# PoserPython Methods Manual

for Windows and Mac OS X®

POSER® | 8
Easily Create 3D Character Art and Animation

## Trademark and Copyright Notice

Please visit http://poser8.smithmicro.com for updates and changes to this manual.”

# Chapter 1: Basic PoserPython Concepts

This section describes some of PoserPython's basic concepts in order to provide context.

## How Python Integrates with Poser

There are two basic types of programming languages:

- ***Compiled***: A special program called a compiler reads the code written by the programmer, translates it into a form directly readable by a computer, and creates an executable program that can run on its own on any supported computer platform. Languages such as C++ fit into this category.

- ***Interpreted***: Interpreted programming languages require a special program called an interpreter to run the code written by the programmer. The interpreter reads the commands found in the code and executes them from beginning to end without translating the original code. The drawback to interpreted languages is that they must use the interpreter, and the code can never run as a standalone program. The advantage, however, is that a programmer can change the code and see the results very quickly without having to wait for the code to compile. Additions, edits, and deletions become fast and simple.

PoserPython consists of a standard Python interpreter that has been extended to recognize and execute commands that are not included with the standard Python language. PoserPython scripts written using the customized commands will only work with the Poser ProPack or subsequent versions (5 and on). You can, however, pass data back and forth between Poser and other Python scripts, programming languages, and applications.

The standard Poser application contains volumes of data about each item (figure, scene, light, camera, prop, etc.) found within a given scene. You control these parameters via the mouse and interface elements such as menus, buttons, dials, etc. However, you cannot directly manipulate the data itself. This helps make Poser easy to use, but does create limitations. For example, you have no way to automate repetitive tasks or record a complex series of actions for further use. PoserPython circumvents these limitations.

PoserPython exposes much of the raw Poser data. Using Python scripts, you can extract data from Poser, manipulate it, and pass it back into Poser. Essentially, you can program Poser to do what you want, and you are no longer confined to the Poser interface and Poser's built-in functionality.

## Basic Python Structure

Python is an object-oriented language. An object is a virtual entity combining structured data and the methods in which the data can be manipulated. A method is a procedure for manipulating data, and an argument defines how the method is carried out. A simplistic but effective analogy is that of basic grammar: An object can be equated with a noun, a method with a verb, and an argument with an adjective or adverb. For example, consider the following:

```
car = Factory.Produce(vehicleXL)
```

In this case, the variable car is the result of the object factory being acted upon by the method produce as modified by the argument vehicleXL (the make and model). To put it in lay terms, the car's existence and everything about the car depends on the factory being told to produce a car of a specified make and model. Consider how the value of the variable car can differ based on the following examples:

- car = `Mechanic.Repair(car, transmission)`
- `car = Junkyard.Salvage(vehicleXL)`
- `car = CarThief`.Steal()

In the first example the car argument is passed in, modified by the Mechanic's Repair method, and returned as a working car. The last example contains no argument. In this case, the car thief may not take external input to decide which car to steal. Again, the object defines a structured set of data, the method is what the object does, and any arguments describe how the method is performed.

car may contain either data or a reference or link to data. Please refer to one of the many Python resources (such as those listed in Other Important Resources on page 8) for an in-depth discussion of mutable vs. immutable types. The data can be virtually anything including letters, numbers, files, etc. As you begin to think of data in terms of objects and manipulating objects, you will find it far easier and faster to write Python scripts.

## Sample Python Script

This section provides a brief example of how a Python script might look. For this example, let's say that you have an open Poser scene consisting of a figure with its left forearm already selected. The forearm is called an actor. An actor is any element of a Poser scene (body part, prop, etc.) and this manual uses the two terms interchangeably. Let's say you want to set the x scale to 88 percent.

```
scene = Poser.Scene()
actor = Scene.CurrentActor()
parm = actor.ParameterByCode(Poser.ParmCodeXSCALE)
parm.SetValue(88)
```

Let's look at the above script in detail:

The script begins by obtaining a variable called `scene`, which is a reference to the current Poser scene. That scene contains numerous actors. Recall that the left forearm is already selected, so all the script needs to do is request the scene's current actor to define the variable actor. Next, the variable parm consists of a reference to the left forearm's specified parameter, in this case the parameter XSCALE. A parameter code (`ParmCode`) designates an easy to remember word to signify the desired parameter. Lastly, the value of the parameter to which `parm` refers is reset to 88, which will cause the left forearm to shrink to 88% of its normal size along the X-axis.

## Writing Python Scripts

Write your Python scripts using your favorite text editor.

## wxPython

wxPython allows you to create add-ons that integrate very well with Poser. You can also add your own palettes to the Poser GUI.

Refer to `Runtime:Python:poserScripts:ScriptsMenu:Python Shell.py` for implementation basics.

The most important rule: Retrieve managed window from Poser, don't create your own main event loop.

## Folder Syntax

Python, among other languages, designates certain special characters as symbols beginning with a Windows-style backslash. Two common examples are `\t` ([TAB]) and \n (new line). Thus, in Windows, the notation

```
C:\folder\test.txt
```

is interpreted by Python as

```
C:[TAB]folder[TAB]text.txt or C:
                folder              test.txt
```

To work around this issue, you should use a double backslash (\\) to signify folders. The above example would then become:

```
C:\\folder\\test.txt
```

which would be properly interpreted to mean a specific file path.

Alternatively, you can use Macintosh format, which uses colons to signify folders as follows:

```
:folder:test.txt
```

## Running Your Script

You can run your script directly or via the Poser **Scripts** menu or palette, all of which are described in the following chapter.

## For Further Information

The preceding information was a very brief glimpse into Python and the PoserPython extensions. If you do not currently know Python, you may want to read and learn more. With the power and flexibility this interface offers, developing Python expertise would be well worth it.

To view the Poser Python Methods Manual, choose **Help >PoserPython Manual** from the Poser 8 menu commands.

# Chapter 2: PoserPython Types & Methods Listing

This chapter lists the custom PoserPython types, codes, constants, and methods in a tabular format. Many of these items correspond to items that are controllable from the Poser interface. Please refer to your Poser Reference Manual for information on each parameter.

## Types

A type is a category of data. Python includes several data types including IntType (integers), FloatType (floating decimal point numbers), StringType (alphanumeric), and NoneType (the type of the special constant None). The PoserPython extensions add the following data types to the standard Python types:

| | |
|---|---|
| **ActorType** | This type of data represents an actor within a figure or scene. Note the term "actor" refers to any individual item that can move, including body parts, cameras, lights, props, etc. Thus, a forearm is an actor, as is hair. A body is a figure. |
| **AnimSetType** | An animation set |
| **ClothSimulatorType** | ClothSimulator data |
| **DictType** | An option dictionary |
| **FigureType** | A figure within a scene |
| **FireFlyOptionsType** | FireFly renderer options data |
| **FunctionType** | Pointer to a callback function |
| **GeomType** | Geometry forming part of a figure or prop |
| **HairType** | Hair data |
| **ImExporterType** | Importer/Exporter data |
| **InputType** | A node input |
| **MaterialType** | Material data |
| **MovieMakerType** | MovieMaker data |
| **ParmType** | A parameter (such as scale, twist, bend, etc.) |
| **PolygonType** | Polygons (geometry surface faces) |
| **SceneType** | A Poser scene |
| **ShaderNodeInputType** | Shader Node Input data |
| **ShaderNodeType** | Shader Node data |

| | |
|---|---|
| **ShaderTreeType** | A ShaderTree |
| **TexPolygonType** | Texture polygon data |
| **TexVertType** | Texture vertices data |
| **TupleType** | A Tuple |
| **VertType** | Vertices (points defining the edges of polygons) |

These additional data types let both you and Poser know exactly what kind of data is being manipulated and what can and cannot be done to that data.

## Codes

In PoserPython, a code is a representation of a given parameter such as a coordinate, display mode, light attribute, etc. These codes make it easy to access Poser's internal data and also make it easy to know which parameter is being called or set. When using them in your scripts, make sure to prefix them with "poser." so that the PoserPython interpreter understands that they are predefined PoserPython member variables. For example, one might call scene.SetDisplayStyle(poser.kDisplayCodeCARTOONNOLINE) when setting the scene's display style.

## Cloth Simulator Codes

These codes are used by a Cloth Simulator. They are typically used in conjunction with the DynamicsProperty() method, to return a property of the specified type. Please refer to the Poser Reference Manual, "Chapter 29: The Cloth Room" for more information about Cloth Simulators.

**`kClothParmCodeAIRDAMPING`**

The Air Damping parameter specifies the cloth's air resistance that occurs whenever the cloth is moving through the air.

**`kClothParmCodeCLOTHCLOTHFORCE`**

The ClothClothForce parameter.

**`kClothParmCodeCLOTHFRICTION`**

The Cloth Self Friction parameter sets the coefficient of friction between one part of the cloth and another, or how easily the cloth moves over itself.

**`kClothParmCodeDAMPINGSTRETCH`**

The Stretch Damping parameter controls the internal energy loss caused by the motion of the cloth fibers against each other.

**kClothParmCodeDENSITY**

The ClothDensity parameter specifies the mass-per-unit area density of the cloth in grams per square centimeter.

**kClothParmCodeDYNAMICFRICTION**

The Dynamic Friction parameter sets the coefficient of friction between the cloth and solid objects when the cloth is in motion.

**kClothParmCodeFRICTIONFROMSOLID**

Enabling Collision Friction ignores the cloth object's Static Friction and Dynamic Friction parameters, instead using those same parameters belonging to the collision objects themselves.

**kClothParmCodeFRICTIONVELOCITYCUTOFF**

The Friction Velocity Cutoff parameter sets the friction velocity cutoff value.

**kClothParmCodeSHEARRESISTANCE**

The Shear Resistance parameter controls the cloth's resistance to in-plane shearing, or side-to-side bending.

**kClothParmCodeSPRINGRESISTANCE**

The Spring Resistance parameter specifies the cloth's spring resistance value.

**kClothParmCodeSTATICFRICTION**

The Static Friction parameter sets the amount of friction between the cloth and solid objects.

**kClothParmCodeTHICKNESS**

The Thickness code constitutes the combination of the cloth's Collision Offset and Collision Depth parameters.

**kClothParmCodeUBENDRATE**

The Bend Rate parameter operating on the U coordinate axis.

**kClothParmCodeUBENDRESISTANCE**

The Fold Resistance parameter specifies the resistance to out-of plane bending (folding). The UBendResistance code specifies the U coordinate axis of the Fold Resistance parameter.

**kClothParmCodeUSCALE**

The Scale parameter operating on the U coordinate axis.

**kClothParmCodeUSEEDGESPRINGS**

The UseEdgeSprings parameter sets whether or not the cloth will use edge spring calculations.

**kClothParmCodeUSTRETCHRESISTANCE**

The Stretch Resistance parameter specifies the cloth's resistance to in-plane bending (stretching). The UStretchResistance code specifies the U coordinate axis of the Stretch Resistance parameter.

**`kClothParmCodeVBENDRATE`**

The Bend Rate parameter operating on the V coordinate axis.

**`kClothParmCodeVBENDRESISTANCE`**

The Fold Resistance parameter specifies the resistance to out-of plane bending (folding). The VBendResistance code specifies the V coordinate axis of the Fold Resistance parameter.

**`kClothParmCodeVSCALE`**

The Scale parameter operating on the V coordinate axis.

**`kClothParmCodeVSTRETCHRESISTANCE`**

The Stretch Resistance parameter specifies the cloth's resistance to in-plane bending (stretching). The VStretchResistance code specifies the V coordinate axis of the Stretch Resistance parameter.

## Dialog Codes

Dialog codes. They are typically used in conjunction with the DialogFileChooser( ) method.

**`kDialogFileChooserOpen`**

Brings up a standard File Open dialog window.

**`kDialogFileChooserSave`**

Brings up a standard File Save dialog window.

## Display Codes

Display codes specify the display mode, normally set by the Display Styles palette. They are typically used in conjunction with the scene. SetDisplayStyle() method.

**`kDisplayCodeCARTOONNOLINE`**

Cartoon with no line display mode.

**`kDisplayCodeEDGESONLY`**

Outline display mode.

**`kDisplayCodeFLATLINED`**

Flat Lined display mode.

**`kDisplayCodeFLATSHADED`**

Flat Shaded display mode.

**`kDisplayCodeHIDDENLINE`**

Hidden Line display mode.

**kDisplayCodeSHADEDOUTLINED**

Shaded Outline display mode.

**kDisplayCodeSILHOUETTE**

Silhouette display mode.

**kDisplayCodeSKETCHSHADED**

Sketch Shaded display mode.

**kDisplayCodeSMOOTHLINED**

Smooth Lined display mode.

**kDisplayCodeSMOOTHSHADED**

Smooth Shaded display mode.

**kDisplayCodeTEXTURESHADED**

Texture Shaded display mode.

**KDisplayCodeUSEPARENTSTYLE**

The style of the actor/figure's parent.

**KDisplayCodeWIREFRAME**

Wireframe display mode.

## FireFly Options Codes

These FireFly Options codes specify various types of post filter algorithms, and define the pen styles used to create the toon outlines.

**kOutlineCodeMEDIUMMARKER**

Specifies the Medium Marker toon outline style.

**kOutlineCodeMEDIUMPEN**

Specifies the Medium Pen toon outline style.

**kOutlineCodeMEDIUMPENCIL**

Specifies the Medium Pencil toon outline style.

**kOutlineCodeTHICKMARKER**

Specifies the Thick Marker toon outline style.

**kOutlineCodeTHICKPEN**

Specifies the Thick Pen toon outline style.

**kOutlineCodeTHICKPENCIL**

Specifies the Thick Pencil toon outline style.

**kOutlineCodeTHINMARKER**

Specifies the Thin Marker toon outline style.

**`kOutlineCodeTHINPEN`**

Specifies the Thin Pen toon outline style.

**`kOutlineCodeTHINPENCIL`**

Specifies the Thin Pencil toon outline style.

**`kPixelFilterCodeBOX`**

Box Post Filter Type

**`kPixelFilterCodeGAUSS`**

Gaussian Post Filter Type

**`kPixelFilterCodeSINC`**

Sinc Post Filter Type

**`KRayAcceleratorCodeHBVO`**

Selects the HBVO ray accelerator.

**`KRayAcceleratorCodeKDTREE`**

Selects the kd tree ray accelerator.

**`KRayAcceleratorCodeVOXEL`**

Selects the Voxel ray accelerator.

**`KTextureCompressorCodeRLE`**

Selects the Run-Length Encoding (RLE) format for texture compression.

**`KTextureCompressorCodeZIP`**

Selects the ZIP format for texture compression.

## Import/Export Codes

The PoserPython options dictionary now includes enumerations of import and export options. These options correspond to UI option strings present in dialog boxes when importing/exporting files from within Poser using the normal interface. For users who need access to the strings as they appear in the dialog boxes, you can query the strings by passing the enumeration constant into the `ImportOptionString()` and `ExportOptionString()` methods (discussed below).

Codes such as `poser.kExOptCodeMULTIFRAME` are pre-defined constants with unique identifying values. For instance,

```
poser.kImOptCodeCENTERED = 0
poser.kImOptCodePLACEONFLOOR = 1
poser.kImOptCodePERCENTFIGSIZE = 2
poser.kImOptCodeOFFSETX = 3
poser.kImOptCodeOFFSETY = 4
```

etc.

The values 0 to 4 do not represent the values or choices the options are set to, but rather, they are simply codes uniquely identifying each option. It is unlikely that you will ever need to know or set them. A more typical use of import/export option enumeration values is illustrated in the following line of code:

```
options[poser.kExOptCodeFIRSTFRAME]
= 2
```

The above example sets the value of the FirstFrame option to 2.

**kExOptCodeASMORPHTARGET**

Saves current figure/prop as a morph target.

**kExOptCodeAUTOSCALE**

Set automatic scaling for BVH export.

**kExOptCodeBODYPARTNAMESINPOLYGROUPS**

Embed body part names in polygon groups.

**kExOptCodeBROADCASTKEY**

Viewpoint broadcast key.

**kExOptCodeEXISTINGGROUPSINPOLYGROUPS**

Keep existing polygon group names.

**kExOptCodeFIGNAMESINGROUPS**

Keep figure names in groups.

**kExOptCodeFIRSTFRAME**

First frame of export.

**kExOptCodeGENERATEHTML**

Viewpoint option generate HTML file with MTS/MTL.

**kExOptCodeGEOMQUALITY**

Viewpoint export geometry quality.

**kExOptCodeGROUPSPERBODYPART**

Keep body part groups.

**kExOptCodeHTMLVIEWPORTHEIGHT**

Viewpoint option HTML window height.

**kExOptCodeHTMLVIEWPORTWIDTH**

Viewpoint option HTML window width.

**kExOptCodeIGNORECAMERAANIM**

Viewpoint export ignores camera motions.

**kExOptCodeIMAGEQUALITY**

Viewpoint export image quality.

**kExOptCodeLASTFRAME**

Last frame of export

**kExOptCodeMULTIFRAME**

Multiple frame export.

**kExOptCodeSAVECOMPRESSED**

Save files in compressed format.

**kExOptCodeSCALEFACTOR**

Scale exported scene by this amount.

**kExOptCodeSOFTEDGESINHTML**

Viewpoint export soft HTML window edges

**kExOptCodeUSEANIMSETS**

Viewpoint export use animation sets.

**kExOptCodeUSEINTERNALNAMES**

Use internal names.

**kExOptCodeUSEWAVELETTEXTURES**

Viewpoint option use wavelet textures.

**kExOptCodeWELDSEAMS**

Welds seams in figures/props.

**kImOptCodeARMALIGNMENTAXIS**

Arm alignment axis

**kImOptCodeAUTOSCALE**

Automatic scaling

**kImOptCodeCENTERED**

Center object

**kImOptCodeFLIPNORMS**

Flip normals.

**kImOptCodeFLIPUTEXTCOORDS**

Flip U texture coordinates.

**kImOptCodeFLIPVTEXTCOORDS**

Flip V texture coordinates.

**`kImOptCodeMAKEPOLYNORMSCONSISTENT`**

Make polygon normals consistent.

**`kImOptCodeOFFSETX`**

X offset amount.

**`kImOptCodeOFFSETY`**

Y offset amount.

