Face Rig Builder

The Face Rig Builder creates face rigs from a face model asset as well as the following elements:

* Skeleton Guide.
* GUI Guide.
* Blend Shapes.
* Driven Keys.
* Preferences.
* Weighting.

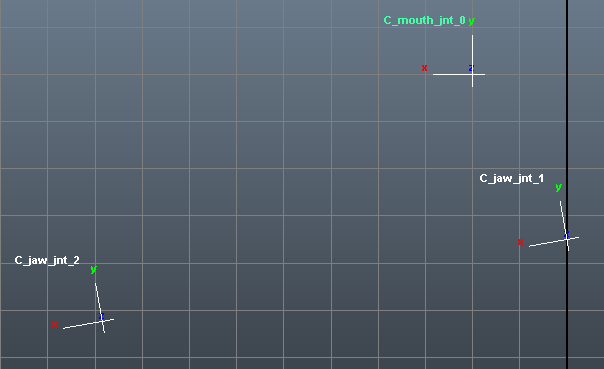
The following is a rundown of the workflows that contribute to building a face rig.

# Skeleton Guide

The skeleton guide is a collection of locators and curves that the face rig builder looks for in order to build a skeleton. You can find an example of the skeleton guide within the rigBuilder software package’s resource folder (../rigBuilder/resource/face/templates/blinky/rig/face/skeletonGuide/blinky\_face\_rig-\_skeletonGuide\_v001.ma).

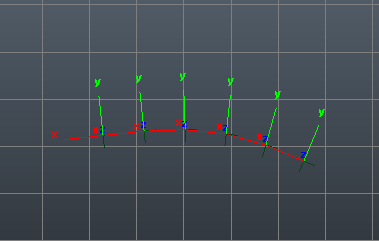
The code that builds the skeleton (rigBuilder.face.utils.skeleton.FaceSkeletonBuilder) looks for guides by name, and creates joints according to guide locations. The guides in the example scene create the skeleton required by the eye, mouth, and tongue modules.

The mouth module requires ***one*** mouth joint (‘C\_mouth\_gui\_0’), as well as ***two*** jaw joints (‘C\_jaw\_gui\_1’, ‘C\_jaw\_gui\_1’) to build the skeleton for the mouth module. The jaw joints will be used to create skeleton for upper and lower jaw, and the mouth joint a parent above both.



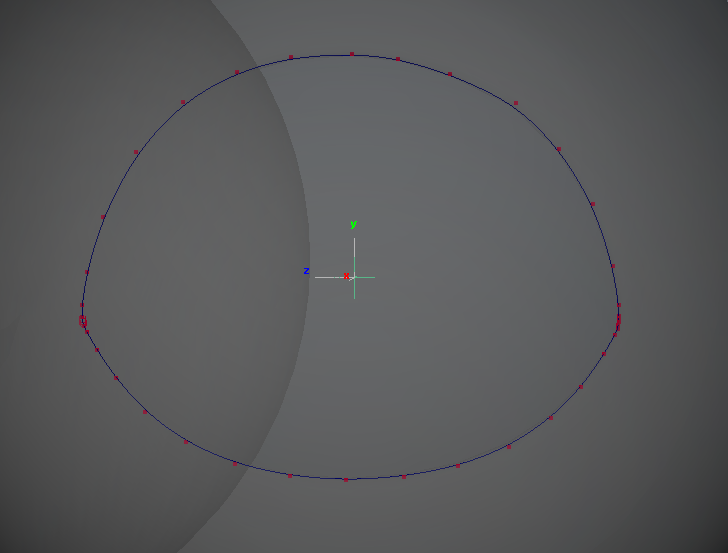
*A side view of an example mouth/jaw guide setup.*

The tongue module requires **at least one** tongue guide, but something like **six** would be preferable. These should be positioned through the tongue geometry at regular intervals, with the X-axis pointing toward the next guide in the chain (as a joint chain would do) and the Y-axis pointing upward.



