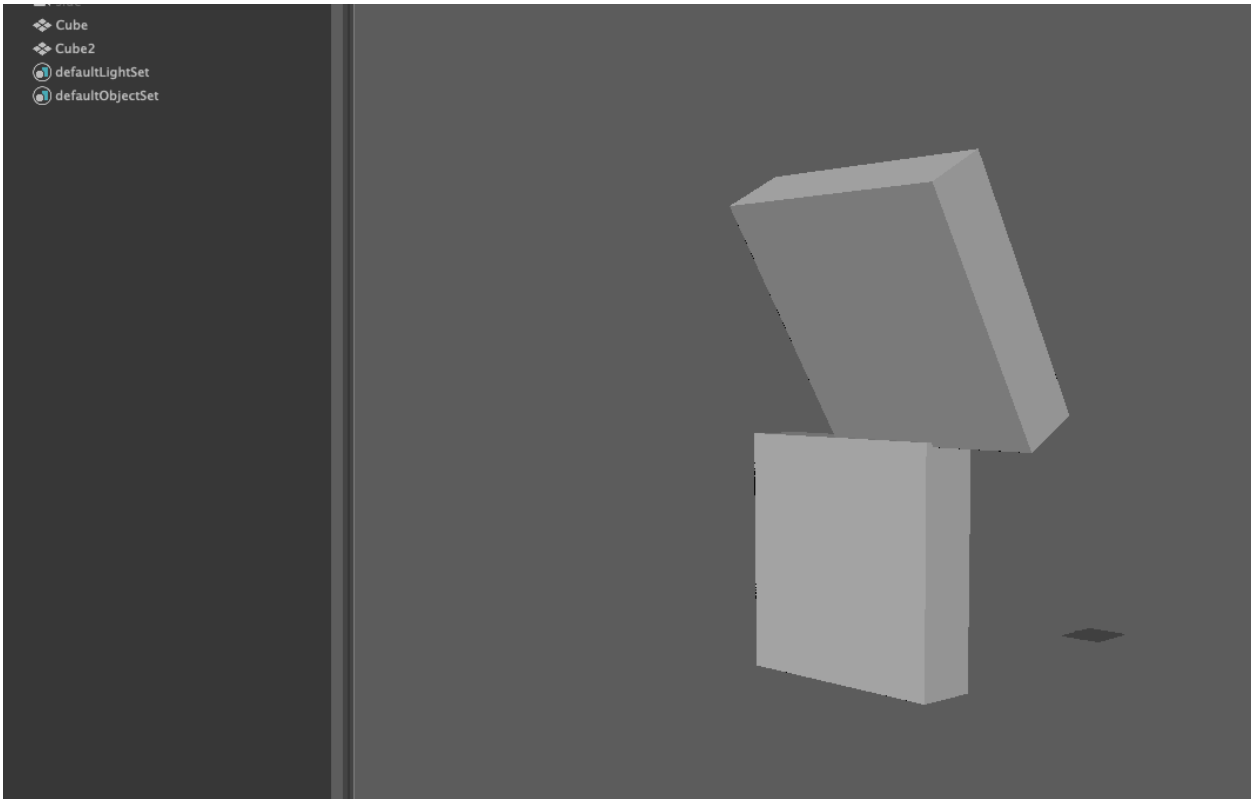
8.6.1 Import

To import an SM from an Unreal Level

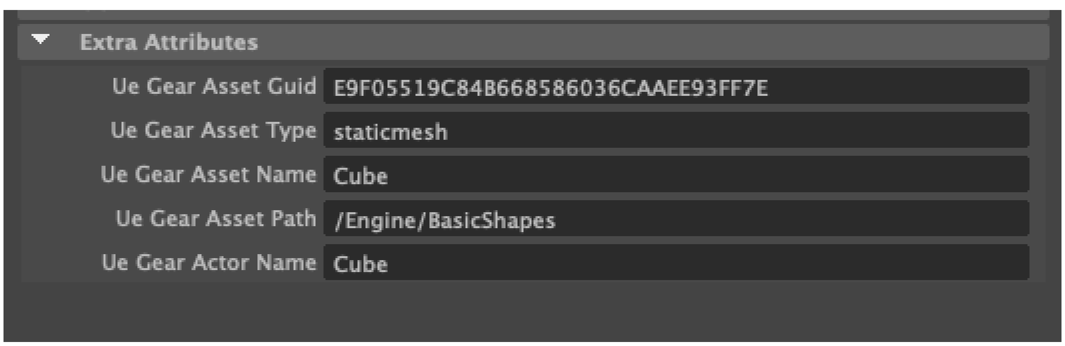


1) Select the asset or assets in Unreal.

2) In Maya click **ueGear > Import Selected Assets from Unreal Level**



The asset should now appear in Maya in the same location as it was in the Unreal Level.



Note: Tags will automatically be generated for each asset, as Unreal has the ability to handle multiple assets with the same name, the tag will store the object guid.

8.6.2 Export

To update the assets position back in the Unreal Level

1) Select the asset or assets in Unreal.
2) In Maya click **ueGear > Update Unreal Assets from Maya Selection**

The transformation data will now be updated in the Unreal Level.

8.7 Skeleton/Rigged Geo

Please see the *Shifter's FBX Exporter*

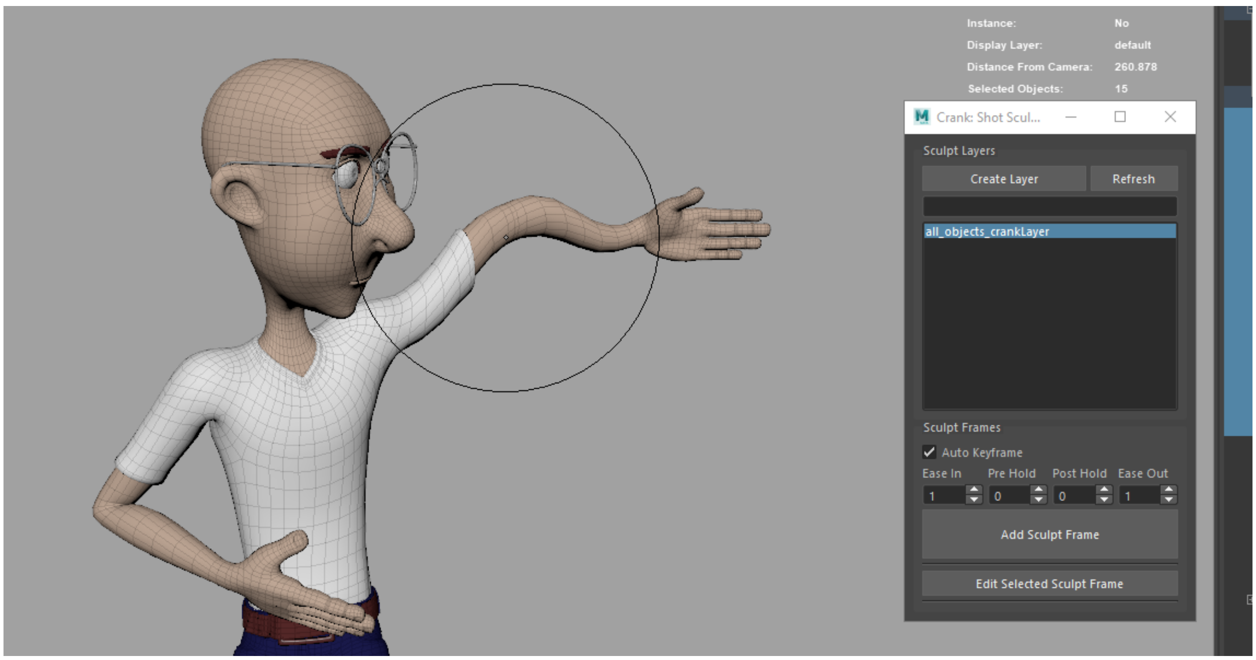
8.8 Animation

Please see the *Shifter's FBX Exporter*

CHAPTER NINE

CRANK USER DOCUMENTATION

Crank is a shot sculpting tool specially designed to handle several object and the same time. This tools can be used for general deformation correction, cloth simulation correction or animation exaggeration.

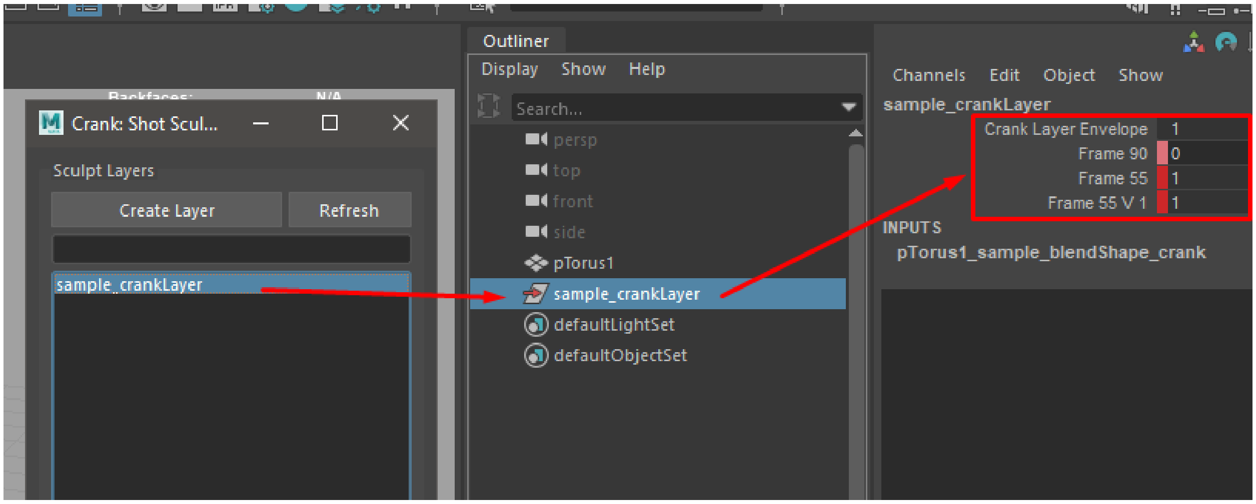


Crank uses only default Maya nodes and blendshape nodes. So not extra plugins are required to open as shot sculpted scene.

Sculpting can be apply to rigged meshes (local or referenced) or to cached animations (Alembic)

In order to deform an object. First we should create a "Crank Layer". The layer can contain one or more objects. Also, an object can be included in more than one layer.

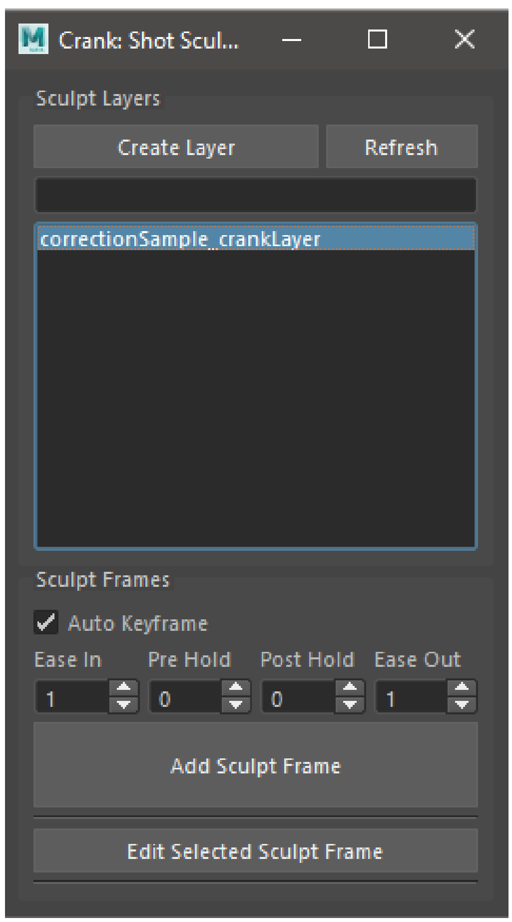
Crank Layers are represented in the scene with a regular Maya's transform node. This node contains a series of custom attributes and connections to handle and store the shot sculpting information and animation.



Several layers can be edit at the same time. But be careful if the same object is in several of this layers.