**`kImOptCodeOFFSETZ`**

Z offset amount.

**`kImOptCodePERCENTFIGSIZE`**

Figure scale in percent

**`kImOptCodePLACEONFLOOR`**

Place object on floor

**`kImOptCodeWELDIDENTICALVERTS`**

Weld identical vertices.

## Language Codes

Language codes are codes representing the various languages for which this copy of Poser may be localized.

**`kLanguageCodeFRENCH`**

Return value for the Poser.Language() method (described below)

**`kLanguageCodeGERMAN`**

Return value for the Poser.Language() method (described below)

**`kLanguageCodeJAPANESE`**

Return value for the Poser.Language() method (described below)

**`kLanguageCodeUSENGLISH`**

Sets US English as the default language.

## Light Codes

These codes are used to set the light types.  They are typically used in conjunction with the actor.SetLightType() method.

**`kLightCodeIMAGE`**

Image Based light.

**`kLightCodeINFINITE`**

Infinite lightLocal light.

**kLightCodeINVLINEARATTEN**

Sets Light Attenuation to Inverse Linear

**kLightCodeINVSQUAREATTEN**

Sets Light Attenuation to Inverse Square

**kLightCodePOSERATTEN**

Sets Light Attenuation to Poser default (Constant)

**kLightCodePOINT**

Point light.

**kLightCodeSPOT**

Spotlight.

## Palette Codes

These codes are used to specify specific palettes within Poser.

**kCmdCodeAPPLYBULGES**

Enables the Apply Bulges checkbox on the Joint Editor. Note: This constant is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or future availability.

**kCmdCodeANIMATIONPALETTE**

Returns the Animation palette.

**kCmdCodeGROUPPALETTE**

Returns the Group Editor palette.

**kCmdCodeJOINTPALETTE**

Returns the Joint Editor palette.

**kCmdCodeLIBRARYPALETTE**

Returns the Library palette.

**kCmdCodeLIBRARYPALETTEFIGURES**

Returns the Library palette, open to the Figures category. Note: This constant is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or future availability.

**kCmdCodePANDPPALETTE**

Returns the Parameters and Properties palette.

**kCmdCodeWALKPALETTE**

Returns the Walk Designer.

**kCmdCodeZEROFIGURE**

Sends the Zero Figure command to the Joint Editor; returns the figure to its original neutral pose. Note: This constant is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or future availability.

## Parameter Codes

These codes are used to specify specific Poser parameters. For example, instead of using the actor.Parameter("xTran") method, the actor.ParameterByCode(poser.kParmCodeXTRAN) can be used to return the x-axis translation parameter for the actor.

**kParmCodeASCALE**

Uniform scale parameter.

**kParmCodeCENTER**

For internal Poser use only.

**KParmCodeCLOTHDYNAMICS**

Dynamic cloth simulation parameter.

**kParmCodeCURVE**

Strength of bend deformation for an object using curve deformation.

**kParmCodeDEFORMERPROP**

Deformer prop channel.

**kParmCodeDEPTHMAPSIZE**

Parameter representing the x and y depth map resolution attached to a given light.

**kParmCodeDEPTHMAPSTRENGTH**

Intensity of shadow produced from a given light. Valid values range from 0.0 to 1.0. A value of 1.0 indicates full shadow, and 0.0 indicates no shadow.

**kParmCodeFOCAL**

Camera focal length parameter.

**kParmCodeFOCUSDISTANCE**

Camera focus distance parameter (affects Depth of Field effect).

**kParmCodeFSTOP**

Camera f-stop parameter (affects Depth of Field effect).

**kParmCodeGEOMCHAN**

For objects containing more than one possible geometry, this parameter specifies which geometry to use (such as the hands in Poser 1 and 2 figures).

**kParmCodeGRASP**

Hand grasp parameter.

**KParmCodeHAIRDYNAMICS**

Dynamic hair simulation parameter.

**kParmCodeHITHER**

Camera parameter specifying a near clipping plane distance.

**kParmCodeKDBLUE**

Blue component of the diffuse color.

**kParmCodeKDGREEN**

Green component of the diffuse color.

**kParmCodeKDINTENSITY**

Uniform intensity scale of the diffuse color.

**kParmCodeKDRED**

Red component of the diffuse color.

**kParmCodeLITEATTENEND**

Light attenuation ending parameter.

**kParmCodeLITEATTENSTART**

Light attenuation starting parameter.

**kParmCodeLITEFALLOFFEND**

Ending distance for a light's falloff zone.

**kParmCodeLITEFALLOFFSTART**

Starting distance for a light's falloff zone.

**kParmCodePOINTAT**

Degree to which an actor set to point at something will actually point at it.

**kParmCodeSHUTTERCLOSE**

Shutter closing time in fractions of a frame, where 0.0 is the beginning of the frame and 1.0 is the end of the frame. (Requires 3D Motion Blur activated to see visible effect.)

**kParmCodeSHUTTEROPEN**

Shutter opening time in fractions of a frame, where 0.0 is the beginning of the frame and 1.0 is the end of the frame. (Requires 3D Motion Blur activated to see visible effect.)

**kParmCodeSPREAD**

Hand spread parameter.

**kParmCodeTAPERX**

Amount of X-axis taper for the current actor.

**kParmCodeTAPERY**

Amount of Y-axis taper for the current actor.

**kParmCodeTAPERZ**

Amount of Z-axis taper for the current actor.

**kParmCodeTARGET**

Parameter controlling a morph target.

**kParmCodeTGRASP**

Hand's thumb grasp parameter.

**kParmCodeVALUE**

Placeholder for a value.   Usually, these values are used functionally to control other things such as full-body morphs.

**kParmCodeWAVEAMPLITUDE**

Wave object's amplitude parameter.

**kParmCodeWAVEAMPLITUDENOISE**

Wave object's amplitude noise parameter.

**kParmCodeWAVEFREQUENCY**

Wave object's frequency parameter.

**kParmCodeWAVELENGTH**

Wave object's wavelength parameter.

**kParmCodeWAVEOFFSET**

Wave object's offset parameter.

**kParmCodeWAVEPHASE**

Wave object's phase parameter.

**kParmCodeWAVESINUSOIDAL**

Wave object's sinusoidal form parameter.

**kParmCodeWAVESQUARE**

Wave object's square form parameter.

**kParmCodeWAVESTRETCH**

Wave object's stretch parameter.

**kParmCodeWAVETRIANGULAR**

Wave object's triangular form parameter.

**kParmCodeWAVETURBULENCE**

Wave object's turbulence parameter.

**kParmCodeXROT**

Rotation about the X-axis.

**kParmCodeXSCALE**

Amount of scale along the X-axis

**kParmCodeXTRAN**

Translation along the X-axis.

**kParmCodeYON**

Camera parameter specifying a far clip plane distance.

**kParmCodeYROT**

Rotation about the Y-axis.

**kParmCodeYSCALE**

Amount of scale along the Y-axis.

**kParmCodeYTRAN**

Translation along the Y-axis.

**kParmCodeZROT**

Rotation about the Z-axis.

**kParmCodeZSCALE**

Amount of scale along the Z-axis

**kParmCodeZTRAN**

Translation along the Z-axis.

## Room Codes

The following codes specify individual rooms within Poser that can be called within the PoserPython interface.

**KCmdCodeCLOTHROOM**

Specifies the Cloth room.

**KCmdCodeCONTENTROOM**

Specifies the Content room.

**KCmdCodeFACEOOM**

Specifies the Face room.

**KCmdCodeHAIRROOM**

Specifies the Hair room.

**`KCmdCodeMATERIALROOM`**

Specifies the Material room.

**`KCmdCodePOSEROOM`**

Specifies the Pose room.

**`KCmdCodeSETUPROOM`**

Specifies the Setup room.

## Scene Codes (Image Output Compression)

The following scene codes specify which image output compression will be used.

**`kTIFF_ADOBE_DEFLATE`**

Selects the Adobe DEFLATE image compression type for TIFF files.

**`kTIFF_DEFAULT`**

Selects the default TIFF image compression.

**`kTIFF_DEFLATE`**

Selects the DEFLATE image compression type for TIFF files.

**`kTIFF_JPEG`**

Selects the JPEG image compression type for TIFF files.

**`kTIFF_LZW`**

Selects the LZW image compression type for TIFF files.

**`kTIFF_NONE`**

Selects no image compression type for TIFF files.

**`kTIFF_PACKBITS`**

Selects the PACKBITS image compression type for TIFF files.

## Shader Node Codes

These codes specify types of shader nodes.  Please refer to the Poser Reference Manual, “Part 6: Materials”, for more information about these shader node types.

**`kNodeTypeCodeAMBIENTOCCLUSION`**

Specifies an Ambient Occlusion raytrace lighting node.

**`kNodeTypeCodeANISOTROPIC`**

Specifies an Anisotropic specular lighting node.

**`kNodeTypeCodeATMOSPHERE`**

Specifies the Root Atmosphere shader node.

**kNodeTypeCodeBACKGROUND**

Specifies the Root Background shader node.

**kNodeTypeCodeBLENDER**

Specifies a Blender math node.

**kNodeTypeCodeBLINN**

Specifies a Blinn specular lighting node.

**kNodeTypeCodeBRICK**

Specifies a Brick 2D texture node.

**kNodeTypeCodeCELLULAR**

Specifies a Cellular 3D texture node.

**kNodeTypeCodeCLAY**

Specifies a Clay diffuse lighting node.

**kNodeTypeCodeCLOUDS**

Specifies a Clouds 3D texture node.

**kNodeTypeCodeCOLORMATH**

Specifies a Color Math math node.

**kNodeTypeCodeCOLORRAMP**

Specifies a Color Ramp math node.

**kNodeTypeCodeCOMP**

Specifies a Component math node.

**kNodeTypeCodeDIFFUSE**

Specifies a standard Diffuse lighting node.

**kNodeTypeCodeDNDU**

Specifies a DNDU variable node.

**kNodeTypeCodeDNDV**

Specifies a DNDV variable node.

**kNodeTypeCodeDPDU**

Specifies a DPDU variable node.

**kNodeTypeCodeDPDV**

Specifies a DPDV variable node.

**kNodeTypeCodeDU**

Specifies a DU variable node.

**kNodeTypeCodeDV**

Specifies a DV variable node.

**kNodeTypeCodeEDGEBLEND**

Specifies an Edge Blend math node.

**kNodeTypeCodeFASTSCATTER**

Specifies a FastScatter special lighting node.

**kNodeTypeCodeFBM**

Specifies an FBM 3D texture node.

**kNodeTypeCodeFRACTALSUM**

Specifies a Fractal Sum 3D texture node.

**kNodeTypeCodeFRAME**

Specifies a Frame Number variable node.

**kNodeTypeCodeFRESNEL**

Specifies a Fresnel raytrace lighting node.

**kNodeTypeCodeGATHER**

Specifies a Gather raytrace lighting node.

**kNodeTypeCodeGLOSSY**

Specifies a Glossy specular lighting node.

**kNodeTypeCodeGRANITE**

Specifies a Granite 3D texture node.

**kNodeTypeCodeHAIR**

Specifies a Hair special lighting node.

**kNodeTypeCodeHSV**

Specifies an HSV mode User Defined color node.

**kNodeTypeCodeIMAGEMAP**

Specifies an Image Map 2D texture node.

**kNodeTypeCodeLIGHT**

Specifies a Root Light shader node.

**kNodeTypeCodeMARBLE**

Specifies a Marble 3D texture node.

**kNodeTypeCodeMATH**

Specifies a Math Function math node.

**kNodeTypeCodeMOVIE**

Specifies a Movie 2D texture node.

**kNodeTypeCodeN**

Specifies an N variable node.

**kNodeTypeCodeNOISE**

Specifies a Noise 3D texture node.

**kNodeTypeCodeP**

Specifies a P variable node.

**kNodeTypeCodePHONG**

Specifies a Phong specular lighting node.

**kNodeTypeCodePOSERSURFACE**

Specifies the standard Poser surface root node.

**kNodeTypeCodePOSTERIZE**

**kNodeTypeCodePROBELIGHT**

Specifies a ProbeLight diffuse lighting node.

**kNodeTypeCodeREFLECT**

Specifies a Reflect raytrace lighting node.

**kNodeTypeCodeREFRACT**

Specifies a Refract raytrace lighting node.

**kNodeTypeCodeSIMPLECOLOR**

Specifies a Simple Color math node.

**kNodeTypeCodeSKIN**

Specifies a Skin special lighting node.

**kNodeTypeCodeSPECULAR**

Specifies a standard Specular lighting node.

**kNodeTypeCodeSPHEREMAP**

Specifies a Sphere Map environment map lighting node.

**kNodeTypeCodeSPOTS**

Specifies a Spots 3D texture node.

**kNodeTypeCodeTILE**

Specifies a Tile 2D texture node.

**`kNodeTypeCodeTOON`**

Specifies a Toon diffuse lighting node.

**`kNodeTypeCodeTURBULENCE`**

Specifies a Turbulence 3D texture node.

**`kNodeTypeCodeU`**

Specifies a U Texture Coordinate variable node.

**`kNodeTypeCodeUSERDEFINED`**

Specifies a User Defined custom color math node.

**`kNodeTypeCodeV`**

Specifies a V Texture Coordinate variable node.

**`kNodeTypeCodeVELVET`**

Specifies a Velvet special lighting node.

**`kNodeTypeCodeWAVE2D`**

Specifies a Wave2D 2D texture node.

**`kNodeTypeCodeWAVE3D`**

Specifies a Wave3D 3D texture node.

**`kNodeTypeCodeWEAVE`**

Specifies a Weave 2D texture node.

**`kNodeTypeCodeWOOD`**

Specifies a Wood 3D texture node.

## Shader Node Input Codes

Shader Node Input codes define the types of inputs a node can process.

**`kNodeInputCodeBOOLEAN`**

A Boolean node input takes a binary True/False or On/Off value.

**`kNodeInputCodeCOLOR`**

A Color node input takes a 4 digit RGBA color value.

**`kNodeInputCodeFLOAT`**

A Float node input takes 1 floating-point parameter.

**`kNodeInputCodeINTEGER`**

An Integer node input takes 1 integer parameter.

**`kNodeInputCodeMENU`**

A Menu node input takes 1 item from a list of strings.

**kNodeInputCodeNONE**

This code indicates there are no input types available.

**kNodeInputCodeSTRING**

A String node input takes 1 string parameter.

**kNodeInputCodeVECTOR**

A Vector node input takes 3 floating-point parameters.

## Callback Codes

A callback is a user-defined function called by the Poser code. In the following example, the callback is `eventCallbackFunc`. Users write this with their intended functionality then pass it back to Poser to call at the appropriate time. The constants can be used in the callback function to detect the events that occurred. For example, to test a passed back event type to see if a new actor was selected, do the following:

First define a callback function:

```
def eventCallbackFunc(iScene,
iEventType):
if(iEventType & poser.
kEventCodeACTORSELECTIONCHANGED):
print “A new actor was selected.”
```

Now set this function to be the event callback for the scene:

```
scene = poser.Scene()
scene.SetEventCallback(eventCallback
Func
```

Now, whenever a new actor is selected, the python output window will display a message to that effect.

**kCBFrameChanged**

not used

**kCBSceneChanged**

not used

**kCValueChanged**

not used

**kEventCodeACTORADDED**

Check to see if an actor has been added.

**kEventCodeACTORDELETED**

Check to see if an actor has been deleted.

**kEventCodeACTORSELECTIONCHANGED**

Check to see if a different actor has been selected.

**keventCodeANIMSETSCHANGED**

Check to see if the animation set has changed.

**kEventCodeITEMRENAMED**

Check to see if an item has been renamed.

**kEventCodeKEYSCHANGED**

Check to see if keyframes have changed.

**kEventCodePARMADDED**

Check to see if a parm has been added.

**kEeventCodePARMCHANGED**

Check to see if a parm has been changed.

**kEventCodePARMDELETED**

Check to see if a parm has been deleted.

**kEventCodeSETUPMODE**

Check to see if Poser has been placed in Setup Room mode.

## Methods

This section contains the list of custom PoserPython methods. Each method is listed in a separate table, and each table is laid out as follows:

**Method Name:**

The exact name of the method.

**Explanation:**

What the method does.

**Arguments:**

This is a list of possible arguments valid for the listed method.

**Syntax:**

Displays the method's syntax in the format Return, Method, Arguments, for example: <return value type> Method (<type of argument> argument 1, <type of argument> argument 2). Arguments enclosed in curly braces { … } are optional and may default to a value if not specified by caller. Default values are shown following the equals sign.

**Example:**

Some method listings contain an example of how that method might be used.

NOTE: Please note that file paths differ between Mac and Windows operating systems. A Mac path might appear as MyHar dDisk:SomeFolder:Poser:Runtime:Python: poserScripts:myscript.py, whereas a Windows path might look like C:\Some Folder\Poser\Runtime\Python\poserScripts\myscript.py. This is reflected in the different platform-specific versions of Python, and it is similarly reflected here. Please refer to one of the Python resources listed above for further information. Furthermore, PoserPython allows the user to refer to files relative to the Poser folder, for example: Runtime:Python:poserScripts: myscript.py and Runtime\Python\poser Scripts\ myscript.py, respectively.

# General Methods

## AppLocation

**Explanation**

Query the file path location of the Poser application.

**Arguments**

None

**Syntax**

```
<StringType> AppLocation()
```

## AppVersion

**Explanation**

Query the version of the Poser application.

**Arguments**

None

**Syntax**

```
<StringType> AppVersion()
```

## ClearCommands

**Explanation**

Clear the command stack.

**Arguments**

None

**Syntax**

```
<NoneType> ClearCommand()
```

## CloseDocument

**Explanation**

Close the current Poser document.  When set to a value other than zero, the argument causes the method to discard any changes.

**Arguments**

Discard changes = 0 by default.

**Syntax**

```
<NoneType> CloseDocument({<IntType> discardChanges = 0})
```

## CommandNames

**Explanation**

Returns a list of command names. The first name in the list is the oldest command in the stack.

**Arguments**

None

**Syntax**

```
<List of StrType> CommandNames()
```

## ContentRootLocation

**Explanation**

Query the file path location of the main Poser Runtime.

**Arguments**

None

**Syntax**

```
<StringType> ContentRootLocation()
```

## CurrentCommand

**Explanation**

Returns the current command index.