*A side view of an example tongue guide setup.*

The eye module requires a locator (\*\_eye\_gui\_0) to define the center of the eye (oriented withthe X-axis pointing toward the pupil), as well as two curves (\*\_eyeLidUpper\_gui\_0, \*\_eyeLidLower\_gui\_0) to define the eye lids. Note that the curves should be duplications of the polygon edges running around the eye (using Maya’s *Modify > Convert > Polygon Edges to Curve* tool). ***Do not*** rebuild the curves, only reverse their direction if necessary to ensure that each curve origin is in the inner eye or tear duct area.

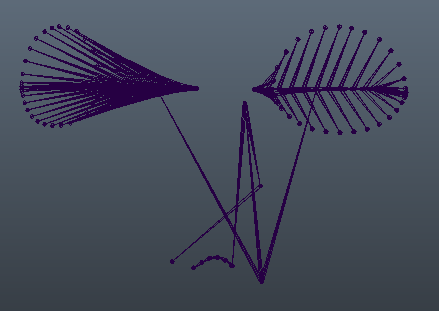


*An example of eye guide setup of the left eye.*

You also need a root joint (‘C\_head\_gui\_0’) that will serve at the uppermost parent of the face skeleton. Ideally this will be matched to the head joint in the body rig, to allow for seamless combination.

Testing the Skeleton Build

You can test the build of a skeleton in the Face Rig Builder GUI in the ***Skeleton Guide*** tab by using the Test Skeleton Build tool.



*An example of a built skeleton.*

Adding Joints

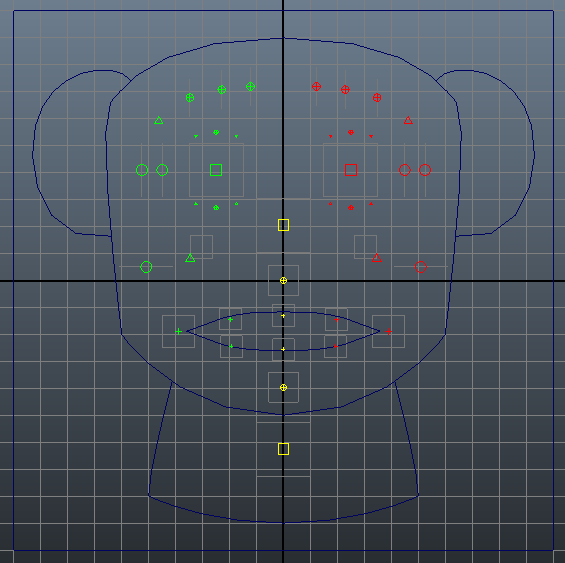
You can add more joints to the face rig skeleton build, by modifying the *FaceSkeletonBuilder* mentioned earlier, to look for relevant guide names and create joints accordingly. I would recommend looking at the mouth section of the skeleton builder as an example of how to add more joints to your face skeleton.

Publishing Skeleton Guides

You can only publish a Skeleton Guide from a clean scene, so don’t try and publish from a built face rig.

# GUI Guide

The GUI guide is a method for constructing interactive Face Control GUI’s for facial animation. They can be opened and edited from the GUI Guide tab of the Face Rig Builder GUI.



*An example of a GUI Guide.*

GUI Templates

You can open a GUI template in the dropdown menu provided. The Generic GUI template is simply a frame, but there is a hierarchy in the scene that the Face Rig Builder relies upon to build the controls. A Koala template has also been supplied to give a rough idea of how a finished GUI Guide might look.

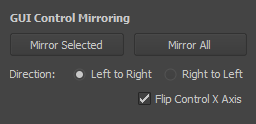
Curves can be drawn and parented under the *|faceGUI\_grp|C\_faceGUI\_ctl\_0|faceGUILayout\_grp* transform to give iconic reference for things like eyes, ears, mouth etc. This will be set to reference display type at build time to make them un-selectable.

Adding Controls

Using the Face Rig Builder GUI, controls can easily be added to new or existing templates, and positioned, oriented, and scaled according to animator preference. By selecting a control type and shape, and entering a position prefix, a description, and an index, you can simply add a new control to the GUI Guide. When checking the *Add Mirrored Control* check box, the opposite side control will also be created for left and right hand side controls.

Mirroring Controls

Convenience functions have also been added to make it easy to position controls for one side of the face and mirror to the opposite sides controls.



*The GUI Control Mirroring Options.*

***Mirror selected***works on two selected controls; a source and a destination (note that the direction radio box has no effect on this particular function).