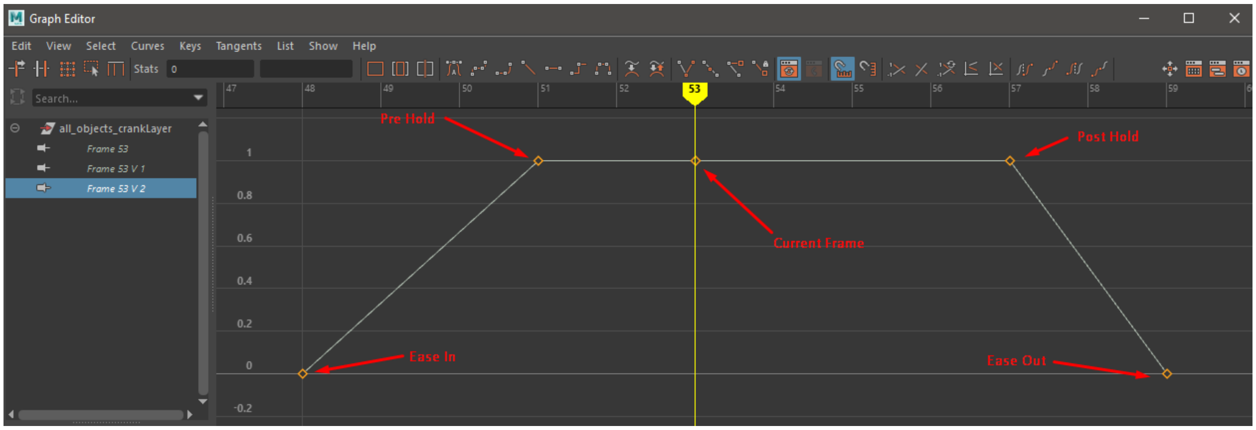
The blendshape nodes are created at the end of the chain and the deltas are set to use tangent space.

This should give the most stable result when the same sculpt is hold in several frames. However in areas where the geometry have collapse a lot (i.e: the internal part of the elbow when the arm is flexed), can result in a unstable interpolation if the sculpt is hold more than one frame. Used in combination with Deltamush can mitigate some of the possible issues



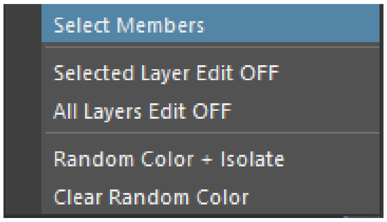
GUI:

- **Create Layer:** Create a new layer from the selected objects.
- **Refresh:** Refresh layers list.
- **Search Filter:** Quick search filter of the Crank Layer list.
- **Layers List: Crank Layers list.**
    - LMB click will select the layer.
    - RMB click show the context menu.



- **Auto Keyframe:** If checked, crank will auto keyframe a range of frames, based on the. Starting from value 0
- **Ease In:** Ease In frames.
- **Pre Hold:** Pre hold frames before current frame. This are the frames where the value of the sculpt blendshape will be 1.
- **Post Hold:** Post hold frames after the current frame. This are the frames where the value of the sculpt blendshape will be 1.
- **Ease Out:** Ease In frames.
- **Add Sculpt Frame:** Add a new sculpt frame for the selected layers. If there is more than one sculpt for each frame it will and a version index. This can be useful to tackle different areas in the same layer.
- **Add Sculpt Frame:** Edit the selected sculpt frame. In order to edit the frame should be first highlighted in the layer channel box. Only one frame can be edited at the same time.

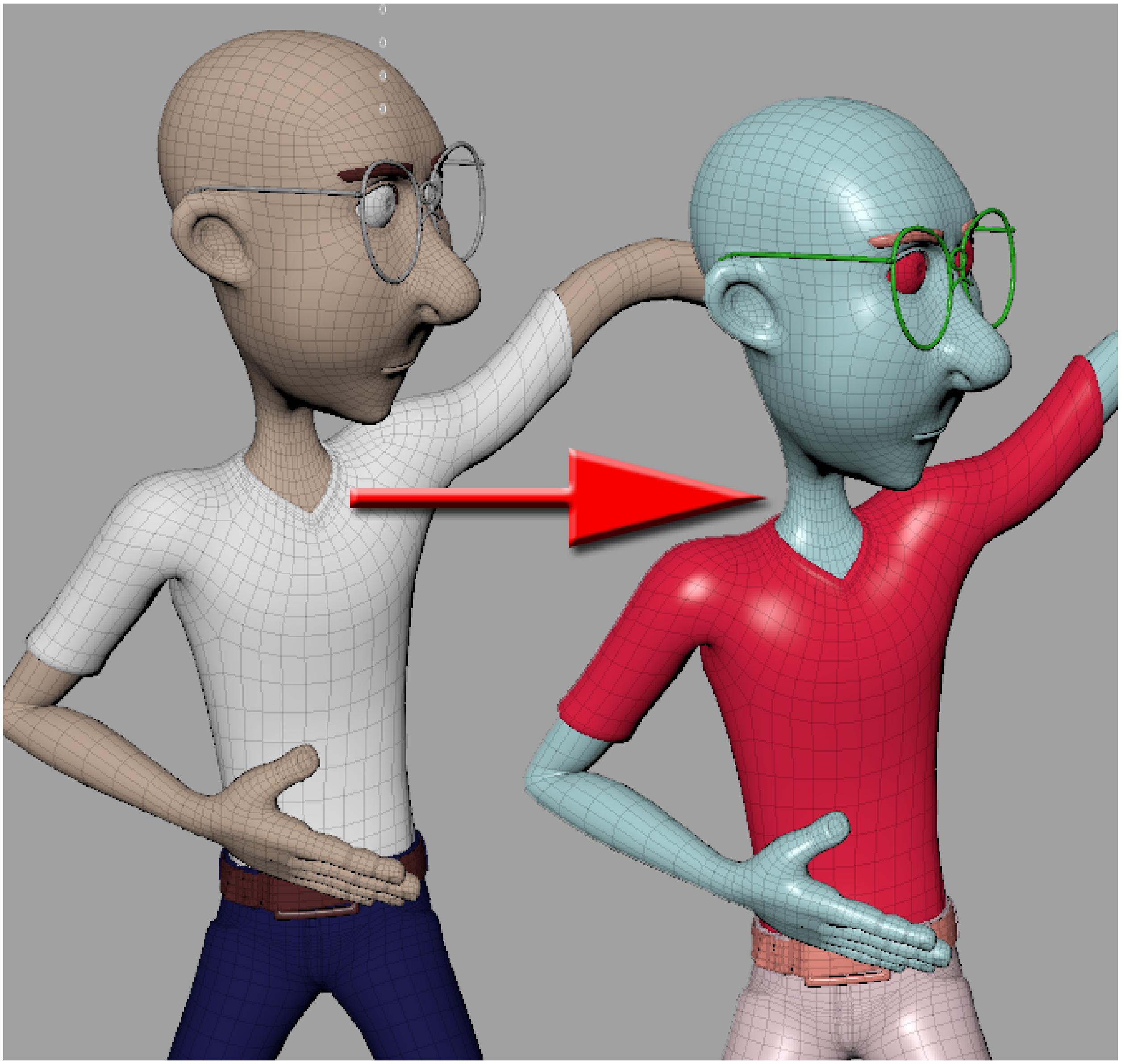
Context Menu:



- **Select Members:** Select the objects affected by the layer.
- **Selected Layer Edit OFF:** Turn off the editing status of the selected layers

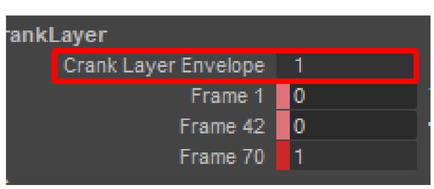
> **Note:** When we add a new sculpt or edit a sculpt frame. The blendshape target of each affected object is set to "Edit" status. To avoid any issue if we forget to set it OFF, a callback is initialized to turn off this edit status if we move the current frame in the timeline.

- **All Layers Edit OFF:** Turn off the editing status of all layers. This is can be also achieved by changing the current frame in the timeline.



- **Random Color + Isolate:** Create a render layer with random color for better visualization. This works based on shading groups, not individual objects.
- **Clear Random Color:** Delete the random color render layer.

WARNING: Currently is not possible to edit (add or remove objects) from a Crank Layer. Also any deleting option have been implemented. in order to deactivate the effect just set the Envelope value of the layer to 0.



CHAPTER

TEN

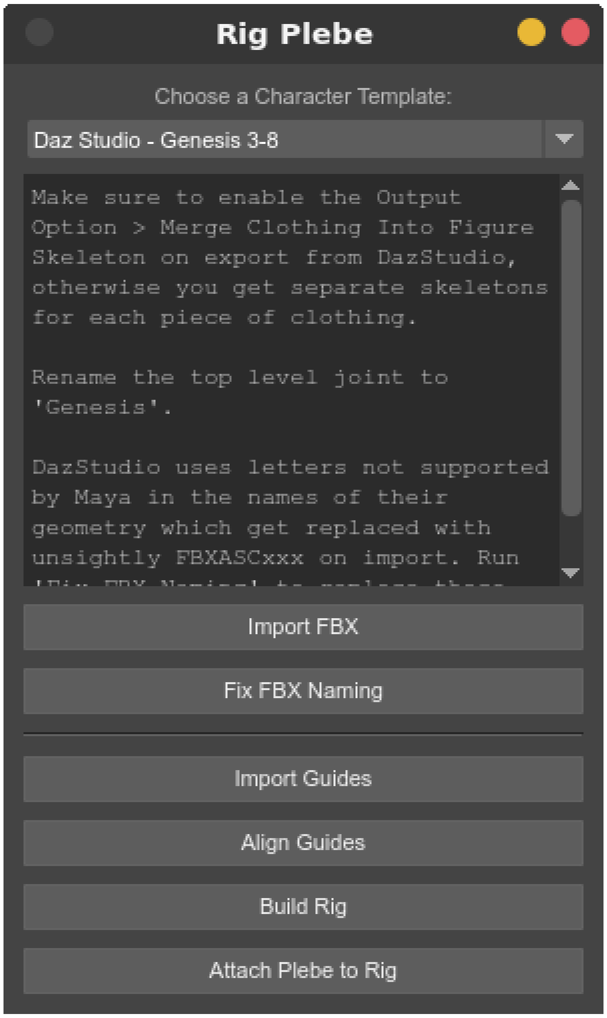
SHIFTER USER DOCUMENTATION

WIP section: Please visit: mGear Youtube channel

- components
- creating new components
- guides templates and basic rig building
- stepped rig building
- scalability and reusability
- gotchas
- tips

10.1 Plebes - Instant Rigged Characters Using mGear

Plebes is a simple template based tool to quickly rig characters from various character generators, such as DazStudio, Character Creator 3, MakeHuman or Mixamo. What it does is build an mGear rig that matches the proportions of your character, and then constrains the joints of that character to the rig with just a few clicks.



10.1.1 How to Rig Plebes

1) Open Plebes interface from the mGear>Shifter>Plebes… menu.
2) Export the character from the character generator as FBX and bring it into Maya.
3) Select the **Character Template** that matches your character generator.
4) Follow the template specific instructions in the Plebes interface.
5) Press **Import Guides** to import the mGear biped guides.
6) Press **Align Guides** to align the mGear guides to your character's joints.
7) Look over the guides, and manually adjust any that are off (e.g. typically the heel and sides of the feet).
8) Press **Build Rig** to build the rig.
9) Press **Attach Plebe to Rig** to constrain the character to the mGear rig. This also removes any locks, keys/connections and/or limits on the translate, rotate and scale attributes on the character's original joints.

You can delete the rig, adjust the guides and rebuild it, like you can normally with mGear, by simply deleting the "rig" group and running the last two steps again.

Note: Some character generators build their characters with completely straight or misaligned elbows and knees, which makes it impossible for mGear to figure out where to aim the knee or elbow, so you may need to rotate the joints slightly before aligning the guides to them, to make sure they are pointing in the right direction.

10.1.2 Known Limitations

Plebes is meant to quickly rig generic characters, typically for use in the background or for crowd agents, so has some limitations. If you need more of a hero rig, you can use the guide placement as a starting point, but it's probably a good idea to skin the character directly to your mGear joints, rather than using **Attach Plebe to Rig**. Other known limitations include:

- Stretching and scaling of limbs may not work correctly for all templates, though it should work fine for all "normal" animation.
- Some characters come with additional joints, such as face joints, that Plebes does not add any controls to.