**Arguments**

None

**Syntax**

```
<IntType> CurrentCommand()
```

## DefineMaterialWacroButton

**Explanation**

Attach a python script to one of 10 user defineable material wacro buttons. This method is related to Wacro Setup.

**Arguments**

This method requires 3 Arguments:

- Index: Index specifies to which button the script will be assigned, from 1 to 10, with 1 being the top button.
- File Path: The file name and path of the script that the button will access.
- Label: How you wish to label the button.

**Syntax**

```
<NoneType> DefineMaterialWacroButton(<IntType> buttonIndex, <StringType> filePath,
<StringType> label)
```

## DefineProjGuideHTMLWidget

**Explanation**

Specify which HTML document will appear in the Project Guide

**Arguments**

Enter the palette title you wish to display while the HTML file is showing, and the file name and path of the HTML file.

**Syntax**

```
<NoneType> DefineProjGuideHTMLWidget(<StringType> title, <StringType> filePath)
```

## DefineProjGuideScriptButton

**Explanation**

Attach a python script to one of the two Project Guide directional buttons.

**Arguments**

Enter the button index for the button you wish to assign, and the file name and path of the script that the button will access.

**Syntax**

```
<NoneType> DefineProjGuideScriptButton(<IntType> buttonIndex, <StringType>
filePath)
```

## DefineScriptButton

**Explanation**

Attach a python script to one of the 10 buttons on the Python Scripts palette.

**Arguments**

This method requires 3 Arguments:

- Button Number: From 1 to 10, the button to which the script will be assigned, with 1 being the top button.
- File Path: The complete path to the desired script.
- Label: How you wish to label the button.

**Syntax**

```
<NoneType> DefineScriptButton(<IntType> buttonIndex, <StringType> filePath,
<StringType> label)
```

**Example**

```
poser.DefineScriptButton (1, “C:\Program Files\Curious Labs\Poser 4\Runtime\
Python\test.py”, “Test Script”)
```

## ExecFile

**Explanation**

Run a Python script using a Mac or Windows pathname.

**Arguments**

Enter the complete path of the script you wish to execute.

**Syntax**

```
<NoneType> ExecFile(<StringType> fileName)
```

**Example**

```
poser.ExecFile ("My Macintosh:Curious Labs:Poser 4:Runtime: Python:test.py")
```

## IsPro

**Explanation**

Return whether the Poser executable is the Pro version. Returns 1 for Pro, 0 otherwise.

**Arguments**

None.

**Syntax**

```
<IntType> Version()
```

## Language

**Explanation**

Query the application's language.  The integer returned will match one of the language codes explained above.

**Arguments**

None

**Syntax**

```
<IntType> Language()
```

## Libraries

**Explanation**

Query the file paths of the Libraries. Returns an array of the Library paths

**Arguments**

None

**Syntax**

```
<StringType> Libraries()
```

## NewDocument

**Explanation**

Open the default Poser document

**Arguments**

None

**Syntax**

```
<NoneType> NewDocument()
```

## NewGeometry

**Explanation**

Create an empty geometry object that is not attached to the scene. Use the actor.SetGeometry() method to attach geometry to an actor once it is built.

**Arguments**

None

**Syntax**

```
<GeomType> NewGeometry()
```

## NumRenderThreads

**Explanation**

Get the number of rendering threads.

**Arguments**

None

**Syntax**

```
<IntType> NumRenderThreads()
```

## OpenDocument

**Explanation**

Open an existing Poser document (.pz3 file). Takes a file path (absolute or relative to the Poser folder) as the argument.

**Arguments**

Enter either the complete or relative path to the Poser document you wish to open.

**Syntax**

```
<NoneType> OpenDocument(<StringType> filePath)
```

**Example**

```
poser.OpenDocument("My Macintosh:Runtime:Scenes: myscene.pz3")
```

## PaletteById

**Explanation**

Returns a specific palette identified by the Poser palette constant (such as kCmdCodeANIMATIONPALETTE).

**Arguments**

Enter the Poser palette identification constant.

**Syntax**

```
<PaletteType> PaletteById(<IntType> constant)
```

## Palettes

**Explanation**

Returns a list of accessible palettes.

**Arguments**

None

**Syntax**

```
<ListType> Palettes()
```

## PrefsLocation :

**Explanation**

Get the Poser Prefs dir

**Arguments**

None

**Syntax**

```
<String> PrefsLocation()
```

## Redo

**Explanation**

Redoes one action.

**Arguments**

None

**Syntax**

```
<NoneType> Redo()
```

## Quit

**Explanation**

Quit the Poser Application.

**Arguments**

None

**Syntax**

```
<NoneType> Quit()
```

## RenderInSeparateProcess

**Explanation**

Query whether FireFly is rendering in a separate process.

**Arguments**

None

**Syntax**

```
<IntType> RenderInSeparateProcess()
```

## RevertDocument

**Explanation**

Revert the current document to the last saved state.

**Arguments**

None

**Syntax**

```
<NoneType> RevertDocument()
```

## Rooms

**Explanation**

Displays a list of the rooms accessible within Poser. You could then iterate through this list, comparing against a Poser Room constant (such as kCmdCodeHAIRROOM), until you found the matching room, and access the specific room from the list in this way.

**Arguments**

None

**Syntax**

```
<ListType> Rooms()
```

## SaveDocument

**Explanation**

Save an existing Poser document.  Takes an optional file name as argument.

**Arguments**

Enter a valid path to save the file to that location/filename.  If you do not specify an argument, the file will be saved in its current path with its current name.

**Syntax**

```
<NoneType> SaveDocument({<StringType> filePath})
```

**Example**

```
poser.SaveDocument("C:\My Documents\Poser Stuff\myscene2.pz3")
```

## Scene

**Explanation**

Return the current Poser scene as an object.

**Arguments**

None

**Syntax**

```
<SceneType> Scene()
```

## ScriptLocation

**Explanation**

Query the file path location of the current Python script.

**Arguments**

None

**Syntax**

```
<StringType> ScriptLocation()
```

## SetNumRenderThreads

**Explanation**

Set the number of rendering threads

**Arguments**

Enter the number of rendering threads you wish to use.

### Syntax

```
<NoneType> SetNumRenderThreads(<IntType> numThreads)
```

## SetRenderInSeparate Process

### Explanation

Set whether FireFly renders in a separate process. A value of 1 enables rendering in a separate process; a value of 0 disables rendering in a separate process.

### Arguments

Enter a value of 1 or 0.

### Syntax

```
<NoneType> SetRenderInSeparateProcess(<IntType> separateProcess)
```

## SetWriteBinaryMorphs :

### Explanation

Set if Poser writes morph targest as binary files.

### Arguments

None

### Syntax

```
<NoneType> SetWriteBinaryMorphs(<IntType>binaryMorphs)
```

## StringResource

**Explanation**

Return the string resource for the given major and minor ID.

**Arguments**

Enter the major and minor ID.

**Syntax**

```
<StringType> StringResource(<IntType> majorID, <IntType> minorID)
```

## TempLocation :

**Explanation**

Get the Poser temp dir

**Arguments**

None

**Syntax**

```
<String> TempLocation()
```

## Undo

**Explanation**

Undoes one action.

**Arguments**

None

**Syntax**

```
<NoneType> Undo()
```

## Version

**Explanation**

Return the version number for Poser.

**Arguments**

None

**Syntax**

```
<StringType> Version()
```

## WriteBinaryMorphs :

**Explanation**

Get if Poser writes morph targest as binary files.

**Arguments**

None

**Syntax**

```
<IntType> WriteBinaryMorphs()
```

### WxApp :

**Explanation**

Get the wxApp object of the Poser application.

**Arguments**

None

**Syntax**

```
<class 'wx._core.App'> wxApp()
```

### WxAuiManager :

**Explanation**

Get the wxAuiManager of the Poser UI.

**Arguments**

None

**Syntax**

```
<> wxAuiManager()
```

## Scene Methods

### Actor

**Explanation**

Find a scene actor by its external name.  This is the name seen in Poser GUI pull-down menus (such as “Left Forearm”).

### Arguments

Enter the desired actor’s external name.

### Syntax

```
<ActorType> Actor(<StringType> actorName)
```

### Example

```
actor = scene.Actor(“Left Forearm”)
```

## ActorByInternalName

### Explanation

Find a scene actor by its internal name. The argument string is the unique identifier for the object kept internally by Poser.

### Arguments

Enter the actor’s internal name.

### Syntax

```
<ActorType> ActorByInternalName(<StringType> internalName)
```

### Example

```
actor = scene.ActorByInternalName(“lRing2”)
```

## Actors

### Explanation

Get a list of the non-figure actor objects in the scene. Actors are items that populate the scene such as props, cameras, lights, or deformers. They can also be body-parts of a figure, although body-part actors will not be returned in this list. To get a list of actors belonging to a figure, use the Actors() method for a figure object.

**Arguments**

None

**Syntax**

```
<ActorType List> Actors()
```

## AnimSet

**Explanation**

Return the specified animation set.

**Arguments**

Enter a valid animation set name.

**Syntax**

```
<AnimSetType> AnimSet(<StringType> AnimSetName)
```

**Example**

```
someAnimSet = scene.AnimSet("MyNewAnimationSet")
```

## AnimSets

**Explanation**

Return a list of all animation sets within the scene

**Arguments**

None

**Syntax**

```
<AnimSetType list> AnimSets()
```

## AntialiasNow

**Explanation**

Draw the current display with anti-aliasing enabled.

**Arguments**

None

**Syntax**

```
<NoneType> AntialiasNow()
```

## AtmosphereShaderTree

**Explanation**

Returns the ShaderTree for the atmosphere.

**Arguments**

None

**Syntax**

```
<ShaderTreeType> AtmosphereShaderTree()
```

## BackgroundColor

**Explanation**

Return the RGB color in the range 0.0 to 1.0 that is being used for the background display.

**Arguments**

None

**Syntax**

```
(<FloatType> R, <FloatType> G, <FloatType> B) BackgroundColor()
```

## BackgroundImage

**Explanation**

Returns the name of the current background image, if any.

**Arguments**

None

**Syntax**

```
<StringType> BackgroundImage()
```

## BackgroundMovie

**Explanation**

Returns the name of the current background movie, if any.

**Arguments**

None

**Syntax**

```
<StringType> BackgroundMovie()
```

## BackgroundShaderTree

**Explanation**

Returns the ShaderTree for the scene's background.

**Arguments**

None

**Syntax**

```
<ShaderTreeType> BackgroundShaderTree()
```

## Cameras

**Explanation**

Return a list of scene cameras.  Note that cameras are a specific kind of actor.

**Arguments**

None

**Syntax**

```
<ActorType List> Cameras()
```

## ClearEventCallback

**Explanation**

Clear the per-event callback set with SetEventCallback()

**Arguments**

None

**Syntax**

```
<NoneType> ClearEventCallback
```

## ClearSound

**Explanation**

Specifies that no sound file is to be associated with this Poser document.

**Arguments**

None

**Syntax**

```
<NoneType> ClearSound()
```

## ClearStartupScript

**Explanation**

Specify that no Python script is to be associated with the current Poser document and un-assign the currently associated startup script.

**Arguments**

None

**Syntax**

```
<NoneType> StartupScript()
```

## ClearWorldspaceCallback

**Explanation**

Clear the per-update callback to process scene elements after the entire scene has been processed to world space.

**Arguments**

None

**Syntax**

```
<NoneType> ClearWorldspaceCallback()
```

## ClothSimulator

**Explanation**

Returns the ClothSimulator with the specified index.

**Arguments**

Specify the index of the desired ClothSimulator.

**Syntax**

```
<ClothSimulatorType> ClothSimulator(<IntType> Index)
```

## ClothSimulatorByName

**Explanation**

Find a ClothSimulator object by its name.

**Arguments**

Specify the name of the ClothSimulator you wish to locate.

**Syntax**

```
<ClothSimulatorType> ClothSimulatorByName(<StringType> name)
```

## CopyToClipboard

**Explanation**

Copy the current display to the clipboard.

**Arguments**

None

**Syntax**

```
<NoneType> CopyToClipboard()
```

## CreateAnimSet

**Explanation**

Create a new animation set with the selected name.  Note that if entering a name of an already existing animation set will cause an exception error.

**Arguments**

Enter your desired animation set name, ensuring there is not already an existing animation set with the same name.

**Syntax**

```
<AnimSetType> CreateAnimSet(<StringType> AnimSetName)
```

**Example**

```
newAnimSet = scene.CreateAnimSet("MyNewAnimationSet")
```

## CreateClothSimulator

**Explanation**

Create a new ClothSimulator object.

**Arguments**

Specify the name of the ClothSimulator object.

**Syntax**

```
<ClothSimulatorType> CreateClothSimulator(<StringType> name)
```

## CreateGeomFromGroup

**Explanation**

Generate a new geometry object from a polygon group.

**Arguments**

Enter a valid group name from which the polygons will be obtained.

**Syntax**

```
<GeomType> CreateGeomFromGroup(<ActorType> actor, <StringType> groupName)
```

**Example**

```
geom = scene.CreateGeomFromGroup(“Abdomen”)
```

## CreateLight

**Explanation**

Create a new spotlight in the scene.

### Arguments

None

### Syntax

```
<NoneType> CreateLight()
```

## CreateMagnet

### Explanation

Create a magnet on the current actor.

### Arguments

None

### Syntax

```
<NoneType> CreateMagnet()
```

## CreatePropFromGeom

### Explanation

Create a new scene prop from a geometry.

### Arguments

This method requires 2 Arguments:

- Geometry: This object can be obtained from existing actor geometry, or it can be built from scratch starting with an empty geometry object.  (See poser.NewGeometry()).
- Prop Name: A string naming the new prop.

**Syntax**

```
<ActorType> CreatePropFromGeom(<GeomType> geometry, <StringType> propName)
```

**Example**

```
newProp = scene.CreatePropFromGeom(someActor.Geometry(), "ImaProp")
```

## CreateWave

**Explanation**

Create a wave deformer on the current actor.

**Arguments**

None

**Syntax**

```
<NoneType> CreateWave()
```

## CurrentActor

**Explanation**

Get the currently selected actor.

**Arguments**

None

**Syntax**

```
<ActorType> CurrentActor()
```

## CurrentCamera

**Explanation**

Get the current camera.  Note that cameras are a specific kind of actor.

**Arguments**

None

**Syntax**

```
<ActorType> CurrentCamera()
```

## CurrentFigure

**Explanation**

Get the currently selected figure.

**Arguments**

None

**Syntax**

```
<FigureType> CurrentFigure()
```

## CurrentFireFlyOptions

**Explanation**

Returns the current FireFly options.

**Arguments**

None

**Syntax**

```
<FireFlyOptionsType> CurrentFireFlyOptions()
```

## CurrentLight

**Explanation**

Get the current light.  Note that lights are a specific kind of actor.

**Arguments**

None

**Syntax**

```
<ActorType> CurrentLight()
```

## CurrentMaterial

**Explanation**

Returns the currently selected material.  Returns None if no material is selected.

**Arguments**

None

**Syntax**

```
<MaterialType> CurrentMaterial()
```

## CurrentRenderEngine

**Explanation**

Get the current render engine.

**Arguments**

None

**Syntax**

```
<IntType> CurrentRenderEngine()
```

## DeleteAnimSet

**Explanation**

Delete the specified animation set.

**Arguments**

Enter your desired animation set name.

**Syntax**

```
<NoneType> DeleteAnimSet(<StringType> AnimSetName)
```

**Example**

```
scene.DeleteAnimSet(“MyNewAnimationSet”)
```

## DeleteCurrentFigure

**Explanation**

Delete the currently selected figure.

**Arguments**

None

**Syntax**

```
<NoneType> DeleteCurrentFigure()
```

## DeleteCurrentProp

**Explanation**

Delete the currently selected prop.

**Arguments**

None

**Syntax**

```
<NoneType> DeleteCurrentProp()
```

## DisplayStyle

**Explanation**

Get the document’s interactive display style. Typical return values correspond to poser member variable constants (such as poser.kDisplayCodeWIREFRAME).

**Arguments**

None

**Syntax**

```
<IntType> DisplayStyle()
```

## Draw

**Explanation**

Redraw modified objects.

**Arguments**

None

**Syntax**

```
<NoneType> Draw()
```

## DrawAll

**Explanation**

Redraw everything in the scene.

**Arguments**

None

**Syntax**

```
<NoneType> DrawAll()
```

## Figure

**Explanation**

Get a figure, given its name.  The argument is the external name in the Poser GUI pull-down menus (such as “Figure 1”).

**Arguments**

Enter the figure’s name.

**Syntax**

```
<FigureType> Figure(<StringType> figureName)
```

**Example**

```
fig = scene.Figure("Figure 1")
```

## Figures

**Explanation**

Get a list of the figure objects in the scene.  Figures are bodies composed of actors in a hierarchy.

**Arguments**

None

**Syntax**

```
<FigureType list> Figures()
```

## FireFlyOptions

**Explanation**

Returns the FireFly options with the specified index.

**Arguments**

Specify the index of the desired FireFly options.

**Syntax**

```
<FireFlyOptionsType> FireFlyOptions(<IntType> index)
```

## FireFlyOptionsByName

**Explanation**

Finds FireFly options using a specified name.

**Arguments**

Specify the name of the desired FireFly options.

**Syntax**

```
<FireFlyOptionsType> FireFlyOptionsByName(<StringType> name)
```

## ForegroundColor

**Explanation**

Return the foreground RGB color in the range 0.0 to 1.0

**Arguments**

None

**Syntax**

```
(<FloatType> R, <FloatType> G, <FloatType> B) ForegroundColor()
```

## Frame

**Explanation**

Return the current frame number. All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

None

**Syntax**

```
<IntType> Frame()
```

## FramesPerSecond

**Explanation**

Return the current frame rate.

**Arguments**

None

**Syntax**

```
<IntType> FramesPerSecond()
```

## GeomFileName

**Explanation**

Returns the filename of the geometry being used by the current actor, if any.

**Arguments**

None

**Syntax**

```
<StringType> actor.GeomFileName()
```

## GroundColor

**Explanation**

Return the ground RGB color in the range 0.0 to 1.0.

