

Writing Scripts

Once you've gotten a feel for how MEL works by working with buttons and expressions, you'll want to move over to writing your own custom scripts. You'll most likely want to create them in Notepad or similar (a bare-bones text editor), rather than in Maya itself, since that way you can avoid the script editor's tendency to disappear your code

Script files defined

In order for Maya to successfully recognize and run your scripts, there are a few requirements placed on the files. First off, they have to be in one of the locations that Maya searches for scripts. Generally this will mean placing your script files in:

My Documents/Maya/*versionNumber*/scripts

Furthermore, in order for the script files to be recognized, their name must end in ".MEL", and they must be saved in plain text format. While you *could* try to create your scripts with something like Word, that would run the risk of inserting additional formatting information that would confuse Maya. The safest and easiest way to go about it is just to use the simplest text editor available (i.e. Notepad).

If you're downloading scripts from the internet, they should be the proper format but, just as with your own scripts, you'll need to place them in the above directory in order for Maya to find them.

Running scripts

In order to run a script once you've either installed it (by placing it in the proper location) or written it, all you need do is to type the name of the script in Maya's command line. The extension is not necessary. So if I had a file called "MyScript.mel", and I had the file in the "scripts" folder, I could type "MyScript" into the command line region and hit enter to get Maya to execute the script.

What actually happens when you type in the name of a script and hit enter is that Maya starts looking in all the places it knows to look for a script file whose name matches what you typed. So in the above example, Maya would start combing through its scripts folders looking for "MyScript.mel". Assuming that Maya found it, and it was a valid mel file (called .mel but saved in text format), Maya would crack the file open and look inside...

Inside a script

Once Maya has found the script, it will attempt to run it. In order for that to work, Maya must find a *procedure* of the same name as the script file. A procedure is simply a collection of MEL statements that collectively achieve some goal. While you can (and likely will) have several procedures in a script, you *must* always have at least one, and you must have one that has the same name as the file. So, every script starts with the following:

```
global proc myScriptName ()  
{  
    Script code goes here  
}
```

The "global proc" tells Maya that it is a procedure, and that it is globally accessible (available to things outside itself, meaning that it can be called from Maya). The parentheses are there to provide a way of including user input, but must be there even if you're not getting input in that manner. Then we see a pair of curly brackets ({, }). In MEL, as in other languages, these are used to delineate "blocks" of code. A block of code is simply a collection of statements. There are many situations and circumstances that give rise to having blocks of code, and it's common to have blocks of code nested within other blocks. For now though, just know that all scripts must have at least one block of code (not much point in a code-less script).

If Maya cracks open the file and sees a procedure of the same name as the file, it will start executing each line of code contained in the script, starting with the top line and working its way down.

Modifying scripts

One of the counter-intuitive things about learning MEL is that if you write a script, run it once to test, and then make changes to it, Maya will continue to run the old version. While this seems both arbitrary and annoying at first glance, there is a good reason for it. In order to be more efficient in memory use, Maya only loads a script into memory the first time you call it. After that, it just does what it did the last time. So even if you've made changes and saved them, Maya will be working from the old set of instructions.

Luckily, Maya gives us the ability to force it to re-load scripts by using the "source" command. The source command will cause Maya to load the most current version of your script into memory, therefore bringing it up to date. For ease of use, it's a good idea to make a shelf button that sources your script and then runs it, as you'll constantly be wanting to do those two things in rapid succession. If I was working on something called "MyScript", I would want a button that included the following two lines of code:

```
source MyScript.mel;  
MyScript;
```

Notice that including the .mel extension, while not necessary to actually run the script (the second line), is necessary in order to source the script (the first line).

Writing scripts- variables and simple output

Okay, so once you've got a script working, you'll want to make it do something interesting. Here's where it starts to get fun. You have access to all of the commands listed in the MEL documentation, as well as a few things that aren't really MEL commands per se, but are nonetheless vital to writing scripts. One such class of things are the commands to declare variables.

A variable can be thought of as four things:

- a name
- a value
- a type
- a position in memory

In other words, a variable is a place in the computer's memory that is reserved for holding a certain kind of information, and that can be referred to by a name you give it.

One "declares" variables by invoking the keyword for the type of variable you want, giving it a name, and (optionally) giving it a starting value. The list of available variable types that you can have in Maya are the same as the available attribute types when adding custom attributes, and include integers, floats, strings, and booleans. In order to create an integer variable, you would do the following:

```
int $myInt ;
```

...which would create an integer variable named "\$myInt". Alternatively, you could also do this:

As you manipulate variables, you'll want to have some way of finding out what your program is doing. One of the easiest ways to go about that is to use the print command to spit out information as to what's happening when. The print statement will take whatever it's given and output it to the upper half of the script editor. If I was writing a script that involved performing a lot of math on the \$myInt variable, its likely that I would want the following line of code appearing multiple places in the script:

```
print($myInt);
```

While the above would work, it would quickly become confusing as to where in the code each output was coming from. In order to fix that, you'd want to use the following approach:

```
print("The value of the variable at line 5 is: " + $myInt);
```