10.1.3 Plebe Templates

What gets aligned and constrained to what is defined by simple JSON templates. Plebes ships with templates for the several commonly used character genrators, but should you want to add more or modify the existing ones, you can easily do so. You can define the location of additional templates by defining the environment variable PLEBE_TEMPLATES_DIR. You can have multiple template dirs, so you can add your custom ones from your home folder or project specific ones as needed, just make sure each tamplate has a unique name.

The templates look like this:

```
{
    "help": "This show up when you hover over the template menu.",
    "root": "CC_Base_BoneRoot",
    "guides": [
        {"guide": "CC_Base_BoneRoot"},
        {"neck_C0_tan0": [
            "CC_Base_NeckTwist01",
            "CC_Base_NeckTwist02"
        ]}
    ],
    "settings": [
        {"arm_L0_root": [
            { "div0": 1 },
            { "div1": 1 },
            { "supportJoints": 0}
        ]}
    ],
    "joints": [
        {"local_C0_ctl": {
            "joint": "CC_Base_BoneRoot",
            "constrain": "111"}
        },
        {"spine_C0_0_jnt": {
            "joint": "CC_Base_Hip",
            "constrain": "110"}
```

(continues on next page)

(continued from previous page)

```
        }
    ]
}
```

- **help** - Documentation that shows up in the interface, detaling any specific things you need to do to work with this template.
- **root** - The top level joint/node from the character generator.
- **guides - List of which guides to position at which joints.**
    - If you match it to a list of joints, like with the neck above, it will be placed between them.
- **settings** - Settings to adjust on the guides before building the rig. Typically this is number of twist joints, but can be any attribute and value combination.
- **joints - List of mGear joints and which of the character's joints to constrain to it.**
    - **joint** - Name of the character's joint to constrain to mGear.
    - **constain** - Three 0 or 1's. First is if to point constraint, second is orient and third is scale.

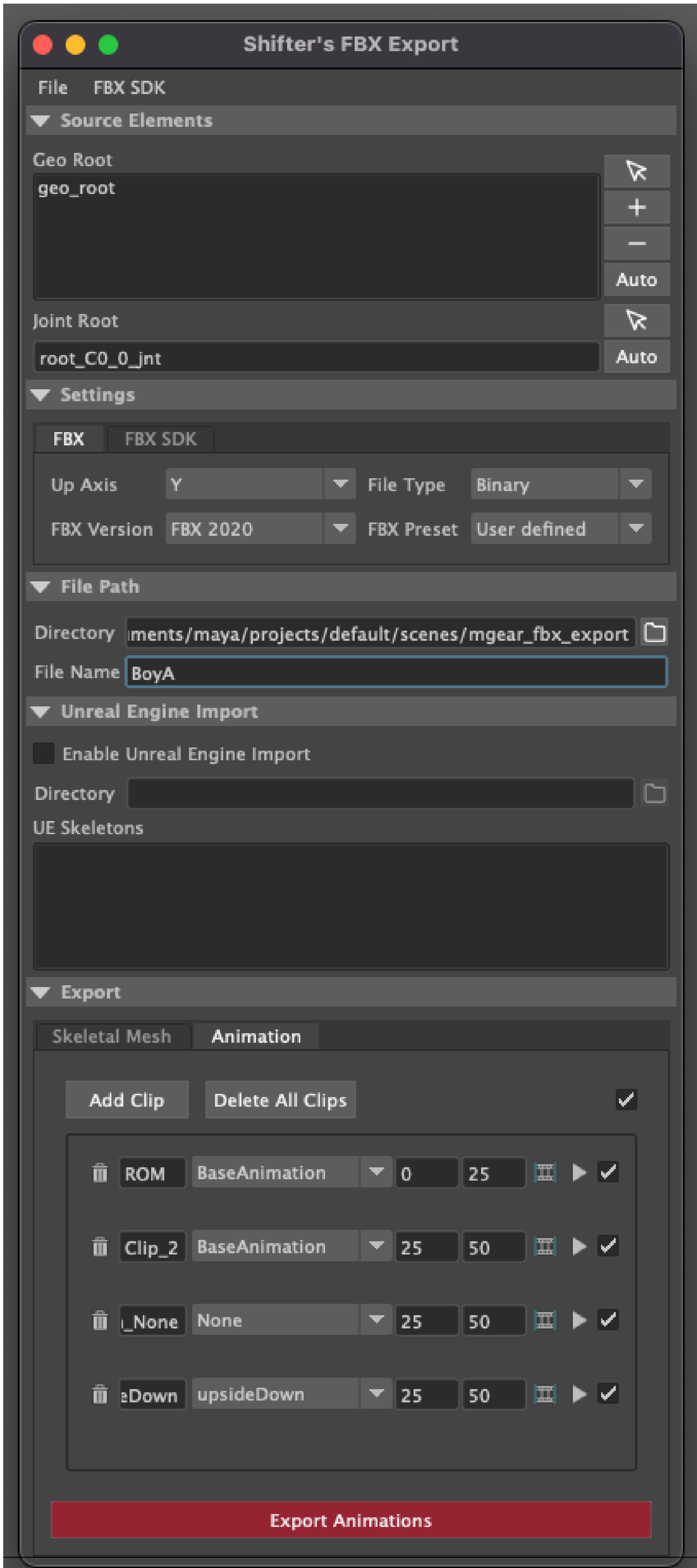
10.2 Shifter's FBX Exporter

The FBX exporter, allows you to export FBXs, as well having an integration into Unreal.

It supports the following Unreal processes:

- Exporting SKMs
- Exporting SKMs and using existing Skeletons
- Exporting Animation
- Exporting Animation layers

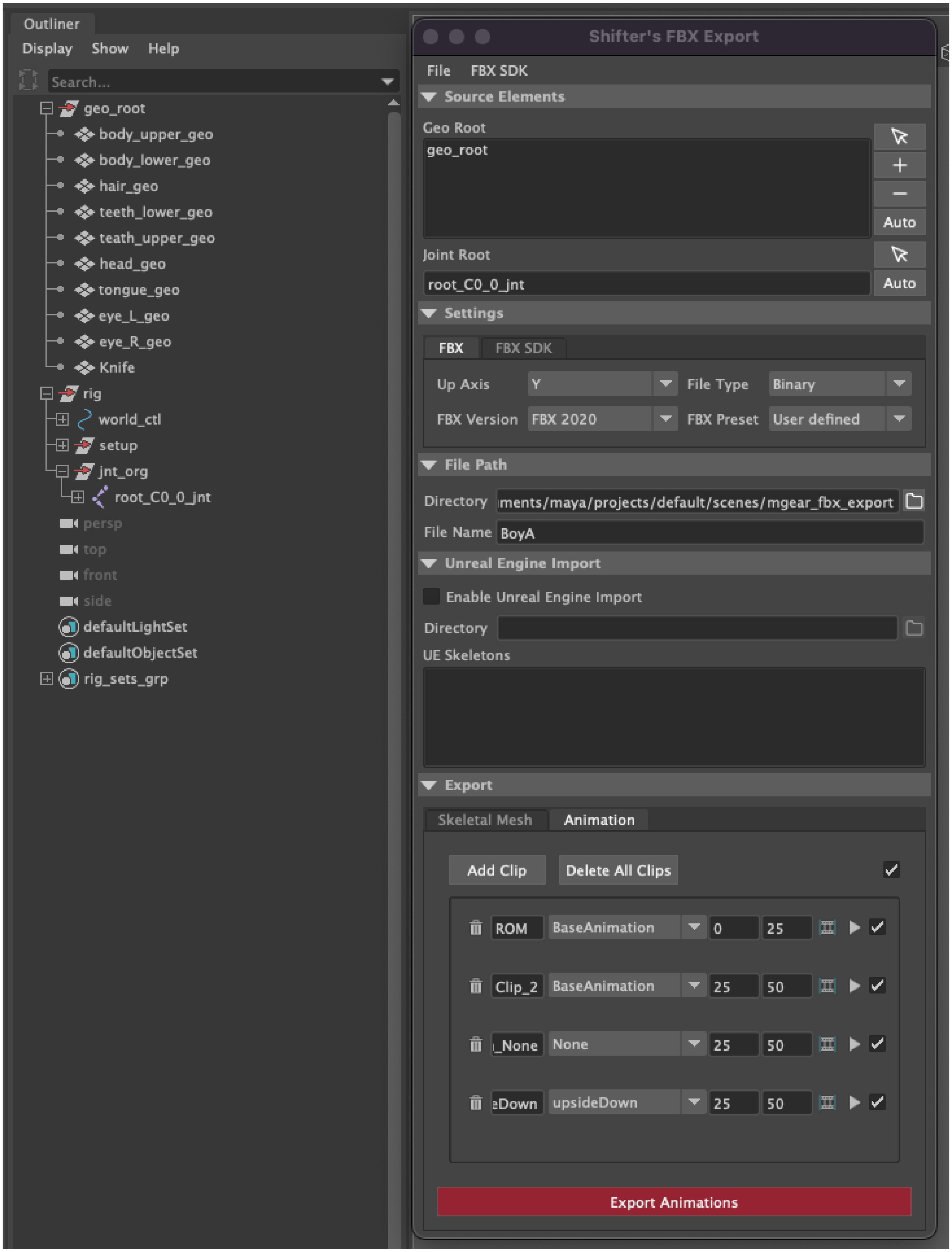
10.2.1 UI


Shifter's FBX Export
File
FBX SDK
Source Elements
Geo Root
geo_root
Auto
Joint Root
root_C0_0_jnt
Auto
Settings
FBX
FBX SDK
Up Axis
Y
File Type
Binary
FBX Version
FBX 2020
FBX Preset
User defined
File Path
Directory
ments/maya/projects/default/scenes/mgear_fbx_export
File Name
BoyA
Unreal Engine Import
Enable Unreal Engine Import
Directory
UE Skeletons
Export
Skeletal Mesh
Animation
Add Clip
Delete All Clips
ROM
BaseAnimation
0
25
Clip_2
BaseAnimation
25
50
_None
None
25
50
eDown
upsideDown
25
50
Export Animations

- **File** - Allows for users to serialise there settings, incase they want to reload them, or use them in a scripted pipeline.

Source Elements

These are the elements that make up the FBX, Geometry and Skeleton structure.



In the image above you can see that the **geo_root** is a group that contains all the geometry objects.

Geo Root: The list of geometry object roots. There can be more then one depending on how you have structured your character.

Joint Root: The root bone of the skeleton.

Settings

You can specify the FBX export settings here.

It also allows for some extra conditioning of the data upon export.

- Remove Namespace
- Clean up scene.

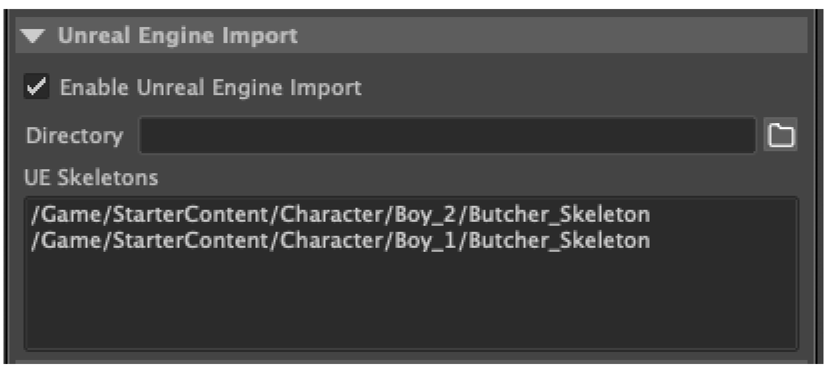
File Path

- **Directory**: Location of the exported FBX files.
- **File Name**: Name of the fbx file that will be generated. This will also be used as the name of the **Unreal Assets**.