**Arguments**

None

**Syntax**

```
(<FloatType> R, <FloatType> G, <FloatType> B) GroundColor()
```

## GroundShadows

**Explanation**

Get status of ground shadow display.

**Arguments**

None

**Syntax**

```
<NoneType> GroundShadows()
```

## ImExporter

**Explanation**

Get the importer/exporter object to access importing and exporting of non-Poser 3D file formats.

**Arguments**

None

### Syntax

```
<ImExporterType> ImExporter()
```

## Lights

### Explanation

Return a list of scene lights.  Note that lights are a specific kind of actor.

### Arguments

None

### Syntax

```
<ActorType List> Lights()
```

## LoadLibraryCamera

### Explanation

Load camera positions from a camera library file (.cm2).  The filename should be a path (either absolute or relative to the Poser folder).  Libraries are typically stored under Poser/Runtime/libraries.

### Arguments

Enter the complete path and file name.

### Syntax

```
<NoneType> LoadLibraryCamera(<StringType> filePath)
```

### Example

```
scene.LoadLibraryCamera("Runtime\Libraries\ MyCamera.cm2")
```

## LoadLibraryFace

**Explanation**

Load face from a face library file (.fc2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryFace(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryFace("\Runtime\Libraries\MyFace. fc2")
```

## LoadLibraryFigure

**Explanation**

Load a figure from a character library file (.cr2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryFigure(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryFigure("\Runtime\Libraries\ MyFigure.cr2")
```

## LoadLibraryHair

**Explanation**

Load figure hair from a hair library file (.hr2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryHair(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryHair("\Runtime\Libraries\MyHair.hr2")
```

## LoadLibraryHand

**Explanation**

Load hand pose from a hand library file (.hd2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

- Filename: Enter the complete path and file name.
- Left Hand: Entering 0 returns a left hand, any other value returns a right.

**Syntax**

```
<NoneType> LoadLibraryHand(<StringType> filePath, {<IntType> leftHand = 0})
```

**Example**

```
scene.LoadLibraryHand("\Runtime\Libraries\MyHands. hd2", 1)
```

## LoadLibraryLight

**Explanation**

Load light positions from a light library file (.lt2).  The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryLight(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryLight("\Runtime\Libraries\ MyLight.lt2")
```

## LoadLibraryPose

**Explanation**

Load pose from a pose library file (.pz2).  The filename should be a path (either absolute or relative to the Poser folder).  Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryPose(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryPose("\Runtime\Libraries\MyPose. pz2")
```

## LoadLibraryProp

**Explanation**

Load a prop from a prop library file (.pp2). Filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> LoadLibraryProp(<StringType> filePath)
```

**Example**

```
scene.LoadLibraryProp("\Runtime\Libraries\MyProp. pp2")
```

## MorphFiles

**Explanation**

Returns a list of the used morphtarget files.

**Arguments**

None

**Syntax**

```
<StringType list> MorphFiles()
```

## MovieMaker

**Explanation**

Get a MovieMaker object to access animation specifics. All methods needed to output animated movies can be accessed from the returned object.

**Arguments**

None

**Syntax**

```
<MovieMakerType> MovieMaker()
```

## NextKeyFrame

**Explanation**

Returns the frame number of the next key frame for the current actor.

**Arguments**

None

**Syntax**

```
<IntType> NextKeyFrame()
```

## NextKeyFrameAll

**Explanation**

Returns the frame number of the next key frame in the current scene.

**Arguments**

None

**Syntax**

```
<IntType> NextKeyFrameAll()
```

## NumBodyParts

**Explanation**

Return the number of body parts in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumBodyParts()
```

## NumBumpMaps

**Explanation**

Return the number of bump-maps in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumBumpMaps()
```

## NumCameras

**Explanation**

Return the number of cameras in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumCameras()
```

## NumClothSimulators

**Explanation**

Returns the number of ClothSimulators in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumClothSimulators()
```

## NumFigures

**Explanation**

Return the number of figures in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumFigures()
```

## NumFrames

**Explanation**

Return the number of frames of animation.

**Arguments**

None

**Syntax**

```
<IntType> NumFrames()
```

## NumGeometries

**Explanation**

Return the number of geometries in the scene (equal to the number of props [numProps] + plus the number of body parts [numBodyParts]).

**Arguments**

None

**Syntax**

```
<IntType> NumGeometries()
```

## NumImageMaps

**Explanation**

Return the number of image-maps in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumImageMaps()
```

## NumLights

**Explanation**

Return the number of lights in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumLights()
```

## NumProps

**Explanation**

Return the number of props in the scene.

**Arguments**

None

**Syntax**

```
<IntType> NumProps()
```

## OutputRange

**Explanation**

Return a tuple containing the frame range to be used for image and library output. All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

None

**Syntax**

```
(<IntType> x, <IntType> y) OutputRange()
```

## OutputRes

**Explanation**

Return a tuple containing the output image. The resolution consists of a horizontal and a vertical number of pixels.

**Arguments**

None

**Syntax**

```
(<IntType> x, <IntType> y) OutputRes()
```

## PrevKeyFrame

**Explanation**

Return the frame number of the previous key frame for the current actor.

**Arguments**

None

**Syntax**

```
<IntType> PrevKeyFrame()
```

## PrevKeyFrameAll

**Explanation**

Return the frame number of the previous key frame in the scene.

**Arguments**

None

**Syntax**

```
<IntType> PrevKeyFrameAll()
```

## ProcessSomeEvents

**Explanation**

Process the specified number of Poser events.

**Arguments**

Enter the number of events to process (integer value).

**Syntax**

```
<NoneType> ProcessSomeEvents({<IntType> numEvents = <argument>)
```

**Example**

```
ProcessSomeEvents(numEvents = 1)
```

## Render

**Explanation**

Render to the current view.

**Arguments**

None

**Syntax**

```
<NoneType> Render()
```

## RenderDimAutoscale

**Explanation**

Get the current autoscale resolution setting. Choices are: 0 for Exact Size, 1 for Fit to Preview and 2 for Match to Preview.

**Arguments**

None

**Syntax**

```
(<IntType> option) RenderDimAutoscale()
```

## RenderAntiAliased

**Explanation**

Query renderer's use of anti-aliasing.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderAntiAliased()
```

## RenderBumpMaps

**Explanation**

Query the renderer's use of bump maps. A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderBumpMaps()
```

## RenderCastShadows

**Explanation**

Query rendering of shadows. A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<NoneType> RenderCastShadows()
```

## RenderIgnoreShaderTrees

**Explanation**

Query whether the render engine will ignore shader trees.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<NoneType> RenderIgnoreShaderTrees()
```

## RenderOnBGColor

**Explanation**

Query render-on-background-color option.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderOnBGColor()
```

## RenderOnBGPict

**Explanation**

Query render-on-background-picture option.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderOnBGPict()
```

## RenderOnBlack

**Explanation**

Query render-on-black option.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderOnBlack()
```

## RenderOverType

**Explanation**

Query render-over type.  The return values are 0, 1, 2, and 3 for color, black, bg pict  (background picture), and current shader respectively.

**Arguments**

None

**Syntax**

```
<IntType> RenderOverType()
```

## RenderTextureMaps

**Explanation**

Query the renderer’s use of texture maps.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderTextureaps()
```

## RenderToNewWindow

**Explanation**

Query render-to-new-window option.  A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off.

**Arguments**

None

**Syntax**

```
<IntType> RenderToNewWindow()
```

## Resolution

**Explanation**

Get the curent resolution value (DPI).

**Arguments**

None

**Syntax**

```
(<FloatType> res) Resolution()
```

## ResolutionScale

**Explanation**

Get the curent resolution scale. Choices are: 0 for Full, 1 for Half and 2 for Quarter.

**Arguments**

None

**Syntax**

```
(<FloatType> scale) ResolutionScale()
```

## ResolvePendingTextures

**Explanation**

Resolve any texture paths that may not yet have been searched for.  In general Poser will not look for textures unless they are needed.  This method forces textures to be found.

**Arguments**

None

**Syntax**

```
<NoneType> ResolvePendingTextures()
```

## SaveImage

**Explanation**

Write the current view to an image file by specifying a format suffix (such as “jpg”) and an output filename. When writing out jpg the compression can be specified (10=best compression, 100=best quality). For TIFF images the compression type can be specified (such as kTIFF_LZW), otherwise the compression parameter is ignored. Currently supported image format suffixes are “bmp”, “jpg”, “pct”, “png”, and “tif” . Output filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

- Format : Enter the three-character file suffix for your desired image format. Supported formats are BMP, JPG, PCT, PNG, and TIF.
- Filename: Enter the complete path and filename.

**Syntax**

```
<NoneType> SaveImage(<StringType> formatSuffix, <StringType> filePath, <IntType>
compression)
```

**Example**

```
scene.SaveImage (“bmp”, “C:\My Documents\My Pictures\mypic.bmp”)
```

## SaveImage

**Explanation**

Write the current view to an image file by specifying a format suffix (such as “jpg”) and an output filename. Currently supported image format suffixes are “bmp”, “jpg”, “pct”, “png”, and “tif”. Output filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

- Format : Enter the three-character file suffix for your desired image format. Supported formats are BMP, JPG, PCT, PNG, and TIF.
- Filename: Enter the complete path and filename.

**Syntax**

```
<NoneType> SaveJPG(<StringType> formatSuffix, <StringType> filePath)
```

**Example**

```
scene.SaveImage (“bmp”, “C:\My Documents\My Pictures\mypic.bmp”)
```

## SetRenderDimAutoscale

**Explanation**

Set the choice for the autoscale resolution dimensions. Options are: 0 for Exact Size (as given by OutputRes), 1 for Fit to Preview and 2 for Match to Preview.

**Arguments**

Enter an autoscale option.

**Syntax**

```
<NoneType> SetRenderDimAutoscale(<IntType> option)
```

**Example**

```
scene.SetRenderDimAutoscale(1)
```

## SetResolution

**Explanation**

Set the resolution value(DPI). Optionally provide an argument for the units (0 for inches, 1 for cm).

**Arguments**

Set 0 for inches, 1 for cm

#### Syntax

```
<NoneType> SetResolution (<FloatType> scale {, <IntType> unit = 0)
```

#### Example

```
scene.SetResolution(250, 1)
```

## SetResolutionScale

#### Explanation

Set the choice for the resolution scale.

#### Arguments

Options are: 0 for Full, 1 for Half and 2 for Quarter.

#### Syntax

```
<NoneType> SetResolutionScale(<IntType> scale)
```

#### Example

```
scene.SetResolutionScale(1)
```

## SaveLibraryCamera

#### Explanation

Save the current cameras to a camera library file (.cm2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

#### Arguments

Filename: Enter the complete path and filename.

- Multiple frames: Enter 0 for a single frame, any other value for multiple frames.

- Start Frame: Enter the starting frame of the current animation to save.
- End Frame: Enter the ending frame of the current animation to save.

**Syntax**

```
<NoneType> SaveLibraryCamera(<StringType> filePath, {<IntType> multiFrame,
<IntType> startFrame, <IntType> endFrame})
```

**Example**

```
scene.SaveLibraryCamera("Runtime\Libraries\ MyCamera.cm2", 1,25,68)
```

## SaveLibraryFace

**Explanation**

Save the current face as a face library file (.fc2). The Filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Filename: Enter the complete path and filename.

- Multiple frames: Enter 0 for a single frame, any other value for multiple frames.
- Start Frame: Enter the starting frame of the current animation to save.
- End Frame: Enter the ending frame of the current animation to save.

**Syntax**

```
<NoneType> SaveLibraryFace(<StringType> filePath, {<IntType> multiFrame = 0,
<IntType> startFrame = 0, <IntType> endFrame = 0})
```

**Example**

```
scene.SaveLibraryFace("\Runtime\Libraries\MyFace.fc2", 1,25,68)
```

## SaveLibraryFigure

**Explanation**

Save current figure to a character library file (.cr2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryFigure(<StringType> filePath)
```

**Example**

```
scene.SaveLibraryFigure("Runtime:Libraries: MyFigure.cr2")
```

## SaveLibraryHair

**Explanation**

Save figure hair to a hair library file (.hr2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryHair(<StringType> filePath)
```

**Example**

```
scene.SaveLibraryHair("Runtime:Libraries:MyHair. hr2")
```

## SaveLibraryHand

**Explanation**

Save hand pose to a hand library file (.hd2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryHand(<StringType> filePath, {<IntType> multiFrame = 0,
<IntType> startFrame = 0, <IntType> endFrame = 0})
```

**Example**

```
scene.SaveLibraryHand("Runtime:Libraries:MyHair. hd2")
```

## SaveLibraryLight

**Explanation**

Save current lights to a light library file (.lt2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryLight(<StringType> filePath, {<IntType> multiFrame,
<IntType> startFrame, <IntType> endFrame})
```

**Example**

```
scene.SaveLibraryLight("Runtime:Libraries:MyLight. lt2")
```

## SaveLibraryPose

**Explanation**

Save current pose as a pose library file (.pz2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryPose(<StringType> filePath, {<IntType> includeMorphTargets,
<IntType> multiFrame, <IntType> startFrame, <IntType> endFrame})
```

**Example**

```
scene.SaveLibraryPose(“Runtime:Libraries:MyPose. pz2”)
```

## SaveLibraryProp

**Explanation**

Save current prop as a prop library file (.pp2). The filename should be a path (either absolute or relative to the Poser folder). Libraries are typically stored under Poser/Runtime/libraries.

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SaveLibraryProp(<StringType> filePath)
```

**Example**

```
scene.SaveLibraryProp(“Runtime:Libraries:MyProp. pp2”)
```

## SceneBox

**Explanation**

Get the Bounding Box of the scene in inches.

**Arguments**

None

**Syntax**

```
<BoundingBoxTuppel> SceneBox()
```

## SelectActor

**Explanation**

Set the current actor (i.e. Select an actor).

**Arguments**

Enter a valid Poser actor object.

**Syntax**

```
<NoneType> SelectActor(<ActorType> actor)
```

**Example**

```
scene.SelectActor(scene.Actor("GROUND"))
```

## SelectFigure

**Explanation**

Set the current figure (i.e. Select a figure).

### Arguments

Enter a valid Poser figure object.

### Syntax

```
<NoneType> SelectFigure(<FigureType> figure)
```

### Example

```
scene.SelectFigure(scene.Figure(“JamesCasual”))
```

## SelectMaterial

### Explanation

Select the specified  material in the Material Room.

### Arguments

Enter the material you wish to select.

### Syntax

```
<NoneType> SelectMaterial(<MaterialType> material)
```

## SetBackgroundColor

### Explanation

Set the background RGB color using values in the range 0.0 to 1.0)

### Arguments

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
<NoneType> SetBackgroundColor(<FloatType> R, <FloatType> G, <FloatType> B)
```

**Example**

```
scene.SetBackgroundColor(0.4,0.5,0.6)
```

## SetBackgroundImage

**Explanation**

Set the background image to the specified file. The filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SetBackgroundImage(<StringType> filePath)
```

**Example**

```
scene.SetBackgroundImage("D:\Images\MyImage.jpg")
```

## SetBackgroundMovie

**Explanation**

Set background movie to show behind scene. The filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

Enter the complete file name and path.

**Syntax**

```
<NoneType> SetBackgroundMovie(<StringType> movieName)
```

**Example**

```
scene.SetBackgroundImage("D:\Movies\MyMovie.avi")
```

## SetCurrentCamera

**Explanation**

Set the current camera.  Note that cameras are a specific kind of actor.

**Arguments**

Enter a valid Poser camera object.

**Syntax**

```
<NoneType> SetCurrentCamera(<ActorType> camera)
```

**Example**

```
SetCurrentCamera(leftCamera)
```

## SetCurrentLight

**Explanation**

Set the current light.  Note that lights are a specific kind of actor.

**Arguments**

Enter a valid Poser light actor.

**Syntax**

```
<NoneType> SetCurrentLight(<ActorType> light)
```

**Example**

```
scene.SetCurrentLight(spotLight)
```

## SetCurrentRenderEngine

**Explanation**

Set the current render engine.

**Arguments**

Specify the desired render engine.

**Syntax**

```
<NoneType> SetCurrentRenderEngine(<IntType> Engine)
```

## SetDisplayStyle

**Explanation**

Set interactive display style of the document.  Typical values are constants defined as poser member variables (such as poser.
kDisplayCodeWIREFRAME).

**Arguments**

Enter a valid display code.

**Syntax**

```
<NoneType> SetDisplayStyle(<IntType> displayCode)
```

**Example**

```
scene.SetDisplayStyle(poser.kDisplayCodeSMOOTH LINED)
```

## SetEventCallback

**Explanation**

Set a per-event callback function that will be called for every Poser event.  The callback function passed in should take two

Arguments: A scene object and an eventCode.  Bit wise, the eventCode can be compared to known eventCode constants to detect the type of events occurring.

### Arguments

Enter a valid scene object and a valid eventCode.

### Syntax

```
<NoneType> SetEventCallback (<FunctionType> newCD, {<Object> cbArgs})
```

### Example

Click the Sample Callbacks button in the Python palette to see an example using this method.

## SetForegroundColor

### Explanation

Set the foreground RGB color using values in the range 0.0 to 1.0)

### Arguments

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

### Syntax

```
<NoneType> SetForegroundColor(<FloatType> R, <FloatType> G, <FloatType> B)
```

### Example

```
scene.SetForegroundColor(0.4,0.5,0.6)
```

## SetFrame

**Explanation**

Set the current frame number. All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<NoneType> SetFrame(<IntType> frame)
```

**Example**

```
scene.SetFrame(23)
```

## SetGroundColor

**Explanation**

Set the ground RGB color using values in the range 0.0 to 1.0)

**Arguments**

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
<NoneType> SetGroundColor(<FloatType> R, <FloatType> G, <FloatType> B)
```

**Example**

```
scene.SetGroundColor(0.4,0.5,0.6)
```

## SetGroundShadows

**Explanation**

Toggle display of ground shadows.  The default argument of 1 specifies that the option should be turned on.  To turn it off, call the function with an argument of 0.

**Arguments**

Enter 0 to disable ground shadows, or 1 to enable them.

**Syntax**

```
<NoneType> SetGroundShadows({<IntType> on = 1})
```

**Example**

```
scene.SetGroundShadows(1)
```

## SetMeAsStartupScript

**Explanation**

Specify the current script as the Python script associated with the current Poser doc and executed on startup when the document is re-opened.

**Arguments**

None

**Syntax**

```
<NoneType> SetMeAsStartupScript()
```

## SetOutputRange

**Explanation**

Specify the output frame range to be used for image and library output (for images). All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

- Start Frame (X): Enter a numeric value that is less than or equal to the end frame value.
- End Frame (Y): Enter a numeric value that is greater than or equal to the start frame value.

**Syntax**

```
<NoneType> SetOutputRange(<IntType> x, <IntType> y)
```

**Example**

```
scene.SetOutputRange(25,67)
```

## SetOutputRes

**Explanation**

Set output resolution (for images). Resolution consists of a horizontal and a vertical number of pixels.

**Arguments**

Enter a dimension in pixels using the format x,y.

**Syntax**

```
<NoneType> SetOutputRes(<IntType> x, <IntType> y)
```

**Example**

```
scene.SetOutput Res(640,480)
```

## SetRenderAntiAliased

**Explanation**

Toggle renderer anti-aliasing.  The default argument of 1 specifies that the option should be turned on.  To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable anti-aliasing, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderAntiAliased({<IntType> on = 1})
```

**Example**

```
scene.SetRenderAntiAliased(0)
```

## SetRenderBumpMaps

**Explanation**

Toggle renderer use of bump maps.  The default argument of 1 specifies that the option should be turned on.  To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable bump map use, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderBumpMaps({<IntType> on = 1})
```

**Example**

```
scene.SetRenderBumpMaps(1)
```

## SetRenderCastShadows

**Explanation**

Toggle rendering of shadows.  The default argument of 1 specifies that the option should be turned on.  To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable cast shadow rendering, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderCastShadows({<IntType> on = 1})
```

**Example**

```
scene.SetRenderCastShadows(1)
```

## SetRenderIgnoreShaderTrees

**Explanation**

Toggle whether or not the render engine will ignore shader trees.  The default argument of 1 specifies that the option should be turned on.  To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable ignoring of shader trees, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderIgnoreShaderTrees({<IntType> on = 1})
```

## SetRenderOnBGColor

**Explanation**

Set the renderer to render over background color. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable rendering over the background color, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderOnBGColor({<IntType> on = 1})
```

**Example**

```
scene.SetRenderOnBGColor(1)
```

## SetRenderOnBGPict

**Explanation**

Set the renderer to render over background picture. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0.

