**PYMEL**

**Tighter MEL Integration**

Calling MEL from python is still an unfortunate necessity, so PyMEL makes it as easy as possible. This release builds on PyMEL's already vastly improved method, which allows you to call a mel procedure as if it was a python function:

values = ['one', 'two', 'three', 'four']

# default

maya.mel.eval( 'stringArrayRemoveDuplicates( {"'+'","'.join(values)+'"})')

# PyMEL

mel.stringArrayRemoveDuplicates( values )

In the new release, when a MEL script called from PyMEL raises an error, you will get the specific MEL error message in the python traceback, along with line numbers!

For example, here's a procedure "myScript" with a line that will result in an error:

>>> mel.eval( '''global proc myScript( string $stringArg, float $floatArray[] ){

... float $donuts = `ls -type camera`;}''')

When we call the procedure via PyMEL, we can quickly determine the problem, because PyMEL gives us the error and the line number:

>>> mel.myScript( 'foo', [] )

Traceback (most recent call last):

...

MelConversionError: Error occurred during execution of MEL script: line 2: Cannot convert data of type string[] to type float.

Also, getting and setting MEL global variables is accomplished via a special dictionary-like object, which keeps it simple and intuitive:

>>> melGlobals['$gMainFileMenu']

mainFileMenu

>>> melGlobals['$gGridDisplayGridLinesDefault'] = 2

### For the MEL Scripter

When we say PyMEL is concise and easy to read, we mean it.

#### MEL

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

string $shapes[] = `listRelatives -s $sel[0]`;

string $conn[] = `listConnections -s 1 -d 0 $shapes[0]`;

setAttr ( $conn[0] + ".radius") 3;

#### PyMEL

selected()[0].getShape().inputs()[0].radius.set(3)

## Powerful Classes

Node classes for every node type

camTrans, cam = camera() # create a new camera

cam.setFocalLength(100)

fov = cam.getHorizontalFieldOfView()

cam.dolly( -3 )

cam.track(left=10)

cam.addBookmark('new')

An Attribute class organizes all the attribute commands in one place

s = polySphere()[0]

if s.visibility.isKeyable() and not s.visibility.isLocked():

s.visibility.set( True )

s.visibility.lock()

print s.visibility.type()

Manipulate file paths with ease

#backup all mb files in the current scene's directory

basedir = sceneName().parent

backupDir = basedir / "backup" #slash op joins paths

if not backupDir.exists:

backupDir.mkdir()

for file in basedir.files( '\*.mb' ):

print "backing up: ", file.name

file.copy( backupDir / (file.namebase + ".old") )

Work with shape components, perform vector math, and easily set object attributes with the results

#select all faces that point up in world space

s = polySphere()[0]

for face in s.faces:

if face.getNormal('world').y > 0.0:

select( face, add=1)

Manage optionVars as a python dictionary

if 'numbers' not in optionVar:

optionVar['numbers'] = [1,24,47]

optionVar['numbers'].append(9)

numArray = optionVar.pop('numbers')

PyMEL provides customized operators for succinct scripting:

cam = camera()[0]

sphere = polySphere()[0]

sphere | cam # parent the camera to the sphere

cam.tx >> cam.ty # connect operator

cam.tx // cam.ty # disconnect operator

## Includes Tools to Ease Your Transition to Python

* **mel-to-python translator** for converting mel script into python scripts
* **python-to-mel plugin factory** for turning python classes into mel commands
* **ipymel** customized python interpreter