Unreal Engine Import

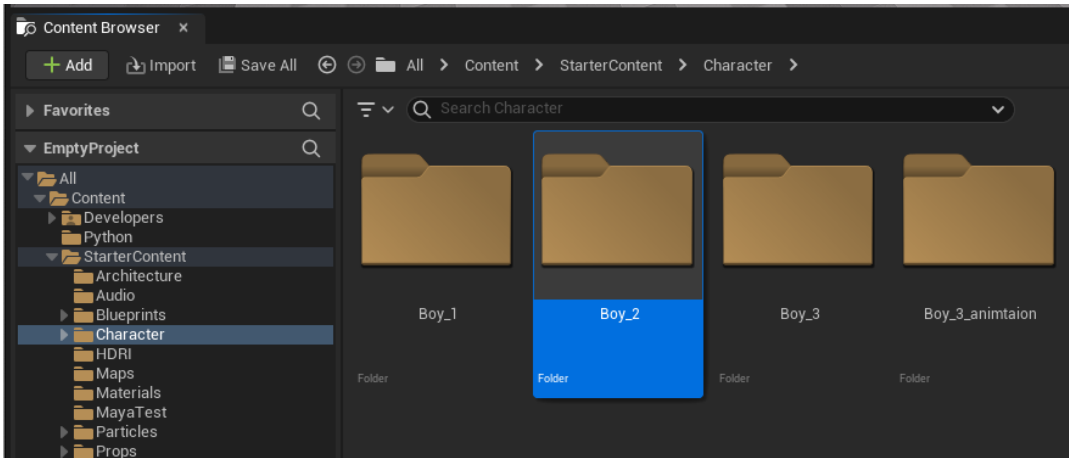
- **Enable Unreal Engine Import**: Enabling this, will allow for the other Unreal UI elements to become active.

It also **refreshes** the **Unreal Skeleton** list, by querying the current open Unreal Project.

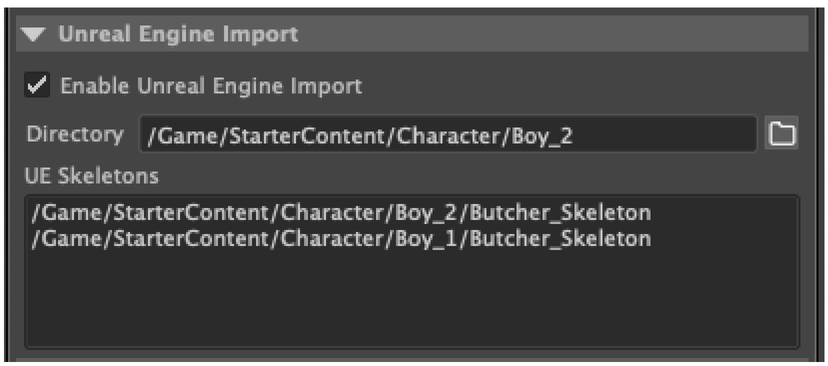


Note: If you have selected a skeleton and no longer what to import using the selected skeleton, or wish to generate a new skeleton on import, ctr click on the selected element to deselect it.

- **Directory**: The import location in Unreal for the SKM and Animations.



1) Navigate to the folder in Unreal that you want to import to.
2) Select the folder in Unreal's **Content Browser**

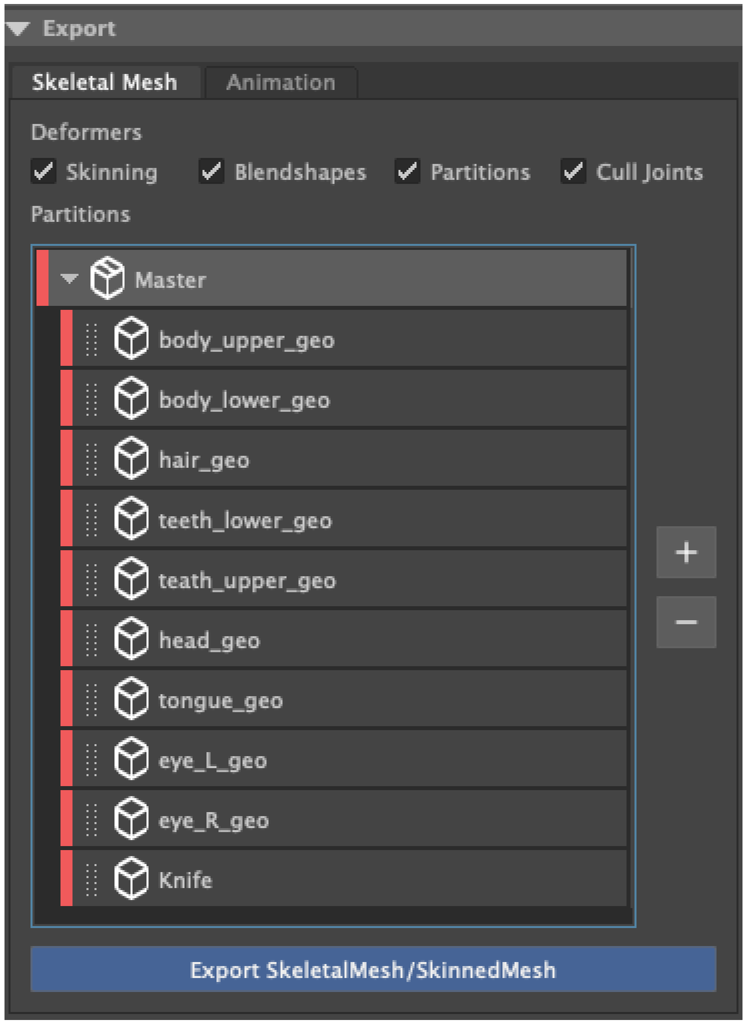


3) Click the folder icon in the Shifter UI.
4) The Package path to the directory will be retrieved from Unreal. You can modify it as you please, and the folder structure will be generated on import.

10.2.2 Export

Skeletal Mesh

Allows for the exporting of Skeletons and Geometry.



- **Skinning**: Export Skinning data
- **Blendshapes**: Export Blendshapes that exist on the geometry.
- **Partitions**: Export Partitioned FBXs.
- **Cull Joints**: With Cull Joints enabled, generated fbx partition files will have all unneccesary leaf nodes removed.

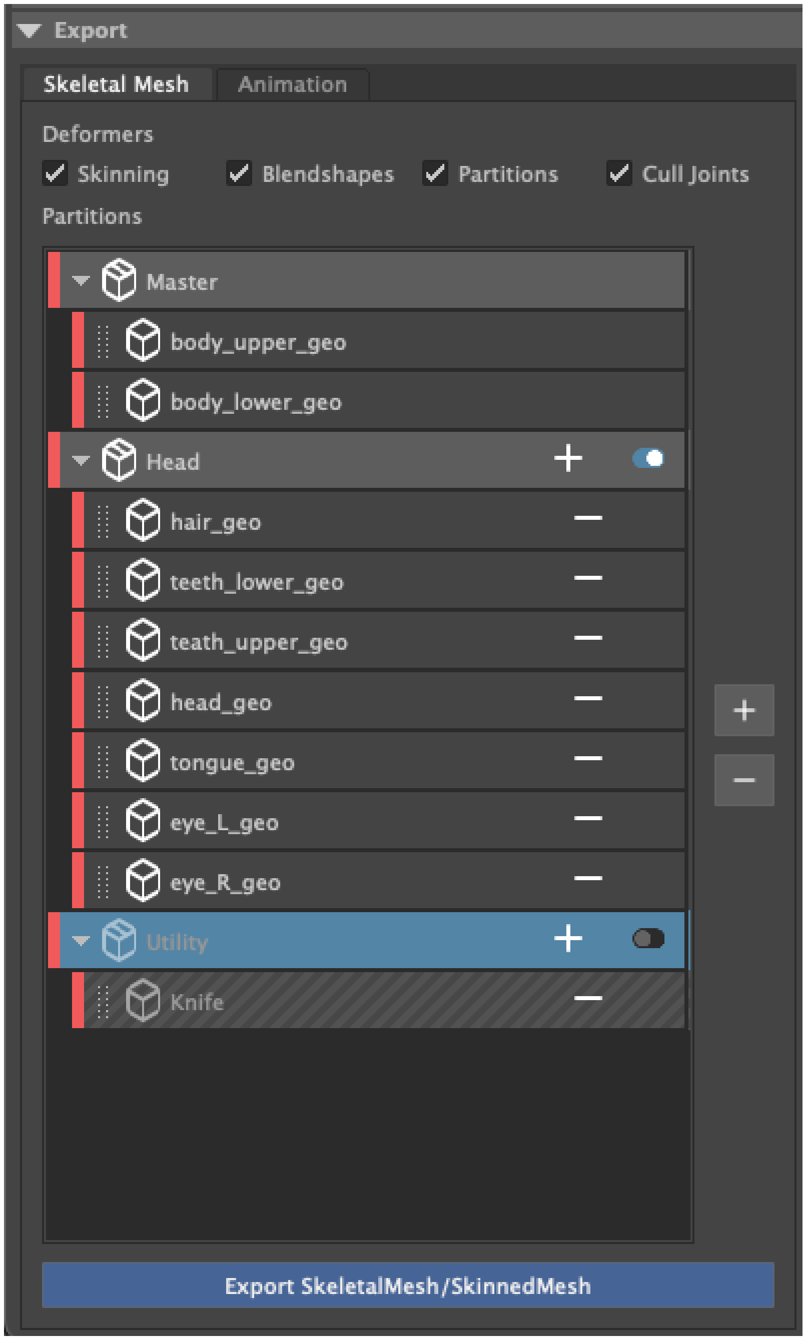
Unnessary leaf nodes, would be any joint that is no longer driving geometry and no longer required to drive any other joints that has skinning, to influence geometry that is part of the partition.

Note: Exporting the Skeletal Mesh, will trigger a mayabatch session, that performs all the file conditioning, and then performs the fbx exporting.

Partitions

Performs partitioning of geometric data. Partitions are designed to allow you to export once, and generate an FBX per a partition. Each generated FBX partition, will only contain the geometry that has been added to the partition.

- When you add the **geometry roots**, all geometry child objects will get added to the **Master** partition.



- Press the **"+"** button to create a custom partition. Once it has been created you can drag any other geometry objects from the master partition, to the custom partition.
- Right click on a Partition to change its colour, duplicate or delete it.
- Toggling the button on the partition, will disable it from being exported.

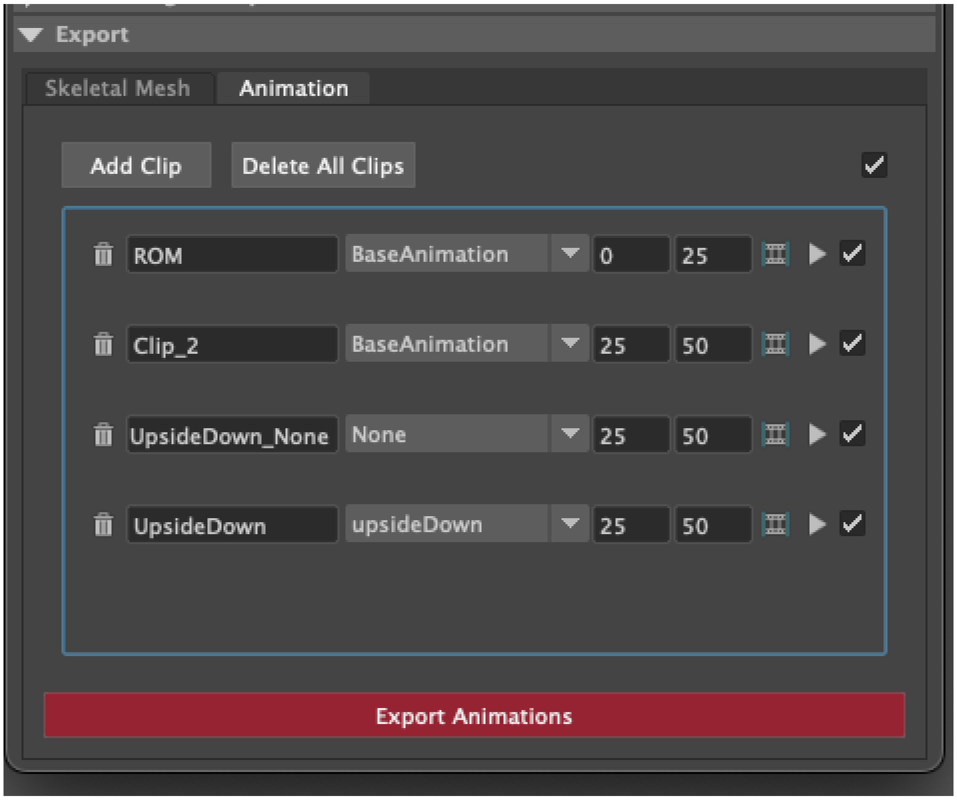
Export Skeletal/SkinnedMesh: Performs the FBX export, and if **"Enable Unreal Engine Import"** is active, the fbx's will be imported into the active Unreal Engine project.