**Arguments**

<NoneType> RenderOnBGPict({<IntType> on = 1})

**Syntax**

```
Enter 1 to enable rendering over the current background picture, or 0 to disable
it.
```

**Example**

```
scene.RenderOnBGPicture(0)
```

## SetRenderOnBlack

**Explanation**

Set the renderer to render over black. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable rendering against a black background, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderOnBlack({<IntType> on = 1})
```

**Example**

```
scene.SetRenderOnBlack(1)
```

## SetRenderOverType

**Explanation**

Set the renderer to render over color, black, background picture, or the current shader tree. Type values are 0, 1, 2, 3 for color, black, bg pict (background picture), and current shader respectively.

**Arguments**

Enter the type value for the render-over type you wish to specify.

**Syntax**

```
<NoneType> SetRenderOverType({<IntType> type})
```

## SetRenderTextureMaps

**Explanation**

Toggle the renderer’s use of texture maps. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable bump map use, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderTextureMaps({<IntType> on = 1})
```

**Example**

```
scene.SetRender (1)
```

## SetRenderToNewWindow

**Explanation**

Toggle render-to-new-window option. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to render to a new window, or 0 to disable it.

**Syntax**

```
<NoneType> SetRenderToNewWindow({<IntType> on = 1})
```

**Example**

```
scene.SetRenderToNewWindow(0)
```

## SetShadowColor

**Explanation**

Set the shadow RGB color using values in the range 0.0 to 1.0)

**Arguments**

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
<NoneType> SetShadowColor(<FloatType> R, <FloatType> G, <FloatType> B)
```

**Example**

```
scene.SetShadowColor(1.0,1.0,0.3)
```

## SetSound

**Explanation**

Specify the sound file to be associated with this Poser document.  Sound files play during animation.

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> SetSound(<StringType> filePath)
```

**Example**

```
scene.SetSound("C:\My Music\Sound1.wav")
```

## SetSoundRange

**Explanation**

Specify the frame range over which the sound should be played
during animation.

**Arguments**

Enter valid starting and ending frames for the sound.

**Syntax**

```
<NoneType> SetSoundRange(<IntType> startFrame, <IntType> endFrame)
```

**Example**

```
scene.SetSoundRange(5,12)
```

## SetStartupScript

**Explanation**

Specify the Python script to associate with the current Poser document and executed on startup when the file is re-opened.  The filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> SetStartupScript(<StringType> filePath)
```

**Example**

```
scene.SetStartupScript("\Runtime\Python\script.py")
```

## SetWorldspaceCallback

**Explanation**

Set a per-update callback to process scene elements after the entire scene has been processed to world space.

**Arguments**

The callback function should take the scene as an argument and make any scene changes desired. The changes will occur after all objects have placed themselves in world space, but before the final drawing takes place.

**Syntax**

```
<NoneType> SetWorldspaceCallback(<FunctionType> newCB, {<Object> cbArgs})
```

**Example**

```
(See sample scripts)
```

## ShadowColor

**Explanation**

Return the shadow RGB color using values in the range 0.0 to 1.0)

**Arguments**

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
(<FloatType> R, <FloatType> G, <FloatType> B) ShadowColor()
```

**Example**

```
scene.ShadowColor(1.0,1.0,0.3)
```

## Sound

**Explanation**

Return the name of the sound file associated with the current Poser document that plays during animations.

**Arguments**

None

**Syntax**

```
<StringType> Sound()
```

## SoundRange

**Explanation**

Return the frame range over which the sound is played during animation.  Returns a tuple containing the start frame and the end frame.

**Arguments**

None

**Syntax**

```
(<IntType>, <IntType>) SoundRange()
```

## StartupScript

**Explanation**

Return the Python script to be associated with the current Poser document and executed on startup when the document is reopened. The returned filename is a path (either absolute or relative to the Poser folder).

**Arguments**

None

**Syntax**

```
<StringType> StartupScript()
```

## UpdateBGPreview

**Explanation**

Updates the preview's background.  Call this function after you modify the background shader.

**Arguments**

None

**Syntax**

```
<NoneType> UpdateBGPreview()
```

## WacroLights

**Explanation**

Returns a list of light actors on which a script is to be executed.  The script can then iterate over this list in order to apply light modifications to all lights.

**Arguments**

None

**Syntax**

```
<ActorType list> WacroLights()
```

## WacroMaterials

**Explanation**

Returns a list of materials on which a wacro is to be executed.  This method is intended for use inside wacros; they should iterate over this list.

**Arguments**

None

**Syntax**

```
<MaterialType list> GetWacroMaterials()
```

## WorldToScreen

**Explanation**

Takes a set of (x, y, z) world coordinates (the location of a point within the 3D scene) and returns (x, y, z) screen coordinates (the location of that point relative to the screen).

**Arguments**

Enter the x, y, z coordinates of the point for which you wish the screen coordinates.

**Syntax**

```
(<FloatType> x, <FloatType> y, <FloatType> z), WorldToScreen(<FloatType> x,
<FloatType> y, <FloatType> z)
```

## MovieMaker Methods

### Antialias

**Explanation**

Query the antialias settings. A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off

**Arguments**

None

**Syntax**

```
<IntType> Antialias()
```

### FlashAutoPlay

**Explanation**

Query the Flash auto-play option.  Returns 1 if the option is enabled, 0 if disabled.

**Arguments**

None

**Syntax**

```
<IntType> FlashAutoPlay()
```

## FlashDrawInnerLines

**Explanation**

Query the Draw Inner Lines option for Flash export.  A return value of 1 means that the option is on, while a 0 means that the option is off.

**Arguments**

None

**Syntax**

```
<IntType> FlashDrawInnerLines()
```

## FlashDrawOuterLines

**Explanation**

Query the Draw Outer Lines option for Flash export.  A return value of 1 means that the option is on, while a 0 means that the option is off.

**Arguments**

None

**Syntax**

```
<IntType> FlashDrawInnerLines()
```

## FlashLineWidth

**Explanation**

Get the width of drawn lines for Flash export.  Note that both inner and outer lines use the same line width.

**Arguments**

None

**Syntax**

```
<FloatType> FlashLineWidth()
```

## FlashNumColors

**Explanation**

Get the number of colors to be used for Flash export.

**Arguments**

None

**Syntax**

```
<IntType> FlashNumColors()
```

## FlashOverlapColors

**Explanation**

Query the Overlapping Colors option for Flash export.  A return value of 1 means that the option is on, while a 0 means that the option is off.

**Arguments**

None

**Syntax**

```
<IntType> FlashOverlapColors()
```

## FlashQuantizeAll

**Explanation**

Query the Quantize All Frames option for exporting Flash. A return value of 1 means that the option is on, while a 0 means that the option is off. Note that this return value will always be the negation of moviemaker.FlashQuantizeOne.

**Arguments**

None

**Syntax**

```
<IntType> FlashQuantizeAll()
```

## FlashQuantizeFrame

**Explanation**

Get the frame to be quantized when exporting Flash with the quantize-one-frame option on.

**Arguments**

None

**Syntax**

```
<IntType> FlashQuantizeFrame()
```

## FlashQuantizeOne

**Explanation**

Query the Quantize Specified Frame option for exporting Flash. A return value of 1 means that the option is on, while a 0 means that the option is off. Note that this return value will always be the negation of moviemaker.FlashQuantizeAll.

**Arguments**

None

**Syntax**

```
<IntType> FlashQuantizeOne()
```

## FrameOptions

**Explanation**

Return the values for frame rate and increment.

**Arguments**

None

**Syntax**

```
(<IntType> rate, <IntType> increment) FrameOptions()
```

## MakeFlash

**Explanation**

Write the animation to a Flash file (*.swf).

**Arguments**

Enter the complete file name and path for the output file.

**Syntax**

```
<NoneType> MakeMovieFlash(<StringType> filePath)
```

**Example**

```
mm.MakeFlash("C:\MyDocuments\myflashmovie.swf")
```

## MakeMovie

**Explanation**

Write out the animation to file(s). Filepath can be relative to the Poser application or absolute. For image files also provide the file format as 3-letter-string additional argument (eg “png”), When writing out jpg the compression can be specified (10=best compression, 100=best quality), otherwise the compression parameter is  ignored. Use in conjunction with the SetMovieFormat method to define the output type.

**Arguments**

Enter the complete file name and path for the output file.

**Syntax**

```
<NoneType> MakeMovie(<StringType> filePath {, <StringType> fileFormat, <IntType>
compression})
```

**Example**

```
moviemaker.SetMovieFormat(2)
moviemaker.MakeMovie(“TestMovie”, “jpg”, 90)  or
moviemaker.SetMovieFormat(3)
moviemaker.MakeMovie(“C:\\TestMovie.swf”)
```

## MotionBlur

**Explanation**

Query the motion blur settings. A return value of 1 indicates that the option is on, while a value of 0 indicates that it is off. The second parameter is the blur amount

**Arguments**

None

**Syntax**

```
(<IntType> on, <FloatType> amount) MotionBlur()
```

## MovieFormat

**Explanation**

Return the current movie format setting. The codes are: 1 for AVI on Windows or QT on Mac, 2 for Image files, 3 for Flash.

**Arguments**

None

**Syntax**

```
<IntType> MovieFormat()
```

## MovieRenderer

**Explanation**

Return the current movie renderer setting.

**Arguments**

None

**Syntax**

```
<IntType> MovieRenderer()
```

## OutputEndFrame

**Explanation**

Return the last frame to be used in making the movie. All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

None

**Syntax**

```
<IntType> OutputEndFrame()
```

## OutputRes

**Explanation**

Return a tuple containing output resolution (for movies).

**Arguments**

None

**Syntax**

```
(<IntType> x, <IntType> y) OutputRes
```

## OutputStartFrame

**Explanation**

Return the first frame to be used in making the movie. All frame numbers in PoserPython are relative to a starting frame of 0. For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

None

**Syntax**

```
<IntType> OutputStartFrame()
```

## SetAntialias

**Explanation**

Toggle the antialias value. The default argument of 1 specifies that the option should be turned on. To turn it off, call the function with an argument of 0

**Arguments**

Enter 0 to disable antialiasing, or 1 to enable it.

**Syntax**

```
<NoneType> SetAntialias({<IntType> on = 1})
```

## SetFlashAutoPlay

**Explanation**

Set the Auto Play option for Flash Export.

**Arguments**

Enter 1 to enable the option, or 0 to disable it.

**Syntax**

```
<NoneType> SetFlashAutoPlay({<IntType> on})
```

#### Example

```
mm.SetFlashAutoPlay(1)
```

## SetFlashDrawInnerLines

#### Explanation

Toggle drawing of inner lines for Flash export. The default argument of 0 specifies that the overlapping-colors option is off. To turn it on, call the function with an argument of 1.

#### Arguments

Enter 1 to enable drawing inner lines, or 0 to disable.

#### Syntax

```
<NoneType> SetFlashDrawInnerLines({<IntType> on = 0})
```

#### Example

```
mm.SetFlashDrawInnerLines(1)
```

## SetFlashDrawOuterLines

#### Explanation

Toggle drawing of outer lines for Flash export. The default argument of 1 specifies that the overlapping-colors option is on. To turn it off, call the function with an argument of 0.

#### Arguments

Enter 1 to enable drawing outer lines, or 0 to disable.

#### Syntax

```
<NoneType> SetFlashDrawOuterLines({<IntType> on = 1})
```

**Example**

```
mm.SetFlashDrawOuterLines(1)
```

## SetFlashLineWidth

**Explanation**

Set the width of drawn lines for Flash export.  Note that both inner and outer lines use the same line width.

**Arguments**

Enter any valid floating-point number.

**Syntax**

```
<NoneType> SetFlashLineWidth({<FloatType> width = 1.0})
```

**Example**

```
mm.SetFlashLineWidth(2.43)
```

## SetFlashNumColors

**Explanation**

Set the number of colors to be used for Flash export.

**Arguments**

Enter the number of colors to use.

**Syntax**

```
<NoneType> SetFlashNumColors({<IntType> numColors = 4})
```

**Example**

```
mm.SetFlashNumColors(6)
```

## SetFlashOverlapColors

**Explanation**

Toggle overlapping colors for Flash export. The default argument of 1 specifies that the overlapping-colors option is on. To turn it off, call the function with an argument of 0.

**Arguments**

Enter 1 to enable overlapping colors, or 0 to disable.

**Syntax**

```
<NoneType> SetFlashOverlapColors({<IntType> on = 1})
```

**Example**

```
mm.SetFlashOverlapColors(1)
```

## SetFlashQuantizeAll

**Explanation**

Quantize all frames when exporting flash.

**Arguments**

None

**Syntax**

```
<NoneType> SetFlashQuantizeAll()
```

## SetFlashQuantizeFrame

**Explanation**

Specify the frame to be quantized when exporting Flash with the quantize-one-frame option on.

**Arguments**

Enter the number of the selected frame.

**Syntax**

```
<NoneType> SetFlashQuantizeFrame({<IntType> frame})
```

**Example**

```
mm.SetFlashQuantizeFrame(4)
```

## SetFlashQuantizeOne

**Explanation**

Quantize a specified frame when exporting Flash. If the frame argument is supplied, the quantize frame will be set to it. Otherwise, the existing value will be used.

**Arguments**

Enter the desired frame number.

**Syntax**

```
<NoneType> SetFlashQuantizeOne({<IntType> frame})
```

**Example**

```
mm.SetFlashQuantizeOne(12)
```

## SetFrameOptions

**Explanation**

Set the values for frame rate and increment.

### Arguments

Enter two integer values, for frame rate and frame increment respectively

### Syntax

```
<NoneType> SetFrameOptions(<IntType> rate, <IntType> increment)
```

### Example

```
moviemaker.SetFrameOptions(24, 4)
```

## SetMotionBlur

### Explanation

Set the values for motion blur settings. Default is ON (1), with a value of 0.5. To turn it off, call the function with a single argument of 0

### Arguments

Enter the desired motion blur setting and optionally value.

### Syntax

```
<NoneType> MotionBlur(<IntType> on = 1, {<FloatType> amount})
```

### Example

```
moviemaker.MotionBlur(1, 0.75)
```

## SetMovieFormat

### Explanation

Set the movie format. The codes are: 1 for AVI on Windows or QT on Mac, 2 for Image files, 3 for Flash.

#### Arguments

Enter the desired movie-making output format.

#### Syntax

```
<NoneType> SetMovieFormat(<IntType> Format)
```

## SetMovieRenderer

#### Explanation

Set the movie renderer, use the same codes as the scene render engine.

#### Arguments

Enter the desired movie renderer.

#### Syntax

```
<NoneType> SetMovieRenderer(<IntType> Renderer)
```

## SetOutputEndFrame

#### Explanation

Set the last frame to be used in making the movie.  All frame numbers in PoserPython are relative to a starting frame of 0.  For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

#### Arguments

Enter the number of the ending frame.

#### Syntax

```
<NoneType> SetOutputEndFrame(<IntType> frame)
```

**Example**

```
mm.SetOutputEndFrame(60)
```

## SetOutputRes

**Explanation**

Set output resolution (for movies).

**Arguments**

Enter the X and Y resolution in pixels.

**Syntax**

```
<NoneType> SetOutputRes(<IntType> x, <IntType> y)
```

**Example**

```
mm.SetOutputRes(640,640)
```

## SetOutputStartFrame

**Explanation**

Set the first frame to be used in making the movie.  All frame numbers in PoserPython are relative to a starting frame of 0.  For this reason, a frame number in Python is 1 less than the equivalent frame as referenced from the Poser GUI.

**Arguments**

Enter the number of the starting frame.

