

MEL for character rigging

One of the many uses to which MEL is often put is to ease in the process of character rigging and animation. While this often means making GUIs for the selection of character controls, it can also mean creating skeletons and setting up controllers in the first place.

Making bones

Creating a bone via MEL is relatively straightforward. To create a bone, call the joint command and specify a position with the -p flag. If you don't have anything selected, you'll end up with a new bone at the specified location. If you have a joint selected when calling the joint command, though, the new joint will be created as a child of the selected joint.

This means that when making skeletons with MEL, you'll often need to move around in the hierarchy of the skeleton as you create the bones. For that, use the pickWalk command, which will have the same result as having used the arrow keys from within Maya. For example, to create a branching structure of three bones, you could do the following:

```
string $firstBone = ` joint -p 0 0 0 `; // create root bone
string $secondBone = ` joint -p 1 0 2 `; // create a child bone that's one unit to the
right
pickWalk -direction up ; // move up the hieracrhy one step, selecting
the root bone
string $thirdBone = ` joint -p 1 0 -2 `; // create the third bone
```

For more complex applications, it may make more sense to just store the root bone in a variable and select it directly, rather than using pickWalk.

Note also that the joint command has a -r (relative) flag that will cause the position of the new joint to be relative to that of its parent. Once you've got bones set up, you'll likely want to set up controls for them...

Automating Set Driven Key setup

For items like hands, it's common to create custom attributes and use them to drive the rotational values of bones via Set Driven Key. The first step is to create the custom attribute, and that can be done with the following line of code (assuming the object to which to add it is selected):

```
addAttr -ln curl -keyable true -at double -min 0 -max 10; // creates a floating-point
value named // "curl"
that ranges from 0 to 10
```

However, by default, the new attribute isn't keyable, meaning that it doesn't show up in the channel box. To fix that, either include the -keyable flag with a value of true, or follow the above line of code with the following (assuming the object has been stored in \$sel[0]):

```
setAttr -edit -keyable true ($sel[0]+".curl"); // causes the curl attribute to be
keyable
```

The above line can also be used to make attributes not keyable, taking them out of the channel box, which might be useful to reduce clutter. Once you have the new attribute, you'll want to set it up to control something, likely with set driven key. The process of setting up set driven key with MEL has the same steps as doing it normally in Maya, in that you:

- set the controller value to its minimum
- set the controlled value to its minimum
- set a key
- set the controller value to its maximum
- set the controlled value to its maximum
- set a key

All of the above resolves to two MEL commands, each used multiple times. The setAttr command can be used to set the values of the attributes, and the setDrivenKeyframe can be used to set the keyframes themselves. So if you had a curl attribute on \$sel[0] that went from 0 to 10 and you wanted to make it control the Z-rotation of joint7 from 0 to -45, you could use the following code:

```

setAttr ($sel[0]=".curl") 0; // set controller to min
setAttr ("joint7.rotateZ") 0; // set controlled to min
setDrivenKeyframe -cd ($sel[0]+".curl") ("joint7.rotateZ"); // set driven key
setAttr ($sel[0]+".curl") 10; // set controller to max
setAttr ("joint7.rotateZ") -45; // set controlled to max
setDrivenKeyframe -cd ($sel[0]+".curl") ("joint7.rotateZ"); // set driven key

```

The -cd flag of the setDrivenKeyframe specifies the "current driver", or the controller object- no flag is required to specify the controlled object.

Setting up expressions with MEL

In addition to setting up set driven key, it often makes sense to tie custom attributes to expressions, which can also be done via MEL. Writing the expression itself is done in the same way as in the expression editor, with the difference being that the entire expression is passed as a string to the expression command, which actually makes the expression. Since creating the string will often involve complex combinations of string variables and literals, it's generally safer to create the expression in a temporary string variable and pass that to the expression command. For example, the following code will create a custom attribute and use it in an expression to make the selected object bob up and down:

```

string $sel[] = `ls -sl`;

// adds a speed attribute to the selected object
addAttr -ln speed -keyable true -at double;

// creates an expression string equivalent to:
// "myObj.translateY = sin(time * myObj.speed);"
string $exp = ($sel[0]+".translateY = sin(time * " + $sel[0]+".speed ) ;");

// creates the expression from the above string
expression -s $exp;

```