Note: If you want to use a pre-existing skeleton in Unreal, make sure to have selected the *skeleton* in the *Unreal Engine*

Imports section. If you have not, a new Skeleton will be generated on import into Unreal.

Animation

Exports the Maya animation as an FBX. **Clips** allow for sections of the maya timeline to be exported, while also utilising the animation layers.

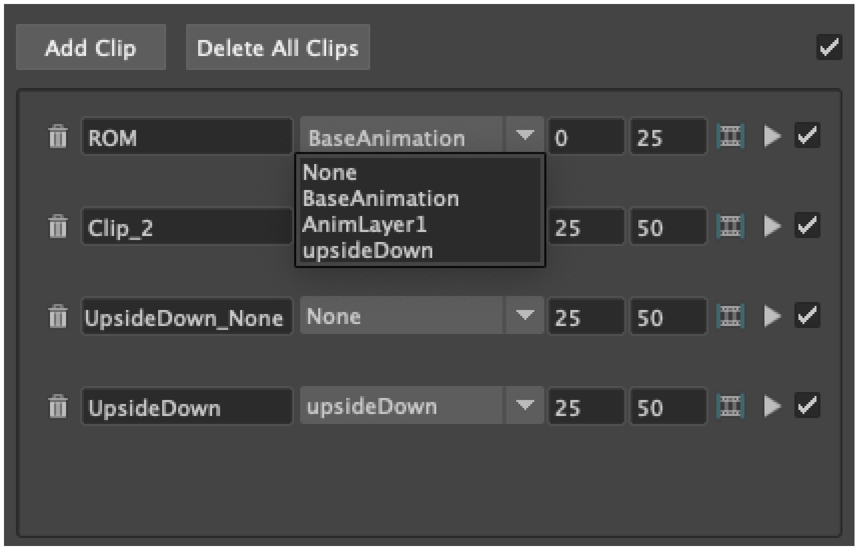


Clip

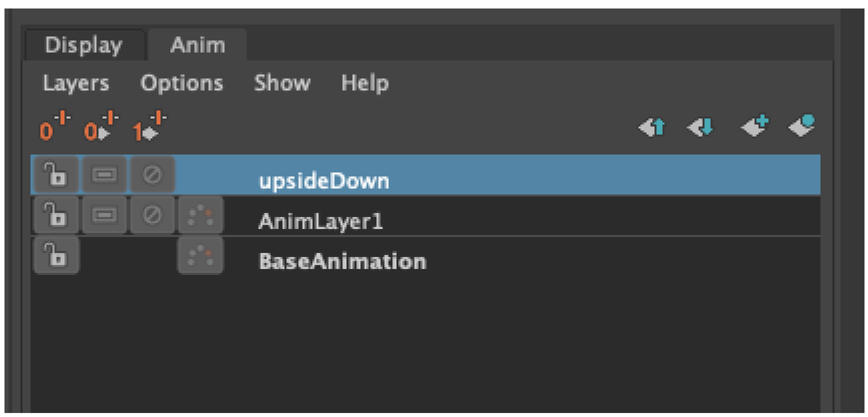
Clips allow you to create named animation exportd, that represent a section of time on the maya timeline. New Clips will automatically read the duration of the Maya timeline, and use that as the start and end frame.

- **Trash can**: Removes the clip.
- **Name of the clip** will be appended to the file. eg. *BoyA_ROM*, *BoyA_Clip_2*
- **Drop-down**: represents the animation layer that will have its animation read, and exported as the final FBX.
- **Start Frame**: The frame number that the animation will start at.
- **End Frame**: The frame number that the animation will end at.
- **Set timeline range**: Updated Mayas timeline to fit the range that is specified.
- **Play**: Plays back the clip on loop.
- **Tick Box**: Disabled the animation clip, stopping the clip from being exported.

Animation Layers



- **None**: Uses the current configuration of active and disabled Animation Layers. Exporting exactly what you see in scene.
- Any other selected animation layer, will export the **Animation Layer** and **Mayas BaseAnimation**



CHAPTER

ELEVEN

SHIFTER COMPONENT REFERENCE

Shifter comes with over 40 components you can build your rig from. This section covers the functionality and settings of each one in detail.

11.1 Guide Settings

11.2 Main Settings

Settings shared by all components.

- **Name**: The base name of the component, all the parts of the component get renmaed.
- **Side**: Which side the component is on; Left, Right or Center
- **Component Index**: WIP
- **Connector**: WIP

11.2.1 Joint Connection Settings

- **Use Joint Index**: WIP
- **Parent Joint Index**: WIP

11.2.2 Channel Host Settings

- **Host**: The guide that stores the extra attributes for this component, such as IK/FK blendering and more. This will generally be a control_01 component.

11.2.3 Custom Ctonrollers Group

WIP

11.3 Shifter Components

11.3.1 arm_2jnt_01

Two bone arm setup with:

- **IK/FK blending** for switching between animating with IK or FK
- **Roll** attribute that lets you adjust the elbow direction, without moving the pole vector
- **Twist joints** for better deformations around the shoulder and wrist joints
- **Armpit roll** to manually adjust the twist joints around the shoulder joint
- **Scale** of the entire arm
- **Stretching** so the arm can elongate when the IK control is too far away to reach, including a maximum stretch limit
- **Sliding** of the elbow, which elongates the upper arm while shortening the lower arm, or vice versa
- **Softness** to prevent popping when the IK arm is stretched out straight
- **Reverse** to make the arm bend in the opposite direction when using IK
- **Roundness** for rubber hose style animation
- **Volume** preservation during stretching
- **Elbow** controller for accurately positioning the elbow, for instance when a character is resting her elbows on a table
- **Space switchers** for the IK, Up Vector and Elbow controllers that controls which space they follow

This is the default arm comoponent used by the biped template rig.

Guide Positioning

WIP

Compoent Settings

TODO! Should this be tools tips in the interface itself.

- **IK/FK Blender**: Default

Source

Link to Module (how)

11.3.2 arm_2jnt_02

WIP

11.3.3 arm_2jnt_03

WIP

11.3.4 arm_2jnt_04

WIP

11.3.5 arm_2jnt_freeTangents_01

WIP

11.3.6 arm_ms_2jnt_01

WIP

11.3.7 cable_01

WIP

11.3.8 chain_01

WIP

11.3.9 chain_FK_spline_01

WIP

11.3.10 chain_FK_spline_02

WIP

11.3.11 chain_FK_spline_variable_IK_01

WIP

11.3.12 chain_IK_spline_variable_FK_01

WIP

11.3.13 chain_IK_spline_variable_FK_stack_01

WIP

11.3.14 chain_net_01

WIP

11.3.15 chain_spring_01

WIP

11.3.16 chain_stack_01

WIP

11.3.17 chain_whip_01

WIP

11.3.18 control_01

WIP

11.3.19 eye_01

WIP

11.3.20 foot_bk_01

WIP

11.3.21 hydraulic_01

WIP

11.3.22 leg_2jnt_01

WIP

11.3.23 leg_2jnt_02

WIP

11.3.24 leg_2jnt_freeTangents_01

WIP

11.3.25 leg_3jnt_01

WIP

11.3.26 leg_ms_2jnt_01

WIP

11.3.27 lite_chain_01

WIP

11.3.28 lite_chain_stack_01

WIP

11.3.29 meta_01

WIP

11.3.30 mouth_01

WIP

11.3.31 mouth_02

WIP

11.3.32 neck_ik_01

WIP

11.3.33 sdk_control_01

WIP

11.3.34 shoulder_01

WIP

11.3.35 shoulder_02

WIP

11.3.36 shoulder_ms_01

WIP

11.3.37 spine_FK_01

WIP

11.3.38 spine_S_shape_01

WIP

11.3.39 spine_ik_01

WIP

11.3.40 spine_ik_02

WIP

11.3.41 squash4Sides_01

WIP

11.3.42 squash_01

WIP

11.3.43 tangent_spline_01

WIP

11.3.44 ui_container_01

WIP

11.3.45 ui_slider_01

WIP

CHAPTER

TWELVE

SYNOPTIC USER DOCUMENTATION

WIP section: Please visit: mGear Youtube channel

- Basic operations
- Space Switcher
- Animation transfer IK/FK

CHAPTER

THIRTEEN

VIDEO TUTORIALS

WIP section: Please visit: mGear Youtube channel

CHAPTER

FOURTEEN

FAQ

14.1 What is mGear?

mGear is a rigging and animation framework for Autodesk Maya. mGear provides a set of convenient modules, tools and c++ solvers to streamline the development of rigging and animation tools.

14.2 Is mGear a modular rigging system?

Yes, mGear have modular rigging system called **Shifter**.

14.3 Is mGear providing animation tools?

Yes. Synoptic, Crank and softTweaks are animation tools. And we have more in the kitchen.

14.4 Will be always free and open source?

Yes!

14.5 Who can use mGear?

mGear is for everybody who needs to create a rigs or develop rigging tools. For example riggers and animators without programming knowledge can use it out of the box to generate infinite variety of rigs combinations. TDs with Python and C++ knowledge can also extend the functionality and adapt it to her/his needs/pipeline.

14.6 Why mGear's Shifter use custom solvers instead of Maya standard solvers?

The default components provides with **Shifter** are using custom solvers to simplify the rig construction encapsulating complex functionality and improve the playback performance. But you can create your own component without using the custom solvers.

14.7 Does mGear have the same functions/tools of Gear Softimage?

No, mGear doesn't have all the functions that Gear has. Some of the functions will be implemented in the future, some of them never will be implemented. It depends if the tool can be apply to Maya workflow and philosophy.

14.8 What I get out of the box?

- Shifter: Modular rigging system. (Exp: bipeds, quadrupeds, birds, creatures, robots, mechanical props, unlimited limbs, etc…)
- C++ Solvers: High performance rigs.
- Synoptic viewer: Animators interface and picker
- Rigbits: General purpose rigging tools.

14.9 Why should I add mGear to my pipeline?

- mGear is free and open source
- Don't have any license cost, so it is ideal for scalability.
- mGear is production proven tool since 2010 (Softimage Gear 1.0)
- Shifter modular rigging system is easy to learn and fun to use.
- Ready for game engine.

14.10 Who is currently developing mGear?

The mGear Development Team. Please check the website or github for more information.