**Syntax**

```
<NoneType> SetOutputStartFrame(<IntType> frame)
```

### Example

```
mm.SetOutputStartFrame(1)
```

## Importer/Exporter Methods

## Export

### Explanation

Export models using plugins. The file suffix argument is the extension typically following files of the type to be exported, such as "dxf". The actual plugin name may be given (e.g. "File Format HAnim") to specify which plugin to choose if there are several plugins that export the same file type. If only one plugin exists that exports files with the given extension, then this argument may be None. The filePath string (which can be an absolute path, or a path relative to the Poser folder) specifies the file to be exported. The default options-dictionary can be acquired by a call to the imexporter.ExportOptions() method with the same fileSuffix as an argument. It can then be modified and passed back to as an arguement to this method. If this argument is omitted, the default options will be used. The optional scene hierarchy callback function allows for specification of object inclusion in the export process. The function should take an actor object and return 1 if the actor is to be included and 0 if the actor is to be excluded. The function will be called back for all actors in the scene. If this argument is omitted, all visible objects will be exported.

### Arguments

File Suffix:

- Viewpoint: vtx
- Biovision (BVH Motion): bvh
- 3D Studio Max: 3ds
- QuickDraw 3DMF: 3df
- AutoCAD: dxf

- Detailer Text: vtx
- Wavefront OBJ: OBJ

Plug-in Names: Poser needs plug-ins to support some export formats. If a valid export format does not appear here, that format is supported directly within the Poser application itself. The plug-in name can typically be set to None. However, if two plug-ins exist which export files ending in the same suffix, then you can use the plug-in name to distinguish between the two.

- Viewpoint: File Format MetaStream
- 3D Studio Max: File Format 3D Studio
- QuickDraw 3DMF: File Format 3DMF
- AutoCAD: File Format DXF
- Wavefront OBJ: File Format Wavefront

File Path: Enter a valid path and filename. The path can be either the complete path or relative to the Poser folder.

Option Dictionary: Enter any non-standard options (optional). If not supplied, the default options will apply.

Function: Call a function if desired (optional). If not supplied, the default items will be exported.

**Syntax**

```
<NoneType> Export(<StringType> fileSuffix, <StringType> pluginName, <StringType>
filePath, {<DictType> options, <FunctionType> sceneSelectionCallback})
```

**Example**

```
Imex.Export("DXF", "File Format DXF", "C:\My Documents\Test.dxf")
```

## ExportOptions

**Explanation**

Get a dictionary of options for the specified exporter. The file suffix argument is the extension typically following files of the type to be exported, such as "dxf". The actual plug-in name may be given (e.g. "File Format HAnim") to specify which plug-in to choose if there are several plug-ins that export the same file type.  If only one plug-in exists that exports files with the given extension, then this argument may be None.

**Arguments**

Enter a valid export file suffix and plug-in name.

**Syntax**

```
<DictType> ExportOptions(<StringType> fileSuffix, <StringType> pluginName)
```

**Example**

```
imex.ExportOptions("obj", None)
```

## ExportOptionString

**Explanation**

Get an export option string for the specified file suffix and plugin name.  The enumeration value is a key from the export options dictionary.

**Arguments**

Enter a valid export file suffix and plug-in name.

**Syntax**

```
<StringType> ExportOptionString(<StringType> fileSuffix, <StringType> pluginName,
<IntType> enumValue)
```

**Example**

```
imex.ExportOptionString("obj", None, poser.kExOptCodeMULTIFRAME)
```

## Import

### Explanation

Import models using plug-ins. The file suffix argument is the extension typically following files of the type to be exported, such as "dxf". The filePath string (which can be an absolute path, or a path relative to the Poser folder) specifies the file to be imported. The default options-dictionary can be acquired by a call to the imexporter.ImportOptions() method with the same fileSuffix as an argument. It can then be modified and passed back to as an argument to this method. If this argument is omitted, the default options will be used.

### Arguments

File Suffix:

- Viewpoint: mtx
- Biovision (BVH Motion): bvh
- 3D Studio Max: 3ds
- QuickDraw 3DMF: 3df
- AutoCAD: dxf
- Detailer Text: vtx
- Wavefront OBJ: OBJ

File Path: Enter a valid path and filename. The path can be either the complete path or relative to the Poser folder.

Option Dictionary: Enter any non-standard options. This is an optional argument. Default options used otherwise.

### Syntax

```
<NoneType> Import(<StringType> fileSuffix, <StringType> filePath, {<DictType>
options})
```

### Example

```
Import("DXF", "C:\My Documents\test.dxf")
```

## ImportOptions

**Explanation**

Get a dictionary of options for the specified importer. The file suffix argument is the extension typically following files of the type to be exported, such as “dxf”. The actual plug-in name may be given (e.g. “File Format HAnim”) to specify which plug-in to choose if there are several plug-ins that import the same file type. If only one plug-in exists that imports files with the given extension, then this argument may be None.

**Arguments**

Enter a valid import file suffix and plug-in name.

**Syntax**

```
<DictType> ImportOption(<StringType> fileSuffix, <StringType> pluginName)
```

**Example**

```
imex.ImportOptions(“OBJ”, none)
```

## ImportOptionString

**Explanation**

Get an import option string for the specified file suffix and plug-in name. The enumeration value is a key from the import options dictionary.

**Arguments**

Enter a valid import file suffix and plug-in name.

**Syntax**

```
<StringType> ImportOptionString(<StringType> fileSuffix, <StringType> pluginName,
<IntType> enumValue)
```

### Example

```
imex.ImportOptionString("OBJ", None, poser.kImOptCodePERCENTFIGSIZE)
```

## Animation Set Methods

### AddAttribute

#### Explanation

Adds a new attribute to the current animation set.

#### Arguments

- Attribute Name: Enter the name of the attribute you wish to add.
- Value: Enter the desired value of the attribute.

#### Syntax

```
<NoneType> AddAttribute(<StringType> name, <StringType> value)
```

#### Example

```
animset.AddAttribute("MyAttribute",1)
```

### AddObjectRange

#### Explanation

Add an object range to the animation set.  The entity provided must be a figure, actor, or parameter.

#### Arguments

- Object: Enter the name of a valid figure, actor, or parameter.

- Start Frame: Enter the number of the starting frame you wish to include (Python frames begin with 0). This number should be less than the end frame number.
- End Frame: Enter the number of the last frame you wish to include (Python frames begin with 0). This number should be greater than the start frame number.

**Syntax**

```
<NoneType> AddObjectRange (<FigureType, Actor Type, or ParmType>, sceneEntity,
<IntType> StartFrame, <IntType> EndFrame)
```

**Example**

```
animset.AddObjectRange(someActor,5,20)
```

## Attributes

**Explanation**

Get a list of all attributes in the current animation set. Attributes are tuples consisting of the name of animation set and the corresponding value strong.

**Arguments**

None

**Syntax**

```
<TupleType list> Attributes()
```

## ObjectRange

**Explanation**

Get the object range for the specified animation set.

### Arguments

None

### Syntax

```
(<IntType> startFrame, <IntType> endFrame) ObjectRange()
```

## RemoveAttribute

### Explanation

Remove an existing attribute from the current animation set.

### Arguments

- Attribute Name: Enter the name of the attribute you wish to add.
- Value: Enter the desired value of the attribute.

### Syntax

```
<NoneType> RemoveAttribute(<StringType> name, {<StringType> value})
```

### Example

```
animset.RemoveAttribute("MyAttribute", 1)
```

## RemoveObjectRange

### Explanation

Remove an existing object range from the current animation set.

### Arguments

- Object: Enter the name of a valid figure, actor, or parameter.
- Start Frame: Enter the number of the starting frame you wish to include (Python frames begin with 0). This number should be less

than the end frame number.

- End Frame: Enter the number of the last frame you wish to include (Python frames begin with 0). This number should be greater than the start frame number.

**Syntax**

```
<NoneType> RemoveObjectRange (<FigureType, ActorType, or ParmType>, sceneEntity,
<IntType> StartFrame, <IntType> EndFrame)
```

**Example**

```
animset.RemoveObjectRange(someActor,5,20)
```

## Actor Methods

### AddKeyFrame

**Explanation**

Add a key frame for this parameter at the specified frame. If no frame is specified, a keyframe will be added at the current frame.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<NoneType> AddKeyFrame({<IntType> frame})
```

**Example**

```
AddKeyFrame(81)
```

## AlignmentRotationXYZ

**Explanation**

Get a tuple comprising the ordered rotation alignment for this actor. (order is X, Y, Z)

**Arguments**

None

**Syntax**

```
(<FloatType>, <FloatType>, <FloatType>) AlignmentRotationXYZ()
```

## AltGeomFileName

**Explanation**

Get the name of the alternate geometry file used by this actor (if specified).

**Arguments**

None

**Syntax**

```
<StringType> AltGeomFileName()
```

## AmbientOcclusion

**Explanation**

Query whether this light (if this actor is an image light) is using ambient occlusion.

**Arguments**

None

**Syntax**

```
<IntType> AmbientOcclusion()
```

## AmbientOcclusionBias

**Explanation**

Get the ambient occlusion bias of this light (if this actor is an image light).

**Arguments**

None

**Syntax**

```
<FloatType> AmbientOcclusionBias()
```

## AmbientOcclusionDistance

**Explanation**

Get the ambient occlusion maximum distance of this light (if this actor is an image light).

**Arguments**

None

**Syntax**

```
<FloatType> AmbientOcclusionDistance()
```

## AmbientOcclusionStrength

**Explanation**

Get the ambient occlusion strength of this light (if this actor is an image light).

**Arguments**

None

**Syntax**

```
<FloatType> AmbientOcclusionStrength()
```

## AtmosphereStrength

**Explanation**

Get the atmosphere strength of this light (if this actor is a light).

**Arguments**

None

**Syntax**

```
<FloatType> AtmosphereStrength()
```

## BackfaceCull

**Explanation**

Query the actor's backface culling flag.

**Arguments**

None

**Syntax**

```
<IntType> BackfaceCull()
```

## Base

**Explanation**

If the actor is a deformer, this method returns its base actor.

**Arguments**

None

**Syntax**

```
<ActorType> ActorBase()
```

## Bends

**Explanation**

Query whether or not the actor’s bend flag is set.

**Arguments**

None

**Syntax**

```
<IntType> Bends()
```

## CastShadows

**Explanation**

Query whether this actor casts shadows.

**Arguments**

None

**Syntax**

```
<IntType> CastShadows()
```

## Children

**Explanation**

Get a list of the actors that are the children of the actor given.

**Arguments**

None

**Syntax**

```
<ActorType List> Children()
```

## ClearLocalTransformCallback

**Explanation**

Clear the local transform callback.

**Arguments**

None

**Syntax**

```
<NoneType> ClearLocalTransformCallback()
```

## ClearVertexCallback

**Explanation**

Clear the vertex callback.

**Arguments**

None

**Syntax**

```
<NoneType> ClearVertexCallback()
```

## CreaseAngle

**Explanation**

Get the actor’s crease angle.

**Arguments**

None

**Syntax**

```
<FloatType> CreaseAngle()
```

## CreateHairGroup

**Explanation**

Create a new hair group.

**Arguments**

Specify the name of the hair group you wish to create.

**Syntax**

```
<HairType> CreateHairGroup(<StringType> name)
```

## CreateValueParameter

**Explanation**

Create a value parameter on the universe actor. This type of parameter is not linked to Poser elements such as figures, props, etc. Rather, it can be used to add user interface control to your Python scripts.

**Arguments**

Enter a name for the new parameter.

**Syntax**

```
<ParmType> CreateValueParameter(<StringType> valueParmName)
```

**Example**

```
parm = actor.CreateValueParameter("MyNewParameter")
```

## Delete

**Explanation**

Delete the actor from the scene if possible. Note that you cannot delete a body part from a figure.

**Arguments**

None

**Syntax**

```
<NoneType> Delete()
```

## DeleteKeyFrame

**Explanation**

Delete a key frame for this actor at the specified frame.  If no frame is specified, a keyframe will be deleted at the current frame.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<NoneType> DeleteKeyFrame({<IntType> frame})
```

**Example**

```
parm.DeleteKeyFrame(30)
```

## DisplacementBounds

**Explanation**

Get the actor’s displacement bounds.

**Arguments**

None

**Syntax**

```
<FloatType> DisplacementBounds()
```

## DisplayStyle

**Explanation**

Get the interactive display style for this actor. Typical return values correspond to poser member variable constants (such as poser.kDisplayCodeWIREFRAME).

**Arguments**

Enter a valid display code.

**Syntax**

```
<NoneType> SetDisplayStyle(<IntType> displayCode)
```

**Example**

```
actor.SetDisplayStyle(poser.kDisplayCode SMOOTHSHADED)
```

## DropToFloor

**Explanation**

Drop the actor downward (along the Y axis) until it touches the floor (Y==0).

**Arguments**

None

**Syntax**

```
<NoneType> DropToFloor()
```

## EndPoint

**Explanation**

Get the position of the current actor's endpoint. The endpoint is typically also the origin of an object's child. It's also a specified endpoint used for on-screen interactions and potentially for IK relationships. It also typically ends a line along the first rotation (twist) axis.

**Arguments**

None

**Syntax**

```
(<FloatType> x, <FloatType> y, <FloatType> z) EndPoint()
```

## GeomFileName

**Explanation**

Returns the filename of the geometry bring used by the figure, if any.

**Arguments**

None

**Syntax**

```
<StringType> figure.GeomFileName()
```

## Geometry

**Explanation**

Get the geometry for the actor. The returned geometry object can then be queried for vertex, set, or polygon information.

**Arguments**

None

**Syntax**

```
<GeomType> Geometry()
```

## HairGroup

**Explanation**

Get the hair group specified by the index.

**Arguments**

Enter the index of the desired hair group.

**Syntax**

```
<HairType> HairGroup(<IntType> index)
```

## InternalName

**Explanation**

Get the (internal) name for the actor.  The specified string is a unique name ID internal to Poser.

**Arguments**

None

**Syntax**

```
<StringType> InternalName()
```

## IsBase

**Explanation**

Return true only if the actor is a base.  Bases are targets of deformation for deformers such as magnets.

**Arguments**

None

**Syntax**

```
<IntType> IsBase()
```

## IsBodyPart

**Explanation**

Return true only if the actor is a body part.

**Arguments**

none

**Syntax**

```
<IntType> IsBodyPart()
```

## IsCamera

**Explanation**

Return true only if the actor is a camera.

**Arguments**

None

**Syntax**

```
<IntType> IsCamera()
```

## IsDeformer

**Explanation**

Return true only if the actor is a deformer.

**Arguments**

None

**Syntax**

```
<IntType> IsDeformer()
```

## IsHairProp

**Explanation**

Return true only if actor is a hair prop.

**Arguments**

None

**Syntax**

```
<IntType> IsHairProp()
```

## IsLight

**Explanation**

Return true only if the actor is a light.

**Arguments**

None

**Syntax**

```
<IntType> IsLight()
```

## IsProp

**Explanation**

Return true only if the actor is a prop.

**Arguments**

None

**Syntax**

```
<IntType> IsProp()
```

## IsZone

**Explanation**

Return true only if the actor is a zone.  Zones are regions acted upon by deformers such as magnets.

**Arguments**

None

**Syntax**

```
<IntType> IsZone()
```

## ItsFigure

**Explanation**

Get the figure of which this actor is a part.  The return value is a figure object.

**Arguments**

None

**Syntax**

```
<FigureType> ItsFigure()
```

## JointVertexWeights

**Explanation**

Get a list of vertex weights for the specified joint axis on this actor.

**Arguments**

The axis argument should be ‘x’, ‘y’, or ‘z’.  If no such joint is present, the method will return None.

**Syntax**

```
<FloatType list> JointVertexWeights(<StringType> axis)
```

**Example**

```
actor.JointVertexWeight(X)
```

## LightAttenType

**Explanation**

Get the falloff type of the light (if this actor is a light). Possible falloff types are poser.kLightCodePOSER, poser.kLightCodeINVLINEARATTEN and poser.kLightCodeINVSQUAREFALLOFF.

**Arguments**

None

**Syntax**

```
<IntType> LightAttenType()
```

## LightOn

**Explanation**

If the current actor is an image light, query whether it is On.

**Arguments**

None

**Syntax**

```
<IntType> LightOn()
```

## LightType

**Explanation**

Get the type of the light (if the current actor is a light).  Possible light types are infinite (0), spot(1), point(2), image(3).

**Arguments**

None

**Syntax**

```
<IntType> LightType()
```

## LoadMaterialCollection

**Explanation**

Load a material collection for this actor.

**Arguments**

Enter the file name of the material collection you wish to load.

**Syntax**

```
<NoneType> LoadMaterialCollection(<StringType> FileName)
```

## LocalDisplacement

**Explanation**

Get a tuple comprising the local displacement for this actor.

**Arguments**

None

**Syntax**

```
(<FloatType> tx, <FloatType> ty, <FloatType> tz ) LocalDisplacement()
```

## LocalMatrix

**Explanation**

Get the local matrix for the actor.  The local matrix is the matrix that describes the model's relative relationship to its parent in the hierarchy.  Thus the final world matrix of an object is made by concatenating its parent's world matrix with its local matrix to produce its world matrix.

**Arguments**

None

**Syntax**

```
<FloatType 4x4 Tuple> LocalMatrix()
```

## LocalQuaternion

**Explanation**

Get a tuple comprising the quaternion local rotation for this actor.

**Arguments**

None

**Syntax**

```
(<FloatType> qw, <FloatType> qx, <FloatType> qy, <FloatType> qz )
LocalQuaternion()
```

## MarkGeomChanged

**Explanation**

Sets and internal flag on actor noting that the geometry has been changed.  This method should be called after geometry changes so they will be stored properly.

**Arguments**

None

**Syntax**

```
<NoneType> MarkGeomChanged()
```

## Material

**Explanation**

Get a material object by its name. The string argument should typically be a name displayed in the Poser GUI material pull-down menu (e.g. "skin"). The call searches all materials available to this actor.

**Arguments**

Enter a valid material name.

**Syntax**

```
<MaterialType> FindMaterialByName(<StringType> name)
```

**Example**

```
skinMat = actor.FindMaterialByName("skin")
```

## Materials

**Explanation**

Get a list of the materials available to the actor. Note that these materials may actually belong to a parent figure.

**Arguments**

None

**Syntax**

```
<MaterialType List> Materials()
```

**Example**

```
matsList  = actor.Materials()
```

## Memorize

**Explanation**

Set the actor's default parameter values to the current values so that the actor can be reset later (See actor.Reset()).