***Mirror all*** will look at all controls (by name), and mirror based upon the direction provided in the radio box.

***Flip Control X Axis*** works for both mirror functions, and will allow for mirrored behavior on the X-axis of controls.

Adding Custom Templates

If you want to add or update a GUI template, you can do so by saving a maya ascii file into the resources folder of the rigBuilder project (../rigBuilder/resource/face/controlGUI/templates). I would recommend opening an existing template to use as a starting off point. The filename without extension will then be displayed in the GUI templates dropdown (you will need to refresh the GUI).

Adding More Control Types

If you want to add a new control type you can simply save a Maya ASCII file into the resources folder of the rigBuilder project (../rigBuilder/resource/face/controlGUI/controls). I recommend starting with the generic control, and adjusting the existing values to your desired specifications. The filename without extension will then be displayed in the Control Types dropdown (you will need to refresh the GUI).

Adding More Control Shapes

If you want to add more control shapes, you can easily do so by opening the Maya ASCII file found in the resources folder of the rigBuilder project (../rigBuilder/resource/face/controlGUI/controlShapes.ma). By adding your own curve under the *|controlShapes\_grp* node, you can add as many custom curve shapes as you like. Note that the name of your curve will be the name that appears in the shape type option box, and the orientation of the curve will impact the way it behaves in the GUI. I would recommend drawing all curves in the front view.

Publishing GUI Guides

You can only publish a GUI Guide from a clean scene. Do not try and publish from a built face rig.

# Blend Shapes

This step will not be heavily documented as it is really outside of the scope of the rigging system itself, only to say that the rig obviously supports blend shapes, and does not really care how you make them, however they must conform to some specific conventions.

Creating the Standard Shape Layout

A standard layout can be created from a face model component scene, such as the example provided in the rigBuilder projects resources folder (../rigBuilder/resources/face/templates/blinky/model/compo-nent/face/100/base/ma/v001/blinky\_model\_face\_100\_base\_v001.ma).

The layout tool looks for a root named *'model\_face\_100\_base’* (specified in rigBuilder.face.faceEnv) and iterates over the meshes in the hierarchy creating a duplicate mesh for every shape specified in the JSON file in the rigBuilder resource folder (../rigBuilder/resource/face/blendShapeLayout.json).

The meshes are sorted into subfolders generated from the name of the original mesh they were created from. For example:

model\_face\_100\_shape

+-------- C\_face\_grp\_0

| +-------- C\_someShapeName\_shp\_0

| +-------- C\_anotherShapeName\_shp\_0

| +-------- …

+-------- C\_teethUpper\_grp\_0

| +-------- C\_someShapeName\_shp\_0

| +-------- C\_anotherShapeName\_shp\_0

| +-------- …

+-------- …

This structure is very important because it directly influences the process of connecting the shapes to the base geometry when the rig is built.

Blend Shape Tools

The process of creating facial blend shapes is largely a bespoke one. But the Face Rig Builder has some tools to help with the process.

**Invert Blend Shape Weights (Envelope / Targets)**

This tool inverts the painted blend shape weights on related blend shape nodes of a selected mesh. *Envelope* mode will invert the blend shape nodes envelope weighting, whereas Targets mode will invert the blend shape weighting of individual target weights, if the shapes weight is set to 1.0.

**Create Taper Shapes (Envelope / Targets)**

This tool allows for the easy creation of left and right hand side blend shapes from the one mesh. Using blend shape weights, the tool will create a duplicate mesh for every target with weight set to 1.0, as well as a duplicate of the mesh with inverted blend shape weighting. This allows you to paint one side of the mesh to 1.0 and the other to 0.0 and output the result for each side. By painting the blend shapes envelope weights the rigger can make the process extremely simple to output the whole range of “mirrored” shapes from one mesh.

**Split Targets XYZ**

This tool splits any target set to 1.0 on a related blend shape node of a selected mesh, into its individual X, Y and Z components. This is particularly useful when you want to have separate animation curves driving a particular axis, allowing you to reduce the linear behaviour of a blend shape target.