CHAPTER

FIFTEEN

RELEASE LOG

15.1 4.2.1

New Features

- Added the new module chain_02

Enhancements

- arm_2jnt_free_tangents_01 Align wrist to world orientation option #373
- EPIC Arm and leg 02: added T reset pose option #359
- Human Ik Mapper - Batch bake #374
- Rig Builder: Added Pyblish validator and pre script to update guides #365
- Shifter: Unlock visibility on rig top node. New guide settings: Joint size and guide vis after build #367

15.2 4.2.0

New Features

- ueGear: 0.5 Beta
- Shifter Game Tools: FBX exporter #117
- Shifter: FBX exporter Intergration with Unreal #309
- RBF Manager 2.0: Miscellaneous Improvements #324
- Shifter: Rig Builder #115
- Mocap Tool: HumanIK mapper tool #348

Enhancements

- Animbits: Spring Manager misc bugs and updates #317 #349
- Shifter: Add load from selection option in the template explorer #313
- Core: Curve module update #319
- chain_FK_spline_02 and control_01: add support for leaf joints #332
- Shifter: Squash_01 add scale multipliers #33
- Shifter: Add Match guide to joint hierarchy command #350

Bug Fix

- Shifter: Not context menu with ghost controls #251
- Shifter Component bug fix: chain_IK_spline_variable_FK_stack_01 #326 # 325
- Made a metadata for rotateOrder #343 #328
- Shifter: Replace self.__class__ in all components to avoid recursion error #362

15.3 4.1.2

New Features

- Animbits: Spring Manager #266

Enhancements

- shifter: Right click menu add space swich range like synoptic #206

15.4 4.1.1

Enhancements

- EPIC_leg_3jnt_01 add support for 0 division on sections #273
- Add a dagmenu to reset all controllers on viewport menu #286

Bug Fix

- Fixed Error while mgear menu generation on startup #265 #267
- Rigbits: Fixed a bug related to the Mirror Control Shape tool. reported by remicc #252 #174
- Rigbits: Fixed a bug that can't open a fileDialog to import/export a SDK file throughout the GUI. #250
- Shifter: Control_01 and other simple components wrong naming with some custom name rules. #268
- Core: getTransformLookingAt fix axis calculation for -zx and -xy #296
- Channel wrangler move bool channels #217
- SoftTweak tool doesn't keep the right order of the softmod when re-import from .smt #262

15.5 4.1.0

New Features

- Animbits: Space Recorder
- chain_variable_IK #193
- EPIC components improvements: arm 2.0 + leg 2.0 + leaf joint in all components + Misc Improvements #195
- EPIC Meta_01 component #236
- EPIC neck and spine component v2 using splineIK solver #228
- EPIC_chain_IKFK_01 #192
- EPIC_layered_control_01 #226

- Misc: Smart export hotkey #180
- Rigbits: PROXY GEO #196
- Rigbits: Space Manager #152
- Rigbits: Tweaks support for proximity pin #230
- Shifter EPIC quadruped leg component #116
- Shifter: embed guide information in rig #248
- Shifter: Right click context menu for guides #187
- Solvers: Add spring node gravity and simple collision #94

Enhancements

- Added info for the user if (un)installation fails. #247
- Channel Master: New features #74
- Core: Added lineWidth of curves with collect_curve_data on curve.py #148 #151
- Core: attributes: add vector 3 attr method #156
- Core: deformer module + rigbits adding connect with morph #233
- Drag and drop support for more mGear's serialized formats #179
- EPIC Components adding support for custom name description #239
- Epic templates: Change IK reference hand and foot space to follow arm and legs #141
- Maya 2024 compatible.
- Metahuman template detach command + review leaf joints connection/disconnection #52
- Misc: Minimize code in userSetup.py #93
- Rigbits: Eye rigger 2 fixed number of joints #249
- Rigbits: IO Dialog use latest open folder
- Rigbits: Mirror Controls add extra attributes #200
- Rigbits: Misc improvements #129
- Rigbits: Move existing blendshape node to the front of chain #128
- Rigbits: RBF Manager, update SHAPES new node compatibility #244
- Rigbits: Tweaks optional control shape argument
- Shifter 3_jnt_leg Component: Tweak ctl by joint and MISC improvements #138
- Shifter Guide x ray curve in 2022 and new Maya versions #209
- Shifter: add rig_geo_grp set #137
- Shifter: addCtl add to controller set is now optional
- Shifter: better settings for CTL description #191
- Shifter: build from selection should try to autoselect the guide if nothing is selected #170
- Shifter: Build from selection without selecting guide #131
- Shifter: Collect data options update #157
- Shifter: Commands to manage joints connections and delete rig #169

- Shifter: ConnectRef method update #159
- Shifter: custom step UI misc improvements #241
- Shifter: Data collector: collect ctl shapes #132
- Shifter: Data Collector: Track joint solvers inputs #127
- Shifter: Extract controls should filter if is ctl #185
- Shifter: hide node inputs for controls #204
- Shifter: Improve IK/FK matching for legs + foot #92
- Shifter: Joint tagging to track guide locators relation #112
- Shifter: optinal controls orientation #163
- Shifter: Option to create joint_org directly on scene root #104
- Shifter: Resizeable log window #133
- SimpleRig: lock _npo #215
- Update dagmenu.py #216

Bug Fix

- Adding in deregister for springGravity node #153
- anim_utils uses dict.iteritems() and errors in Python 3 #203
- Animbits: SoftTweak support for Maya 2022+
- attribute.py returns None and fails, if all channels are hidden #175
- Build from guide template file incompatible with EPIC components #238
- Core utils: viewport_off decorator fails in certain enviroment #190
- Core: findComponentChildren3 will fail if there is no children #171
- cvwrap missing print brackets for python 3 #84
- drag_n_drop_install script bug #154
- Epic components: Intermediate transforms in joint structure #142
- EPIC leg 02 wrong IK orientation in R side when Z-up #255
- Epic Mannequin Template several problems and bugs #242
- EPIC_legs flip/twist issue and EPIC_arm tangent scale not 0.0 #99
- Export weight maps broken in 2022+
- Game Tools Export: Set index is incorrect, re-connect fails #231
- IKFK match offset in biped template #122
- leg_3jnt_01 module breaks when rotated to be Z-up #161
- Metahuman driver neck bones not driven by mGear EPIC Metahuman rig #232
- Metahunam template right hand fingers bad orientation #173
- mgear menu disappearing issue #254
- mgear viewport menu: Range Switch + missing space switch options #178
- RBF Manager: import errors when 'drivenControlName' is null #149

- RBFManager: check if drivenControlName is valid before testing scene #150
- RBFManager: fix mirroring and add manual entry feature #155
- RBFManager: Mirror ctl action not working #211
- Rigbits: Bake spring menu command not working #83
- Rigbits: Bake Spring nodes #177
- Rigbits: blendshape module issue with 2.7 *args unpacking #160
- Rigbits: RBF fix sorted() call #125
- Rigbits: RBF manager failing to update the UI #124
- Rigbits: SDK IO: Fixed tangents are not supported by setKeyframe #164
- Rigbits: SDK manager reload python3 error #245
- Shifter : connectRef handle negates scaled axis references
- Shifter naming issue #225
- Shifter naming rule issue: If the {index} is removed #221
- Shifter: control_01 is missing ctl role. #167
- Shifter: Delete rig keep joints fails if no joints #186
- Shifter: fix ik/fk transition upv_ctrl #229
- Shifter: Leaf joints not created if connect to existing joints active #183
- Shifter: Rebuild rig on existing joints crash if joints has guide_relatives already created #165
- Shifter: upvector space bad index issue affecting several components #198
- Synoptic tabs list missing in guide configuration #256

15.6 4.0.9

Enhancements

- Maya 2023 compatible. (OSX and Linux only mgear_solvers are available. WeightDriver and other C++ 3rd party plugins are not yet available)
- Rigbits: Facial Rigger 2.0 BETA (Not yet exposed in menu)
- Shifter Component: Expose Foot roll default value in the component settings
- Shifter: addParamAnim exact name argument
- Shifter: Build log options
- Shifter: Extract controls keep color
- Shifter: Shifter: Improve IK/FK matching for legs + foot
- Shifter_EPIC_components: Joint name descriptions exposes in settings new tab

Bug Fix

- Rigbits: Facial rigger had some issues with Py3
- Shifter: component: chain_IK_spline_variable_FK_01 TypeError
- Shifter: FK/FK Match on Metahuman Leg Broken

- Shifter_EPIC_components: Epic_arm mirrored mid_ctr problem
- Shifter_EPIC_components: EPIC_leg_01 (Right) is broken

15.7 4.0.7

Enhancements

- Rigbits: Channel master external data support and various improvements
- mGear_Core: New env var "MGEAR_PROJECT_NAME" to set the project name in mGear menu
- Shifter: Pebles: Skin transfer and more templates
- Shifter: Data collector option to store data on joint custom attr
- mGear_Core: anim_utils: IK/FK match with keyframe only key the blend value on uiHost

Bug Fix

- Shifter_components: 3jnt_leg: joint flip issue fixed
- Shifter_EPIC_componentsMetahuman template twist flip problem fixed
- Logo missing from installer
- Shifter_EPIC_componentsMetahuman template toes offset IK/FK
- Shifter: custom step path fix for OSX
- mGear_core: Python3 reloadModule error fix

15.8 4.0.3

New Features

- Project is back to mono repository on Github
- Python 3 Support and Maya 2022
- Shifter: Auto-snap for metahuman biped Template
- Shifter: connect to existing joint in the scene
- Shifter: Data collector for IO with other DCCs (Experimental Feature)
- Shifter: New components. Epic mannequin components, chain_ori_loc_01
- Shifter: New/Updated biped template
- Shifter: RGB color support for controls

Enhancements

Rigbits: Removed lagacy facial tools * Anim_picker: Edit picker shape using curves * mGear menu icons * Shifter Component: Meta_01 new option to define how joints are connected * Shifter: Added optional x-ray for controls on Maya 2022 * Shifter: Control_01 leaf joint option (Creates a joint without the ctl) * Shifter: Guides blade new shape and color. Also new attribute to change the size * Shifter: Metahuman and Mannequin templates updated and new naming on controls * Shifter: Naming rule have separated side labels for controls and joints * Shifter: Naming rule support for index padding * Shifter: Updated pole vector FK/IK match

Bug Fix

- General bug fixes in all modules, Python3 compatibility and Maya 2022. More info https://github.com/orgs/mgear-dev/projects/20

15.9 3.7.11

Enhancements

- mgear_dist: New drag and drop installer [mgear_dist#62]
- Shifter: Extending the CustomShifterStep base class functionality. [shifter#109]
- mGear_core: Added meshNavigation.edgeLoopBetweenVertices [mgear_core#77]
- mGear_core: Added create raycast node function in applyop.py [mgear_core#90]

Bug Fix

- Shifter: Error when joint name start with number [shifter#111]
- mGear_core: Bad IKRot rol reference anim_utils.py [shifter#82]
- mGear_core: Remove compile PyQt ui menu command for Maya 2022 compatibility [shifter#81]
- mGear_core: Knots saved in json file and read if they exist [shifter#76]
- Rigbits: Fix missing import in menu.py [rigbits#68]
- Rigbits: rbf manager, import error catch and cleanup [rigbits#73]
- Rigbits: Fix eyebrow joint orientation [rigbits#72]
- Shifter_EPIC_components: Improve joint placement precision on arm, leg and spine. [shifter_epic_components#20]
- Shifter_EPIC_components: Fixed relation dict value of "knee" in EPIC_leg_01 which causes building failure in certain cases. [shifter_epic_components#19]

15.10 3.7.8

New Features

- CFXbits: Xgen IGS boost: New tool to create curve based grooming with xgen interactive grooming splines [cfxbits#1]
- mGear solvers: New matrixConstraint node [mgear_solvers#5]
- mGear_core: Add support for drag n drop of mGear filetypes, .sgt [mgear_core#79]
- mGear_core: Deformer weight IO module [mgear_core#75]
- mgear_dist: Drag and Drop easy installer [mgear_dist#56]
- Shifter: Configurable naming template. [shifter#83]
- Shifter: Joint orientation options. [shifter#73]
- Shifter: Plebes (a tool for rigging character generator characters with mGear). [shifter#96]
- Shifter_EPIC_components: New set of componets specially design for Unreal engine and Games in general.