**Arguments**

None

**Syntax**

```
<NoneType> Memorize()
```

## Name

**Explanation**

Get the (external) name for the current actor.  The specified name is the same one seen in Poser's GUI pull-down menus.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## NextKeyFrame

**Explanation**

Get the next frame in which the parameter has a key frame set.

**Arguments**

None

**Syntax**

```
<IntType> NextKeyFrame()
```

## NumHairGroups

**Explanation**

Returns the number of hair groups.

**Arguments**

None

**Syntax**

```
<IntType> NumHairGroups()
```

## OnOff

**Explanation**

Query the display status of the actor in the scene.  Returns 1 if actor is currently displayed, and 0 if actor is not currently displayed.

**Arguments**

None

**Syntax**

```
<IntType> OnOff()
```

## Orientation

**Explanation**

Get the orientation of the actor's coordinate system (x,y,z ordered rotation).

ArgumenNone

**Syntax**

```
(<FloatType> x, <FloatType> y, <FloatType> z) Orientation()
```

## Origin

**Explanation**

Get the position of the current actor's origin in world space.

**Arguments**

None

**Syntax**

```
(<FloatType> x, <FloatType> y, <FloatType> z) Origin()
```

## Parameter

**Explanation**

Get the named parameter object. The string argument is the external parameter name (e.g. "TranslationX").

**Arguments**

Enter a valid parameter name

**Syntax**

```
<ParmType> Parameter(<StringType> parmName)
```

### Example

```
parm = Parameter("TranslationX")
```

## PrevKeyFrame

### Explanation

Get the previous frame in which the parameter has a key frame set.

### Arguments

None

### Syntax

```
<IntType> PrevKeyFrame()
```

## ParameterByCode

### Explanation

Get the first parameter object matching the given code. Typical code values are constants defined as poser member variables (such as poser.kParmCodeXTRAN).

### Arguments

Enter a valid parameter code.

### Syntax

```
<ParmType> ParameterByCode(<IntType> parmCode)
```

### Example

```
parm = ParameterByCode(poser.kParmCodeXSCALE)
```

## Parameters

**Explanation**

Get the settable parameters for the current actor.  The return value is a list of parameter objects such as those depicted in Poser's GUI dials (e.g. "TranslationX").

**Arguments**

None

**Syntax**

```
<ParmType List> Parameters()
```

## Parent

**Explanation**

Get the parent actor of the current actor.

**Arguments**

None

**Syntax**

```
<ActorType> Parent()
```

## PointAt

**Explanation**

Set the target actor for this actor to point towards.

**Arguments**

Enter the actor that you wish this actor to point at.

### Syntax

```
<NoneType> SetPointAt(<ActorType> target)
```

### Examples:

```
ACTOR.PointAt()
```

Returns the actor where ACTOR is pointing to (return None is nothing).

```
ACTOR.SetPointAt(actor)
```

Sets ACTOR to point at actor.

## RayTraceShadows

### Explanation

Query whether this light (if this actor is a light) is using Raytracing for shadows.

### Arguments

None

### Syntax

```
<IntType> RayTraceShadows()
```

## RemoveValueParameter

### Explanation

Remove a value parameter from an actor.  This type of parameter is not linked to Poser elements such as figures, props, etc.  Rather, it can be used to add user interface control to your Python scripts.

### Arguments

Enter the name of the parameter you wish to delete.

**Syntax**

```
<NoneType> RemoveValueParameter(<StringType> valueParmName)
```

**Example**

```
actor.RemoveValueParameter("MyNewParameter")
```

## Reset

**Explanation**

Reset the actor to its default, or last memorized, values (See actor.Memorize()).

**Arguments**

None

**Syntax**

```
<NoneType> Reset()
```

## SaveMaterialCollection

**Explanation**

Save the material collection of this actor.  Note  that only selected materials will be included.  See the SetSelected and Selected methods in the Material Methods section for information on selecting and querying selection of materials.

**Arguments**

Enter the file name for the material collection.

**Syntax**

```
<NoneType> SaveMaterialCollection(<StringType> FileName)
```

## ScaleMatrix

**Explanation**

Get the scale matrix for the actor.

**Arguments**

None

**Syntax**

```
<FloatType 4x4 Tuple> ScaleMatrix()
```

## SetAlignmentRotationXYZ

**Explanation**

Set the tuple comprising the ordered rotation alignment for this actor.

**Arguments**

Enter valid floating-point values for X, Y, and Z rotation (order is X, Y, and Z).  Angles are in degrees.

**Syntax**

```
<NoneType> SetAlignmentRotationXYZ(<FloatType> Rx, <FloatType> Ry, <FloatType>
Rz )
```

**Example**

```
actor.SetAlignmentRotationXYZ(4.53, 7.61, 1.01)
```

## SetAmbientOcclusion

**Explanation**

Set whether this light (if this actor is an image light) is using ambient occlusion.

**Arguments**

Enter 1 to use ambient occlusion, or 0 to disable ambient occlusion for this light.

**Syntax**

```
<NoneType> SetAmbientOcclusion(<IntType> ambientocclusion)
```

## SetAmbientOcclusionBias

**Explanation**

Set the ambient occlusion bias of this light (if this actor is an image light).

**Arguments**

Enter the desired bias for this light.

**Syntax**

```
<NoneType> SetAmbientOcclusionBias(<FloatType> bias)
```

## SetAmbientOcclusionDistance

**Explanation**

Set the ambient occlusion maximum distance of this light (if this actor is an image light).

**Arguments**

Enter the desired maximum distance for this light.

**Syntax**

```
<NoneType> SetAmbientOcclusionDistance(<FloatType> distance)
```

## SetAmbientOcclusionStrength

**Explanation**

Set the ambient occlusion strength of this light (if this actor is an image light).

**Arguments**

Enter the ambient occlusion strength value.

**Syntax**

```
<NoneType> SetAmbientOcclusionStrength(<FloatType> strength)
```

## SetAtmosphereStrength

**Explanation**

Set the atmosphere strength for this light (if this actor is a light).

**Arguments**

Atmosphere strength value.

**Syntax**

```
<NoneType> SetAtmosphereStrength(<FloatType> atmStrength)
```

**Example**

```
actor.SetAtmosphereStrength(0.15)
```

## SetBackfaceCull

**Explanation**

Set the actor's backface culling flag.

**Arguments**

Enter 1 to activate backface culling during rendering, or 0 to disable backface culling.

**Syntax**

```
<NoneType> SetBackfaceCull(<IntType> on = 1)
```

## SetBends

**Explanation**

Sets the actor's bend flag.

**Arguments**

Enter 1 to set the flag, 0 to disable it.

**Syntax**

```
<NoneType> SetBends({<IntType> bends=<1 or 0>})
```

**Example**

```
SetBends(bends=1)
```

## SetCastShadows

**Explanation**

Set whether this actor casts shadows.

**Arguments**

Enter 1 to cast shadows, or 0 to disable shadows.

**Syntax**

```
<NoneType> SetCastsShadows(<IntType> Cast)
```

## SetCreaseAngle

**Explanation**

Set the actor's crease angle.

**Arguments**

Crease angle in degrees.

**Syntax**

```
<NoneType> SetCreaseAngle(<FloatType> angle)
```

## SetDisplacementBounds

**Explanation**

Set the actor's displacement bounds.

**Arguments**

Displacement bounds value.

**Syntax**

```
<NoneType> SetDisplacementBounds(<FloatType> dispBounds)
```

**Example**

```
actor.SetDisplacementBounds(0.9)
```

## SetDisplayStyle

**Explanation**

Set the display style to be used for this actor. Typical display code constants are defined as poser member variables (e.g. poser.kDisplayCodeWIREFRAME).

**Arguments**

Enter a valid display code.

**Syntax**

```
<NoneType> SetDisplayStyle(<IntType> displayCode)
```

**Example**

```
actor.SetDisplayStyle(poser.kDisplayCodeFLATLINED)
```

## SetEndPoint

**Explanation**

Set the position of the endpoint of the actor. Typically the endpoint is the origin of an object's child. It's also a specified endpoint used for on screen interactions and potentially for IK relationships. It also typically ends a line along the first rotation (twist) axis.)

**Arguments**

Enter valid X, Y, and Z coordinates.

**Syntax**

```
<NoneType> SetEndpoint(<FloatType> x, <FloatType> y, <FloatType> z)
```

**Example**

```
actor.SetEndpoint(5.38, 3.90, 1.23)
```

## SetGeometry

**Explanation**

Set the geometry for the actor. The actor then takes on the appearance of the given geometry. Valid geometry objects can be taken from other actors or created from scratch with poser.NewGeometry().

#### Arguments

Specify a valid Poser geometry object.

#### Syntax

```
<NoneType> SetGeometry(<GeomType> geometry)
```

#### Example

```
actor.SetGeometry(geom)
```

## SetIncludeInBoundingBox

#### Explanation

Set to determine inclusion in scene bounding box calculations.  Default argument is set to 1, specifying that the actor should be included.  Argument should be set to 0 to exclude actor.

#### Arguments

Enter 1 to include the actor, or 0 to exclude it.

#### Syntax

```
<NoneType> SetIncludeInBoundingBox({<IntType> on = 1})
```

#### Example

```
actor.SetIncludeInBoundingBox(1)
```

## SetLightAttenType

#### Explanation

Set the falloff type of the light to a specified value.  Typical values are constant light falloff codes defined as poser member variables such as poser.kLightCodeINVSQUAREFALLOFF.

**Arguments**

Enter a valid light attenuation code.

**Syntax**

```
<NoneType> SetLightAttenType(<IntType> lightAttenTypeCode)
```

## SetLightOn

**Explanation**

If the current actor is an image light, toggle it on or off.  A value of 1 sets the light actor to On; a value of 0 sets it to OFF.

**Arguments**

Enter a value of 1 or 0.

**Syntax**

```
<NoneType> SetLightOn(<IntType> lightOn)
```

## SetLightType

**Explanation**

Set the type of the light to a specified value.  Typical values are constant light codes defined as poser member variables such as poser.kLightCodeINFINITE.

**Arguments**

Enter a valid light code.

**Syntax**

```
<NoneType> SetLightType(<IntType> lightTypeCode)
```

**Example**

```
actor.SetLightType(poser.kLightCodeSPOT)
```

## SetLocalTransformCallback

**Explanation**

Set a per-update callback to process the actor's local transformation. User-defined callback function is called every time actor undergoes a local transform.

**Arguments**

The callback function should take an actor argument and make desired changes to that actor.

**Syntax**

```
<NoneType> SetLocalTransformCallback(<FunctionType> newCB, {<Object> cbArgs})
```

## SetName

**Explanation**

Renames the current actor.

**Arguments**

Enter a valid actor name

**Syntax**

```
<NoneType> actor.setName(<StringType> ActorName)
```

**Example**

```
actor.setName(MyNewActor)
```

## SetOnOff

**Explanation**

Set the display status of the current actor in the scene.  The argument should be set to 1 to display the actor and 0 to turn actor display off.

**Arguments**

Enter 1 to display the actor, or 0 to toggle it off.

**Syntax**

```
<NoneType> SetOnOff(<IntType> on)
```

**Example**

```
actor.SetOnOff(0)
```

## SetOrientation

**Explanation**

Set the orientation of the actor’s coordinate system in x, y, z ordered rotation.

**Arguments**

Enter valid X, Y, and Z coordinates.

**Syntax**

```
<NoneType> SetOrientation(<FloatType> x, <FloatType> y, <FloatType> z)
```

**Example**

```
actor.SetOrientation(1.83, 4.0, 2.47)
```

## SetOrigin

**Explanation**

Set the position of the actor's origin in world space.

**Arguments**

Enter valid X, Y, and Z coordinates.

**Syntax**

```
<NoneType> SetOrigin(<FloatType> x, <FloatType> y, <FloatType> z)
```

**Example**

```
actor.SetOrigin(1.83, 4.0, 2.47)
```

## SetParameter

**Explanation**

Set the current value of the parameter. The string argument is the external parameter name (e.g. "TranslationX").

**Arguments**

- Parameter Name: Enter a valid parameter name.
- Value: Enter a valid value for the selected parameter.

**Syntax**

```
<NoneType> SetParameter(<StringType> parmName, <FloatType> value)
```

**Example**

```
actor.SetParameter(poser.kParmCodeXSCALE,75)
```

## SetParent

### Explanation

Set the specified actor as the parent of the current actor.  If inheritBends is 1, this actor will acquire the bend parameters of the parent.  If Realign is 1, this actor will be realigned to the local space of the parent.

### Arguments

New Parent: The actor that will become the new parent of this actor.

- Inherit Bends: Defaults to 0.  Enter 1 to specify that this actor should inherit the bends from the new parent.
- Realign: Defaults to 0.  Enter 1 to specify that this actor should be realigned to conform to the new parent.

### Syntax

```
<NoneType> SetParent(<ActorType> newParent, {<IntType> inheritBends = 0,
<IntType> realign = 0})
```

### Example

```
childActor.SetParent(ParentActor, 1, 0)
```

## SetRangeConstant

### Explanation

Set the given frame range to have constant (step) interpolation between keyframes  for all parms of this actor.  Note: automatically sets keyframes at start and end of specified range.

### Arguments

Enter valid start and end frame numbers.

### Syntax

```
<NoneType> SetRangeConstant(<IntType> startFrame, <IntType> endFrame)
```

### Example

```
actor.SetRangeConstant(12,40)
```

## SetRangeLinear

### Explanation

Set the given frame range to have linear interpolation between key frames for all parms of this actor. Note: automatically sets key frames at start and end of specified range.

### Arguments

Enter valid start and end frame numbers.

### Syntax

```
<NoneType> SetRangeLinear(<IntType> startFrame, <IntType> endFrame)
```

### Example

```
actor.SetRangeLinear(12,40)
```

## SetRangeSpline

### Explanation

Set the given frame range to have spline interpolation between key frames for all parms of this actor. Note: automatically sets key frames at start and end of specified range

### Arguments

Enter valid start and end frame numbers.

### Syntax

```
<NoneType> SetRangeSpline(<IntType> startFrame, <IntType> endFrame)
```

**Example**

```
actor.SetRangeSpline(12,40)
```

## SetRayTraceShadows

**Explanation**

Set whether this light (if this actor is a light) is using raytracing for shadows.

**Arguments**

Enter 1 to use raytracing, or 0 to disable raytracing for this light.

**Syntax**

```
<NoneType> SetRayTraceShadows(<IntType> on = 1)
```

## SetShadingRate

**Explanation**

Set the actor's minimum shading rate.

**Arguments**

Minimum shading rate value.

**Syntax**

```
<NoneType> SetShadingRate(<FloatType> minShadingRate)
```

**Example**

```
actor.SetShadingRate(0.6)
```

## SetShadow

**Explanation**

Set whether this light (if this actor is a light) is producing shadows.

**Arguments**

Enter 1 to enable shadows, or 0 to disable shadows for this light.

**Syntax**

```
<NoneType> SetShadow(<IntType> shadow)
```

## SetShadowBiasMax

**Explanation**

Set the maximum shadow bias for depth map shadows on this light (if this actor is a light).

**Arguments**

Maximum shadow bias value.

**Syntax**

```
<NoneType> SetShadowBiasMax(<FloatType> maxShadowBias)
```

**Example**

```
actor.SetShadowBiasMax(2.3)
```

## SetShadowBiasMin

**Explanation**

Set the minimum shadow bias for depth map shadows on this light (if this actor is a light).

**Arguments**

Minimum shadow bias value.

**Syntax**

```
<NoneType> SetShadowBiasMin(<FloatType> minShadowBias)
```

**Example**

```
actor.SetShadowBiasMin(0.3)
```

## SetShadowBlurRadius

**Explanation**

Set the shadow map blur radius for this light (if this actor is a light).

**Arguments**

Shadow blur radius value.

**Syntax**

```
<NoneType> SetShadowBlurRadius(<IntType> shadowBlurRad)
```

**Example**

```
actor.SetShadowBlurRadius(5)
```

## SetSmoothPolys

**Explanation**

Set whether polygon smoothing is enabled for this actor.

**Arguments**

Enter 1 to smooth polygons during rendering, or 0 to disable polygon smoothing.

### Syntax

```
<NoneType> SetSmoothPolys(<IntType> Smooth)
```

## SetSplineBreak

### Explanation

Set the given frame range to have spline interpolation between key frames for all parms of this actor.  Note: automatically sets key frames at start and end of specified range

### Arguments

Enter valid frame numbers then enter 1 for on and 0 for off.

### Syntax

```
<NoneType> SetSplineBreak({<IntType> frame, <IntType> on})
```

### Example

```
actor.SetSplineBreak(12,0)
```

## SetStatic

### Explanation

Set the status of static parameters on the actor.  Argument defaults to 1, specifying that the actor's parameters are static and will not change during animation.  To specify non-static parameters, this function should be called with a 0 argument.

### Arguments

Enter 1 to specify static parameters, or 0 to specify non-standard parameters

### Syntax

```
<NoneType> SetStatic(<IntType> on = 1)
```

**Example**

```
actor.SetStatic(0)
```

## SetVertexCallback

**Explanation**

Set a per-update callback to process actor (vertices) while in local deformed space. The user-defined function will be called back on each full update of an actor and allows for manipulation of the vertices (or other information) at the point in the display pipeline of Poser which processes vertex level deformations.

**Arguments**

The callback function should take an actor object and make desired modifications to it.

**Syntax**

```
<NoneType> SetVertexCallback(<FunctionType> newCB, {<Object> cbArgs})
```

**Example**

```
(See sample script)
```

## SetVisibleInReflections

**Explanation**

Set whether this actor is visible in reflections.

**Arguments**

Enter 1 to make actor visible in reflections during rendering, or 0 to make it invisible.

**Syntax**

```
<NoneType> SetVisibleInReflections(<IntType> on = 1)
```

## SetVisibleInRender

**Explanation**

Set whether this actor is visible in renders.

**Arguments**

Enter 1 to make actor visible in the rendered image, or 0 to make it invisible.

**Syntax**

```
<NoneType> SetVisibleInRender(<IntType> on = 1)
```

## ShadingRate

**Explanation**

Get the actor's minimum shading rate.