Blend Shapes in the Face Rig

As mentioned earlier, the publish blend shape scene can have deformer history if the rigger desires, with the condition that any additional DAG nodes are parented under the top node *model\_face\_100\_shape.* Upon building a WIP face rig, the blend shape scene will be referenced into the scene and parented under the *|rig\_face\_anim|geometry\_grp* node. The shape targets will be connected to the base face geometry, based upon the hierarchy underneath the top shape node.

The blend shape file reference can be altered, republished and then updated in the WIP face rig if required, without the need to rebuild the rig.

When a face rig is published the reference will be imported into the scene, and the contents of the scene will be deleted leaving only the blend shape node connected to the output geometry. This means that any construction history on the blend shape targets will be rendered redundant upon publish of a face rig asset.

Publishing Blend Shapes

You can only publish blend shapes from a clean scene. Do not try and publish from a built face rig. The scene can be published with deformers on shapes, as long as any additional dag nodes are parented under the *|model\_face\_100\_shape* node. The scene will refuse to publish otherwise.

# Building the Face Rig

The minimum requirement for building a face rig is a face model, and a face skeleton guide. This allows for the face rig modules to be constructed, and rough weighting could be initiated. Ideally though you would also want a GUI guide to create the control system, and at least a basic set of blend shapes (even proxies will do), so that you can setup the initial control connections.

Modules

Face Rig Modules are python class wrappers for a large set of complex instructions. The eye module at this stage is the most complex, weighing in at some 1300 lines of code. You can find the modules in the rigBuilder.face.modules package, with the base class RigModule found within the \_\_init\_\_ module. By leveraging python’s inheritance, it provides a predictable structure to every face module.

The modules are instantiated within the rigBuilder.face.faceCore module, which is where all the elements of the face rig come together to build the finished product.

Creating a Module

To create a module, you need to create a python module in the rigBuilder.face.modules package, let’s say for examples sake nose.py. Within this module you would create a python class called NoseModule (again the name is irrelevant, it should just be unique), that inherits the RigModule class:

import rigBuilder.face.modules

class NoseModule (rigBuilder.face.modules.RigModule):

def \_\_init\_\_(position=’C’, name=’nose’, \*args, \*\*kwargs):

super(NoseModule, self).\_\_init\_\_(position, name)

’’’ Note that the only thing that the RigModule requires is a module position from [‘C’, ’L’, ‘R’], and a unique name string. The rest is up to you. ’’’

’’’ In the \_\_init\_\_ you can handle any custom arguments you might require for your module. ’’’

def buildModule(self):

’’’ This is where you would put the code that builds the rig module. ’’’

return True

|interface\_grp & |preference\_grp

The RigModule class has two variables, *self.interfaceNode* & *self.preferenceNode,* which are created when the class is instantiated. Each contains a reference to a PyMEL Transform object that is contained within the module’s hierarchal structure. These nodes are the module’s way of connecting to the rig. The interface node, *self.interfaceNode*, is where you would add any attributes that relate to driving parts of the module. These would be what you would connect controls to for example. The preference node, *self.preferenceNode* is where you would add any preferential attributes, like multipliers, minimum or maximum range values.

When the face rig is built, the modules are individually constructed, and then the interface and preference attributes are copied from the modules up to the rigs equivalent nodes *|interface\_grp* & *|preference\_grp* to make one convenient location to make connections, and set and export values to/from.

# Driven Keys

A rigger can setup driven keys via Maya’s standard Set Driven Key interface. Ideally these would be connections between the GUI controls and the blend shape and interface node, however you are not limited to those options exclusively. The Driven Key Exporter will export all Driven Keys in the WIP facial rig scene, including those connected via *blendWeighted* nodes, and will attempt to reconstruct them on any subsequent rebuild, provided of course that the exported attributes exist within the scene. This gives a rigger an enormous scope to connect any number of deformation options, and to have the ability to refine the behaviour through adjustment of the animation curve behaviour.

# Preferences

The rig module preference attributes on the *preferences\_grp* node will be exported, and imported on rebuild of the face rig, provided of course that the exported attributes exist within the scene.

# Weighting

The weighting exporter will export the skinning data for all skinClusters attached to meshes in the *model* namespace, as well as any control position data in the scene. These will be imported on rebuild of the face rig, provided of course that the necessary elements exist within the scene.