Enhancements

- mGear_core: General update to add CFXbits required functions [mgear_core#63]

- mGear_core: Skinning mismatch vertex warning should include the name of the object [mgear_core#63]
- Shifter: Add support for #_blade in chain coponents. [shifter#107]
- Shifter: Attributes naming using component short name(instance Name) not component type name. [shifter#95]
- Shifter: IO return shifter rig object for NXT tools integration. [shifter#94]
- simpleRig: Improve automatic hierarchy creation [simpleRig#8]

Bug Fix

- Anim Picker: Create picker improvements [anim_picker#21]
- Anim Picker: Duplicate behavior creates instances [anim_picker#24]
- Anim Picker: Duplicating pickers, spacing issue [anim_picker#22]
- Anim Picker: Fail gracefully when space switch controls are not found [anim_picker#33]
- Anim Picker: save overlay offset when change windows size [anim_picker#19]
- Anim Picker: UI buttons hidden in OSX [anim_picker#34]
- Animbits: Channel Master: Channel Master: Sync with Graph editor. [animbits#54]
- Animbits: Channel Master: sync selected channels in graph editor. [animbits#55]
- mGear solvers: added in the clamp values for the squash and stretch node [mgear_solvers#6]
- mGear_core: anim_utils: improve IK FK match pole vector calculation [mgear_core#65]
- mGear_core: Attribute module new functions: Make it work with control custom names [mgear_core#62]
- mGear_core: Mirro/flip pose not working with custom names [mgear_core#71]
- mGear_core: Mirror/flip pose fail [mgear_core#70]
- mGear_core: QApplication instance dont have widgetAt method on Maya 2020 [mgear_core#66]
- mGear_core: shifter_classic_components repeatedly added to sys.path [mgear_core#69]
- mGear_core: Stripe pipes from skinCluster names [mgear_core#64]
- mgear_dist: Incorrect grammar in UI [mgear_dist#26]
- mgear_dist: update menus to str command [mgear_dist#53]
- Rigbits: Add attr ctrl tweaks [rigbits#60]
- Rigbits: Add control and tweaks module controls need to create “isCtrl” control tag [rigbits#50]
- Rigbits: Facial rigger is compatible with Shifter’s game tools [rigbits#37]
- Rigbits: Mirror controls required target shape to exist [rigbits#56]
- Rigbits: RBF manager mirror with custom names [rigbits#63]
- Shifter: Game tools fix connection issue with new matrix constraint node. [shifter#108]
- Shifter: Game tools is not disconnecting all the connections between rig and model. [shifter#68]
- Shifter: Guide component scale inconsistency at creation time. [shifter#97]
- Shifter: replaces backslashes with forward slashes for Mac OS. [shifter#101]
- Shifter: Set by default Force uniform scaling to ON. [shifter#79]
- Shifter_classic_components: Change on Shifter leg_2jnt_tangent component settings UI [shifter_classic_components#81]

- Shifter_classic_components: Control_01 component space switching with mgear viewport menu [shifter_classic_components#82]
- Shifter_classic_components: Fix for issue “Menu: Ctrl+Shift results in broken shelf items” [shifter_classic_components#87]

WARNING

- mgear_dist: dropping support for Maya 2017 and older [mgear_dist#60]

15.11 3.6.0

New Features

- Shifter_classic_components: chain_spring_lite_stack_master_01: New component [shifter_classic_components#79]

Enhancements

- Anim Picker: Add create picker menu items based on selection [anim_picker#18]
- Anim Picker: Make select controls display more noticeable [anim_picker#16]
- Animbits: Channel Master: Add channels from any section in ChannelBox. [animbits#50]
- Animbits: Channel Master: Auto color options. [animbits#51]
- Animbits: Channel Master: option to configure channel order. [animbits#37]
- Animbits: Channel Master: Turn off real time update on scrubbing. [animbits#51]
- Animbits: Channel Master: Use selected channels for copy/paste keyframes. [animbits#52]
- Animbits: softTweak: add surface fallof option [animbits#53]
- mGear_core: attribute module new functions: get_selected_channels_full_path + collect_attrs [mgear_core#56]
- Shifter: Add Joint Names parameter for customizing joint names in guide settings. [shifter#85]
- Shifter_classic_components: lite_chain_stack_02 component: add blend option to turn off the connection [shifter_classic_components#78]

Bug Fix

- Animbits: Channel Master: Blendshape node channels bug. [animbits#49]
- Shifter: Importing old guides with missing parameters error. [shifter#69]

15.12 3.5.1

Bug Fix

- mGear_core: When copy skin, match the skinningMethod as well [mgear_core#55]
- Rigbits: RBF Manager mirror bug with Flex Add_attribute [rigbits#54]

15.13 3.5.0

New Features

- Animbits: Channel Master [animbits#14]
- Shifter: Auto Fit Guide (Beta preview). [shifter#82]

Enhancements

- Anim Picker: Make select controls display more noticeable [anim_picker#16]

Bug Fix

- Anim Picker: CentOS and windows Maya 2019/2020 TypeErrorr [anim_picker#15]
- mGear_core: dagmenu error when parent switch with keys on and rig with namespace [mgear_core#53]
- mGear_core: Fix loop crash when quering tag childrens [mgear_core#52]
- mGear_core: Fixed path handling in exportSkinPack if it is called with arguments. [mgear_core#37]
- mGear_core: getRootNode doesn't find the root correctly [mgear_core#51]
- mGear_core: Mirror function causes tag attributes to mirror their content [mgear_core#47]
- mGear_core: Parent switch dag menu not working when root node is parented under a non referenced heararchy. [mgear_core#48]

15.14 3.4.0

New Features

- Anim Picker: New Animation Picker [anim_picker#2]
- mGear_core: mGear viewport menu [mgear_core#38]
- Rigbits: SDK Manager [rigbits#42]
- Shifter_classic_components: SDK manager special component [shifter_classic_components#75]

15.15 3.3.1

Bug Fix

- Rigbits: Facial rigger tools QT aligment argument [rigbits#44]

15.16 3.3.0

New Features

- Shifter_classic_components: Cable component [shifter_classic_components#73]
- Shifter_classic_components: UI_slider and UI_container component [shifter_classic_components#66]
- Rigbits: New eyebrow Rigger [rigbits#40]

Enhancements

- Shifter_classic_components: Control_01: Expose more space switch options [shifter_classic_components#7]

15.17 3.2.1

Enhancements

- Shifter_classic_components: arm_2jnt_04: wrist align and plane normal [shifter_classic_components#58] [shifter_classic_components#59]
- Shifter_classic_components: S_Spine change the relative connections [shifter_classic_components#67]
- mGear_core: Added 2D guide root for Shifter components [mgear_core#36]
- Shifter: Build log window clears instead of reopening. [shifter#74]

Bug Fix

- Shifter: Fixed a guide renaming issue. [shifter#71]
- Shifter: Renamed Connexion to Connection in some places.. [shifter#75]
- Shifter: Renaming components will fail if the names are not unique. [shifter#70]
- Shifter_classic_components: foot_bk_01 component roll_ctrl issue [shifter_classic_components#68]
- Shifter_classic_components: Visual axis reference for control_01 and arm_2jnt_04 is not scaling correctly [shifter_classic_components#57]
- Shifter_classic_components: Fixes building of chain_01 when set to IK only [shifter_classic_components#65]
- Shifter_classic_components: spine_S_shape rename bug [shifter_classic_components#50]
- mGear_core: dag.findComponentChildren2 fails after a rig was built. [mgear_core#32] [mgear_core#35]
- mGear_core: QDragListView ignores drop event on self [mgear_core#34][mgear_core#33]

15.18 3.2.0

New Features

- Animbits: Animation GPU cache manager [animbits#11]
- Rigbits: New Facial Rigger [rigbits#28][rigbits#27][rigbits#64][rigbits#33][rigbits#32]
- Shifter_classic_components: new arm and leg with elbow and knee thickness control [shifter_classic_components#55]
- Shifter_classic_components: New component arm_2jnt_03 with align wrist with guide option [shifter_classic_components#53]
- Shifter_classic_components: New component mouth_02 [shifter_classic_components#51]

Enhancements

- Rigbits: Mirror Controls Shape Tool [rigbits#25]
- Rigbits: RBF manager updated with support for non-control objects [rigbits#31]
- Shifter_classic_components: control_01, arm_2jnt_04 add orientation visual feedback [shifter_classic_components#54]

15.19 3.1.1

New Features

- shifter_classic_components: New Component: chain_IK_with_variable FK and stack connection [shifter_classic_components#43]
- shifter_classic_components: New Component: chain_net_01 [shifter_classic_components#42]
- shifter_classic_components: new component: Lite chain stack [shifter_classic_components#40]

Enhancements

- mgear_core:implemented filesize compression for jSkin and gSkin (pull request #28)
- Rigbits: Update tweakers modules [rigbits#18]
- Shifter: add optional uihost argument on addAnimParam and addAnimEnumParam [shifter#60]
- Shifter: avoid negative scaling in joints [shifter#59]
- Shifter: inspect settings open tap option [shifter#62]
- Shifter: Shared custom step fix color feedback and hover information [shifter#57]
- shifter_classic_components: chain_net_01: improve pickwalk [shifter_classic_components#47]
- shifter_classic_components: Chains with stack connection should have connection offset options [shifter_classic_components#46]
- shifter_classic_components: Review channel hosts for stack connection chains [shifter_classic_components#44]
- simpleRig: handle geometry selection option when convert to shifter rig [simpleRig#6]
- Synoptic: Fix refresh needed on togglButtons and on visibility/control tabs [synoptic#13]

Bug Fix

- mgear_core: attribute module log error wrong flags [mgear_core#29]
- shifter_classic_components: chain FK with variable IK the extreme controls should not be on 0 or 1.0 of the path [shifter_classic_components#45]

15.20 3.0.5

Bug Fix

- mGear_core: Attribute: moveChannel doesn't support float attr [mgear_core#27]
- mGear_core: Callback manager: UserTimeChangedManager change condition state to playingBackAuto [mgear_core#28]
- Rigbits: Eye rigger and Lips Rigger bad naming in rig curves [rigbits#21]
- Shifter: Export guide to template (.sgt) will break component parent references if name is not unique [shifter#58]

15.21 3.0.4

Bug Fix

- Synoptic: Fix refresh needed on togglButtons and on visibility/control tabs [synoptic#13]
- mGear_core: Node: controller_tag_connect fail if ctl parent doesn't have tag [mgear_core#24]
- Shifter_classic_components: Eye component update structure [shifter_classic_components#39]
- Shifter_classic_components: Spine FK: fisrt joint moving with IK chest control [shifter_classic_components#38]
- Shifter: custom step template still have old name import [shifter#56]
- Rigbits: hotkey creation command has bad imports [rigbits#19]
- Shifter: serialized guide with none parent components issue [shifter#55]
- Rigbits: Ghost control creator and Tweaks should handle ctrl Tag and custom pickwalk [rigbits#20]

15.22 3.0.3

New Features

- Flex: Flex is the mGear models (geometry) update tool inside rigs.
- Shifter: Build Rig from file [shifter#20]
- Shifter: Game Tools, for decouple deform and control rig [shifter#6]
- Shifter: Guide Relative placement [shifter#14]
- Shifter: Guide serialization to json
- Shifter: New Guide manager
- Shifter: Serialized Diff Tool
- Shifter: Serialized Guide Explorer
- Shifter_classic_components: New Component: Chain FK spline with variable IK controls [shifter_classic_components#26]
- Shifter_classic_components: New Component: Chain IK spline with variable FK controls [shifter_classic_components#30]
- Shifter_classic_components: New Component: Chain Stack [shifter_classic_components#32]
- Shifter_classic_components: New Component: shoulder_02 [shifter_classic_components#25]
- Shifter_classic_components: New Component: Spine FK [shifter_classic_components#31]
- Shifter_classic_components: New Component: Tangent_spline_01 [shifter_classic_components#28]
- Shifter_classic_components: New Component: Whip chain [shifter_classic_components#27]

Enhancements

- Animbits: softTweak: make UI dockable [animbits#8]
- Crank: Make UI dockable [crank#3]
- Crank: Shot Sculpting tool, General update initial Goals [crank#1]

- mGear_core: attribute: FCurveParamDef should store the samples from getFCurveValues [mgear_core#12]
- mGear_core: attribute: ParamDef: Dict serialisation [mgear_core#11]
- mGear_core: pyQt: showDialog option to make windows dockable [mgear_core#6]
- mGear_core: Skin module: Review it and update use Json and pickle [mgear_core#20] [mgear_core#23]
- Shifter: Custom step list. Visual cue for shared custom step [shifter#51]
- Shifter: FCurveParamDef should store the samples from getFCurveValues in value of paramDef [shifter#26]
- Shifter: update menu with new functionalities [shifter#37]
- Shifter: Update modal position menu to QT modern version [shifter#46]
- Shifter_classic_components: add new upv roll control to arm_2jnt [shifter_classic_components#36]
- Shifter_classic_components: Add UniScale option for games compatible [shifter_classic_components#9]
- Shifter_classic_components: arm_2jnt_01 and leg_2jnt_01: Make optional the extra support joint in the articulations [shifter_classic_components#3]

API Changes

- mgear_dist: Modularisation of mgear [mgear_dist#11]

Bug Fix

- mGear_core: Attribute: channelWrangler apply config from script fails due to attributeError [mgear_core#21]
- mGear_core: curve: create_curve_from_data_by_name should not take the name from the first shape [mgear_core#17]
- mGear_core: curve: importing curve while rebuild hierarchy will fail if the parent object don't have unique name [mgear_core#18]
- Rigbits: Duplicate symmetry bad import string [Rigbits#13]
- Rigbits: Replace Shape Command doesn't handle if the target object have input connections in the shape [Rigbits#12]
- Shifter: Component connector: standard fallback [shifter#27]
- Shifter: Component space references: add checker for space references names [shifter#16]
- SimpleRig: re-import configuration dont link unselectable geometry [simpleRig#1]

15.23 2.6.1

New Features

- Animbits: Crank shot sculpt [mgear#233]
- Rigbits: RBF Manager: support for non-control objects [mgear#228]

15.24 2.5.24

New Features

- mGear: IO curves [mgear#76]
- Rigbits: RBF Manager [mgear#183]
- Rigbits: set driven key module [mgear#160]
- Simple Rig: 2.0 [mgear#163]
- Synoptic: Control lister Tab [mgear#99]
- Synoptic: geometry visibility manager Tab [mgear#130]
- Synoptic: Spine IK <–> FK animation transfer [mgear#169]

Enhancements

- Animbits: SoftTweak tool update [mgear#167]
- mGear: skin: copy skin [mgear#168]
- Shifter: chain_FK_spline_01: keep length multiplayer channel [mgear#199]
- Shifter: chain_FK_spline_02: add extra Tweak option [mgear#202]
- Shifter: component ctrlGrp should be inherit from parent component [mgear#181]
- Shifter: Component Lite chain and chain FK spline mirror auto pose configuration if override negate axis direction in R [mgear#198]
- Shifter: Component Lite chain and chain FK spline mirror auto pose configuration if override negate axis direction in R [mgear#198]
- Shifter: Control_01: lock sizeRef axis [mgear#156]
- Shifter: Custom Step List: Highlight Background quicksearch [mgear#203]
- Shifter: Lock joint channels if "separated joint structure" is unchek [mgear#182]
- Shifter: Make not keyable the joints channel if jnt_org is checked [mgear#188]
- Shifter: neck_ik: add option to orient IK to world space [mgear#159]
- Shifter: Partial build skip custom steps [mgear#154]
- Shifter: spine_S_Shape: add option to orient IK to world space [mgear#164]
- Shifter: Turn on/off custom steps [mgear#189]

Bug Fix

- mGear: curve.addCnsCurve: modify the center list in some situations [mgear#172]
- Rigbits: Blended Gimmick joints bad naming with multy selection [mgear#153]
- Shifter: 3jnt leg roundness att for knee and ankle [mgear#144]
- Shifter: add_controller_tag. Fail on Maya old versions [mgear#187]
- Shifter: Component: spine_IK_02: Last FK control don't have correct attr [mgear#161]
- Shifter: Controller tag lost if export selection the rig [mgear#175]
- Shifter: Joint connection: Maya evaluation Bug [mgear#210]
- Shifter: leg_2jnt and leg _2jnt_freetangents not taking max stretch default setting [mgear#162]

- Shifter: Spine S Shape: bad build with offset on fk controls [mgear#146]
- Simple Rig: BBox computation fails with lights [mgear#212]
- Synoptic: IK/FK transfer doesn't save keyframes on blend channel [mgear#180]
- Synoptic: IK<->FK transfer strange refresh [mgear#173]

Known Issues

- Shifter: Undo Build from selection crash maya. Now flush Undo to avoid possible crash [mgear#74]

15.25 2.4.2

Bug Fix

- Animbits: SoftTweak root lost relative position to parent [mgear#143]

15.26 2.4.1

Bug Fix

- Shifter: Rotation inverted on joints with negative scale [mgear#142]

15.27 2.4.0

New Features

- Animbits: SoftTweaks tool [mgear#133]
- LINUX: Maya 2018 solvers
- Rigbits: Eye rigger tool [mgear#127]
- Rigbits: Lips Rigger tool [mgear#128]
- Shifter: New Component: Chain FK spline Component [mgear#104]
- Shifter: New Component: Lite FK chain [mgear#115]
- Shifter: New Component: Spine_S_shape [mgear#96]

Enhancements

- Shifter: Add alias names for space references [mgear#110]
- Shifter: Add visual crv connection for the upVector controls [mgear#124]
- Shifter: arm and leg 2jnt: add optional controls x Joint [mgear#114]
- Shifter: chain_FK_spline: add option to control visibility of controls [mgear#136]
- Shifter: Hide controls on Playback rig setting [mgear#131]
- Shifter: Improve parallel evaluation [mgear#123]
- Shifter: Lite_chain and Chain_FK_spline. Option to override side negation [mgear#139]
- Shifter: Neck_ik_01: add option to have only IK space reference [mgear#132]

- Shifter: Review rollspline solver precision values [mgear#138]
- Shifter: Set all controls shape to d1 curves [mgear#118]
- Shifter: Set to False the default use of uniscale in joints [mgear#117]
- Shifter: Update component with Proxy attributes [mgear#111]

Bug Fix

- Shifter: Bindpose bug with custom controllers grp [mgear#134]
- Shifter: Component addJnt error if negative scaling [mgear#141]
- Shifter: Extracted controls doesn't clean shape name [mgear#135]
- Shifter: leg_2jnt_01 maxStretch setting is lost at build time [mgear#140]
- Shifter: Maya 2018.2 flip in leg_2jnt_01 component [mgear#125]

15.28 2.3.0

Enhancements

- mGear: Attribute: addAttribute not setting default attribute value. [mgear#84]
- mGear: Attribute: update with lock and unlock attribute functions [mgear#83]
- mGear: PEP8 Style Refactor [mgear#100]
- mGear: Refactor all exception handling [mgear#88]
- mGear: Vendoring QT [mgear#89]
- Shifter: Build command review and log popup window [mgear#73]
- Shifter: Change Global_C0_ctl to World_ctl [mgear#66]
- Shifter: Control_01: Add option to have mirror behaviour [mgear#68]
- Shifter: Improve rig build speed [mgear#65]
- Shifter: Leg_2jnts_freeTangents_01:no ikFoot in upvref attribute [mgear#62]
- Shifter: Reload components in custom path [mgear#78]
- Shifter: Update guide structure in pre custom step [mgear#101]
- Simple Rig: Update functionality revision [mgear#71]
- Synoptic: spring bake util [mgear#61]

Bug Fix

- Rigbits: createCTL function issue [mgear#59]
- Rigbits: export skin pack error with crvs [mgear#56]
- Rigbits: skin: There is a case in exportSkin function breaks the existing file [mgear#58]
- Shifter: 3 joint leg: soft Ik range min in graph editor [mgear#82]
- Shifter: arm_2jnt_freeTangents_01 no attribute 'rollRef' [mgear#63]
- Shifter: Arms auto upvector and shoulder space jump [mgear#85]
- Shifter: Chain_spring_01: pop if manipulate FK ctl after Bake [mgear#75]

- Shifter: Connect Ctl_vis [mgear#103]
- Shifter: Control_01: rotation axis is missing Y lock [mgear#74]
- Shifter: Japanese Ascii [mgear#79]
- Shifter: Spring chain: lock control parent and bake spring bug [mgear#67]
- Shifter: Synoptic: IK/FK Match with arm_ms_2jnt_01 [mgear#80]

Known Issues

- Shifter: Undo Build from selection crash maya [mgear#74]

15.29 2.2.4

Enhancements

- Shifter: Global scale and size of controllers. [mgear#50]

15.30 2.2.3

Enhancements

- Shifter: Custom Steps: Added Stop Build and Try again option if step fail.[mgear#43]

Bug Fix

- Synoptic: Match IK/FK with split ctl for trans and rot [mgear#54]

15.31 2.2.2

Enhancements

- Shifter: Components: Legs: Mirror axis behavior on upv and mid ctl [mgear#47]
- Shifter: Componets: Arms: IK ctl mirror behaviour [mgear#48]
- Shifter: arm roll new reference connector [mgear#53]

Bug Fix

- Shifter: component UI min division hang. Check all components [mgear#42]
- Shifter: quadruped rig not being created in 2018 [mgear#44]
- Shifter: Close settings Exception on Maya 2018: Note: This is a workaround. The issue comes from Maya 2018 [mgear#49]

15.32 2.2.1

Bug Fix

- Shifter: Component: Hydraulic: Fix bad reference connector
- Docs: Text error fix
- Shifter: Text error fix

15.33 2.2.0

New Features

- Maya 2018 compatible
- Simple autorig This a new rigging sytem for basic props.
- Channel Wrangler: Channel manager with export import options.

Enhancements

- Synoptic: key/select all for custom widgets
- Skin IO: IO skin for curves & nurbs
- Skin IO: Now can export with Skin Packs. Every object will be in a separated file.
- Shifter: custom Sets: Now is possible to add custom sets to shifter components
- Shifter: Now all the controls are Tag as a control (> Maya 2016.5)
- Shifter: Custom Rig controls navigation
- Shifter: Custom steps IO to JSON file.
- Shifter: Componente: Chain_01: Non uniform scaling for FK controls
- Shifter: Now the controls have unchecked historical interest from ctl shapes for cleaner channel box
- Rigbits: Now replace shape support multiple shapes
- mGear: Menu updated with about info and useful links
- mGear: Added support for RGB color on icons/Controls

Bug Fix

- Shifter: component: freetangent arm and leg: Fixed joint offset in the extremes
- General: Fixed bad parenting for PySide dialogs.

15.34 2.1.1

New Features

- mGear solvers: New vertex position node. This node gets the vertex position in worldspace.
- Rigbits: New rigging commont library with toos and functions to help the rigging system. This library is meant to be use with custom steps or other rigging tools.
- Shifter: Components: New Components from Miles Cheng "arm_ms_2jnt_01", "shoulder_ms_2jnt_01" and "leg_ms_2jnt_01"
- Shifter: Components: New enviroment variable: MGEAR_SHIFTER_COMPONENT_PATH (only project components)
- Shifter: Custom Step: New enviroment variable: MGEAR_SHIFTER_CUSTOMSTEP_PATH to stablish relative paths for the custom steps data.
- Shifter: New Channel naming options

Improvements

- Improved error logging for custom steps and Synoptic.
- Shifter: Clean up jnt_org empty groups after rig build.
- Shifter: Components: Updated neck with optiona tangent controls.
- Shifter: Components: Arm have a new option to separate the IK controls in rotation and translation control
- Shifter: Components: Control extraction name buffer to avoid name clashing for ctl extraction on guides
- Shifter: Components: Pin elbow/knee
- Shifter: Components: Spine updated: Autobend optional control and optional mid tangent control
- Shifter: Components: Arms mid ctl and upv with optinal mirror behaviour.
- Shifter: Custom step using class implementation
- Shifter: Track information (rig Asset, components used version and mGear version)
- Synoptic: General visual and structure improvement. Big Thanks to Yamahigashi-san.
- Synoptic: IK/FK animation transfer
- Shifter: Updated biped guide
- Shifter: Updated Quadruped guide

Bug Fix

- Bad layout on setting windows with HDPI displays.
- Shifter: Components: General clean up and bug fixing (Please check github commint for more info).
- Issue mgear#9 leg_3jnt: Flip offset rz double connection
- Issue mgear#13 Chain_01 IK refs not being connected

15.35 2.0

New Features

- Custom enviroment variables for synoptic: MGEAR_SYNOPTIC_PATH
- cvWrap deformer included.
- Gimmick joints basic tools
- Mocap HumanIK mapping tool for standard Shifter biped
- New Component settings view.
- New Documentation
- New licensing under MIT license terms.
- Pre and Post custom Steps.
- Shifter: Modular rigging sytem rebranded.
- Shifter: Quadrupeds template and new leg component for 3 bones legs.
- Shifter: Single Hierarchy Joint connexion
- Shifter: Update Guides Command.
- Synoptic view Updated.

Inprovements

- Component guides will snap to parent position at creation time.
- Duplicate symmetry can find partial chain names. Is not needed to duplicate from the top root of the branch.
- Groups and dag pose connected to rig base node. This will avoid lost elements if we export selection.
- Guide Blades have new attr to control the roll offset
- mGear version and other useful information in guide root.
- Newly created guide components automatic update of the side and uiHost from the parent attributes.
- Shifter componets full review and functions unified.

CHAPTER

SIXTEEN

LICENSE

MGEAR is under the terms of the MIT License

Copyright (c) 2011-2018 Jeremie Passerin, Miquel Campos - 2018 The mGear-Dev Organization

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

A

F

G

H

N

O

P

Q

R

T

U

V

W