**Arguments**

None

**Syntax**

```
<FloatType> ShadingRate()
```

## Shadow

**Explanation**

Query whether this light (if this actor is a light) is producing shadows.

**Arguments**

None

**Syntax**

```
<IntType> Shadow()
```

## ShadowBiasMax

**Explanation**

Get the maximum shadow bias for depth map shadows on this light (if this actor is a light).

**Arguments**

None

**Syntax**

```
<FloatType> ShadowBiasMax()
```

## ShadowBiasMin

**Explanation**

Get the minimum shadow bias for depth map shadows on this light (if this actor is a light).

**Arguments**

None

**Syntax**

```
<FloatType> ShadowBiasMin()
```

## ShadowBlurRadius

**Explanation**

Get the shadow map blur radius for this light (if this actor is a light).

**Arguments**

None

**Syntax**

```
<IntType> ShadowBlurRadius()
```

## SmoothPolys

**Explanation**

Query whether polygon smoothing is enabled for this actor.

**Arguments**

None

**Syntax**

```
<IntType> SmoothPolys()
```

## SpawnTarget

**Explanation**

Creates a new morph channel on the object using the current state of the vertices.  Typically used when while an actor is being deformed.

**Arguments**

Enter a name for the new morph channel.

**Syntax**

```
<NoneType> SpawnTarget(<StringType> label
```

**Example**

```
actor.SpawnTarget("MyNewMorphTarget")
```

## SpawnTargetFromGeometry

**Explanation**

Creates a new morph channel on the object using a geometry.

**Arguments**

The geometry argument is the deformation target, and the label provides a name for the new morph channel.

**Syntax**

```
<NoneType> SpawnTargetFromGeometry(<GeomType> geometry, <StringType> label)
```

**Example**

```
actor.SpawnTargetFromGeometry(geom, "MyMorphTarget")
```

## Static

**Explanation**

Get the status of the current actor's static parameters.  A return value of 1 means that the actor has static parameters, while a return value of 0 means that the parameters are not static.

**Arguments**

None

**Syntax**

```
<IntType> Static()
```

## TwistVertexWeights

**Explanation**

Get a list of vertex weights for the specified twist axis on this actor.

**Arguments**

The axis argument should be 'x', 'y', or 'z'.  If no such joint is present, the method will return None.

**Syntax**

```
<FloatType list> TwistVertexWeights(<StringType> axis)
```

**Example**

```
actor.TwistVertexWeight(X)
```

## ValueParameter

**Explanation**

Get a value parameter from the universe actor.  This type of parameter is not linked to Poser elements such as figures, props, etc. Rather, it can be used to add user interface control to your Python scripts.

**Arguments**

Enter a valid parameter name.

**Syntax**

```
<ParmType> ValueParameter(<StringType> valueParmName)
```

#### Example

```
parm = actor.ValueParameter(MyNewParameter)
```

## ValueParameters

#### Explanation

Get a list of value parameters from the universe actor. This type of parameter is not linked to Poser elements such as figures, props, etc. Rather, it can be used to add user interface control to your Python scripts.

#### Arguments

None

#### Syntax

```
<ParmType List> ValueParameters()
```

## VisibleInReflections

#### Explanation

Query whether this actor is visible in reflections.

#### Arguments

None

#### Syntax

```
<IntType> VisibleInReflections()
```

## VisibleInRender

**Explanation**

Query whether this actor is visible in renders.

**Arguments**

None

**Syntax**

```
<IntType> VisibleInRender()
```

## WeldGoalActors

**Explanation**

Get a list of actors that are welded to this one. Weld goal actors share edge vertices with this actor and are used to allow for geometric continuity when limbs bend.

**Arguments**

None

**Syntax**

```
<ActorType List> WeldGoalActors()
```

## WeldGoals

**Explanation**

Get a list of vertex index lists that specify the weld goals for this actor. Each vertex index list corresponds to a weld goal actor with the same index. And each such list is composed of vertex indices into that weld goal actor's geometry. Each index list contains as many indices as the geometry of this actor contains vertices, but list items corresponding to vertices with no weld goals are filled with -1's.

**Arguments**

None

**Syntax**

```
<List of IntType Lists> WeldGoals()
```

## WorldDisplacement

**Explanation**

Get a tuple comprising the World displacement for this actor.

**Arguments**

None

**Syntax**

```
(<FloatType> tx, <FloatType> ty, <FloatType> tz) WorldDisplacement()
```

## WorldMatrix :

**Explanation**

Get the world matrix for the actor. The world matrix is the final complete matrix which transforms geometry from its original model space to its final world space.

**Arguments**

None

**Syntax**

```
<FloatType 4x4 Tuple> WorldMatrix()
```

### WorldQuaternion

**Explanation**

Get a tuple comprising the quaternion world rotation for this actor.

**Arguments**

None

**Syntax**

```
(<FloatType> qw, <FloatType> qx, <FloatType> qy, <FloatType> qz )
WorldQuaternion()
```

### Zones

**Explanation**

If the actor is a zone, this method returns the list of zones.

**Arguments**

None

**Syntax**

```
<ActorType List> ActorZones()
```

## Figure Methods

### Actor

**Explanation**

Get an actor, given its name.  This is the name given in Poser's GUI pull-down menus.

**Arguments**

Enter a valid actor name using the Poser external name.

**Syntax**

```
<ActorType> Actor(<StringType> name)
```

**Example**

```
fig.Actor("LeftForearm")
```

## ActorByInternalName

**Explanation**

Get an actor, given its internal name.  This is a unique identifying string stored internally by Poser.

**Arguments**

Enter a valid internal name.

**Syntax**

```
<ActorType> Actor(<StringType> name)
```

**Example**

```
fig.ActorByInternalName("LeftForearm")
```

## Actors

**Explanation**

Get a list of actor objects comprising this figure.

**Arguments**

None

**Syntax**

```
<ActorType List> Actors()
```

## ClearStartupScript

**Explanation**

Specify that no Python script is to be associated with this figure.

**Arguments**

None

**Syntax**

```
<NoneType> ClearStartupScript()
```

## ConformTarget

**Explanation**

Return the figure whose pose this figure is set to conform to.

**Arguments**

None

**Syntax**

```
<FigureType> ConformTarget()
```

## ConformTo

**Explanation**

Pose a specified figure to match the pose of the currently selected figure. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability. We recommend that you use the SetConformTarget method instead.

**Arguments**

Enter the name of the figure you wish to pose.

**Syntax**

```
<NoneType> ConformTo(<FigureType> conformee)
```

## CreateFullBodyMorph

**Explanation**

Create a new full body morph for the specified figure.

**Arguments**

Enter the name of the figure for which you want to create the full body morph.

**Syntax**

```
<NoneType> CreateFullBodyMorph(<StringType> name)
```

## DisplayStyle

**Explanation**

Get the interactive display style for this figure. Typical return values correspond to poser member variable constants (such as poser.kDisplayCodeWIREFRAME).

### Arguments

None

### Syntax

```
<IntType> DisplayStyle()
```

## DropToFloor

### Explanation

Lower the figure (along the Y-axis) until it just touches the floor (y=0).

### Arguments

None

### Syntax

```
<NoneType> DropToFloor()
```

## FigureMeasure

### Explanation

Measures hip height and feet distance.

### Arguments

None

### Syntax

```
<NoneType> FigureMeasure()
```

## GeomFileName

**Explanation**

Returns the filename of the geometry being used by the current actor, if any.

**Arguments**

None

**Syntax**

```
<StringType> GeomFileName()
```

## IkNames

**Explanation**

Get a list of names of inverse kinematics chains for the figure. The index of each name is the index used to get or set the status of the chain using IkStatus() or SetIkStatus().

**Arguments**

None

**Syntax**

```
<StringType list> IkNames()
```

## IkStatus

**Explanation**

Get the status of inverse kinematics for the specified IK chain index.

**Arguments**

Enter an index between 0 and figure.NumIkChains()

**Syntax**

```
<IntType> IkStatus(<IntType> whichLimb)
```

**Example**

```
leftLegOn = figure.IkStatus(0)
```

## InternalName

**Explanation**

Get the internal name for the figure. This is a unique identifying string stored internally by Poser.

**Arguments**

None

**Syntax**

```
<StringType> InternalName()
```

## Material

**Explanation**

Get a material object by its name. The string argument should typically be a name displayed in the Poser GUI material pull-down menu (e.g. "skin"). The call searches all materials available to this actor.

**Arguments**

Enter a valid material name.

**Syntax**

```
<MaterialType> FindMaterialByName(<StringType> name)
```

**Example**

```
skinMat = actor.FindMaterialByName("skin")
```

## Materials

**Explanation**

Get a list of material objects available to this figure.

**Arguments**

None

**Syntax**

```
<MaterialType List> Materials()
```

## Memorize

**Explanation**

Set the figure's default parameter values to the current values so that the figure can be reset to this pose later (See figure.Reset()).

**Arguments**

None

**Syntax**

```
<NoneType> Memorize()
```

## Name

**Explanation**

Get the figure's external name.  This is the name given in Poser's GUI pull-down menus.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## NunIkChains

**Explanation**

Get the number of inverse kinematics chains attached to this figure..

**Arguments**

None

**Syntax**

```
<IntType> NumIkChains()
```

## ParentActor

**Explanation**

Get the parent actor of the figure.  Initially, the parent of a figure is typically the "Body" actor.

**Arguments**

None

**Syntax**

```
<ActorType> ParentActor()
```

## Reset

**Explanation**

Reset figure to default or last memorized pose. (See figure.Memorize().

**Arguments**

None

**Syntax**

```
<NoneType> Reset()
```

## SetConformTarget

**Explanation**

Pose this figure to match the specified figure’s pose.

**Arguments**

Enter the figure to conform to.

**Syntax**

```
<NoneType> SetConformTarget(<FigureType> conformee)
```

**Example**

```
SetConformTarget(Figure1)
```

## SetDisplayStyle

**Explanation**

Set display style of the figure.

**Arguments**

Enter a valid display code.

**Syntax**

```
<NoneType> SetDisplayStyle(<IntType> displayCode)
```

**Example**

```
fig.SetDIsplayStyle(poser.kDIsplayCodeFLATLINED)
```

## SetIkStatus

**Explanation**

Set the status of inverse kinematics for the specified IK chain index.

**Arguments**

Enter an index between 0 and figure.NumIkChains(), as well as a value specifying the status (0 for off, 1 for on).

**Syntax**

```
<NoneType> SetIkStatus(<IntType> whichLimb, <IntType> on)
```

**Example**

```
figure.SetIkStatus(0, 1)
```

## SetMeAsStartupScript

**Explanation**

Specify the current script as the Python script associated with the current Poser doc and executed on startup when the document is re-opened.

**Arguments**

None

**Syntax**

```
fig.SetMeAsStartupScript()
```

## SetOnOff

**Explanation**

Hide/show the figure.  A value of 1 corresponds to "on" while a value of 0 corresponds to "off"

**Arguments**

Enter 1 to toggle the current figure visible, or 0 to toggle it invisible.

**Syntax**

```
<NoneType> SetOnOff(<IntType> on)
```

**Example**

```
fig.SetOnOff(1)
```

## SetParentActor

**Explanation**

Set the parent actor of the figure.  The entire figure will be affected by parameter changes to its parent.  Initially, the parent of a figure is typically the "Body" actor.

**Arguments**

Enter an actor which is to become the new parent of the figure.

**Syntax**

```
<NoneType> SetParentActor(<ActorType> newParent)
```

**Example**

```
fig.SetParentActor(someActor)
```

## SetStartupScript

**Explanation**

Specify the Python script to associate with the current Poser document and executed on startup when the file is re-opened. The filename should be a path (either absolute or relative to the Poser folder).

**Arguments**

Enter the complete path and file name.

**Syntax**

```
<NoneType> SetStartupScript(<StringType> filePath)
```

**Example**

```
fig.SetStartupScript("Runtime\Python\script.py")
```

## SetVisible

**Explanation**

Set the display status of the figure in the scene.

**Arguments**

Enter 1 to display the figure, and 0 to turn figure display off.

**Syntax**

```
<NoneType> SetVisible(<IntType> visible)
```

## StartupScript

**Explanation**

Return the Python script to be associated with the current Poser document and executed on startup when the document is reopened. The returned filename is a path (either absolute or relative to the Poser folder).

**Arguments**

None

**Syntax**

```
<StringType> StartupScript()
```

## StraightenBody

**Explanation**

Straighten the figure's torso area.

**Arguments**

None

**Syntax**

```
<NoneType> StraightenBody()
```

## SwapBottom

**Explanation**

Swap the orientations of the bottom left and bottom right body parts.

**Arguments**

None

**Syntax**

```
<NoneType> SwapBottom()
```

## SwapTop

**Explanation**

Swap the orientations of the top left and top right body parts.

**Arguments**

None

**Syntax**

```
<NoneType> SwapTop()
```

## SymmetryBotLeftToRight

**Explanation**

Copy the bottom left side parameters of the figure to the bottom right side.

**Arguments**

Defaults to 0.  Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmetryBotLeftToRight({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryBotLeftToRight(1)
```

## SymmetryBotRightToLeft

**Explanation**

Copy the bottom left side parameters of the figure to the bottom right side.

**Arguments**

Defaults to 0.  Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmetryBotRightToLeft({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryBotRightToLeft(1)
```

## SymmetryLeftToRight

**Explanation**

Copy the left side parameters of the figure to the right side.

**Arguments**

Defaults to 0.  Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmmetryLeftToRight({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryLeftToRight(1)
```

## SymmetryRightToLeft

**Explanation**

Copy the right side parameters of the figure to the left side.

**Arguments**

Defaults to 0.  Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmmetryRightToLeft({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryRightToLeft(1)
```

## SymmetryTopLeftToRight

**Explanation**

Copy the top left side parameters of the figure to the top right side.

**Arguments**

Defaults to 0.  Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmetryTopLeftToRight({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryTopLeftToRight(1)
```

## SymmetryTopRightToLeft

**Explanation**

Copy the top left side parameters of the figure to the top right side.

**Arguments**

Defaults to 0. Enter a value of 1 to specify that the joints should be copied.

**Syntax**

```
<NoneType> SymmetryTopRightToLeft({<IntType> copyJoints})
```

**Example**

```
fig.SymmetryTopRightToLeft(1)
```

## UnimeshInfo

**Explanation**

Get a 3-item tuple containing as its first item the geometry of this figure as a single mesh. The second item is a list of actors comprising this figure. The third item is a per-actor-vertex-info list. Each item in this list (size of numActors) is a list of vertex indices, specifying the mapping from the original actor vertices, to the vertices of the unimesh geometry.

**Arguments**

None

**Syntax**

```
(<GeomType>, <actor-list>, <per-actor-vertex-info-list>) UnimeshInfo()
```

**Example**

Click the GeomMods button and look for UnimeshDemo sample script.

### Visible

**Explanation**

Query the display status of the figure in the scene. Returns 1 if the figure is currently displayed and 0 if the figure is not currently displayed.

**Arguments**

None

**Syntax**

```
<IntType> Visible()
```

## Material Methods

### AmbientColor

**Explanation**

Get the material's ambient color in the format RGB (values between 0.0 and 1.0).

**Arguments**

None

**Syntax**

```
(<FloatType> r, <FloatType> g, <FloatType> b) AmbientColor()
```

### BumpMapFileName

**Explanation**

Get the material's bump map filename.

**Arguments**

None

**Syntax**

```
<StringType> BumpMapFileName()
```

## BumpStrength

**Explanation**

Get the material's bump strength value.

**Arguments**

None

**Syntax**

```
<FloatType> BumpStrength()
```

## DiffuseColor

**Explanation**

Get the material's diffuse color in the format RGB (values between 0.0 to 1.0).

**Arguments**

None

**Syntax**

```
(<FloatType> r, <FloatType> g, <FloatType> b) DiffuseColor()
```

## ExtName

**Explanation**

Get the external name for the material.

**Arguments**

None

**Syntax**

```
<StringType> ExtName()
```

## LoadMaterialSet

**Explanation**

Load a material from the Library.

**Arguments**

Specify the name of the material you wish to load.

**Syntax**

```
<NoneType> LoadMaterialSet(<StringType> name)
```

## Name

**Explanation**

Get the material's internal name.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## Ns

**Explanation**

Get the material's Ns value.  An Ns value is an exponent used in calculating the specular highlight.  A higher exponent results in a smaller highlight and vice-versa.

**Arguments**

None

**Syntax**

```
<FloatType> Ns()
```

## ReflectionColor

**Explanation**

Get the material's reflection color in the format RGB (values from 0.0 to 1.0).

**Arguments**

None

**Syntax**

```
(<FloatType> r, <FloatType> g, <FloatType> b) ReflectionColor()
```

## ReflectionMapFileName

**Explanation**

Get the material's reflection map filename.

**Arguments**

None

**Syntax**

```
<StringType> ReflectionMapFileName()
```

## ReflectionStrength

**Explanation**

Get the material's reflection strength value.

**Arguments**

None

**Syntax**

```
<FloatType> ReflectionStrength()
```

## SaveMaterialSet

**Explanation**

Save this material to the Library.

**Arguments**

Specify the name of the material you wish to save.

**Syntax**

```
<NoneType> SaveMaterialSet(<StringType> name)
```

## Selected

**Explanation**

Query whether this material is selected.  Use material selection in conjunction with the Actor.  See the SaveMaterialCollection method in the Actor Methods section.

**Arguments**

None

**Syntax**

```
<IntType> Selected()
```

## SetAmbientColor

**Explanation**

Set the material's ambient color.

**Arguments**

None

**Syntax**

```
<NoneType> SetAmbientColor(<FloatType> r, <FloatType> g, <FloatType> b)
```

## SetBumpMapFileName

**Explanation**

Set the material's bump map filename.

**Arguments**

Enter a valid path and filename.

**Syntax**

```
<NoneType> SetBumpMapFileName(<StringType> filePath)
```

**Example**

```
mat.SetBumpMapFilename("E:\Materials\Stone1.bmp")
```

## SetBumpStrength

**Explanation**

Set the material bump strength to a specific value.

**Arguments**

Enter a value between –1.0 and 1.0.

**Syntax**

```
<FloatType> SetBumpStrength(<FloatType> value)
```

**Example**

```
mat.SetBumpStrength(0.5)
```

## SetDiffuseColor

**Explanation**

Set the material's diffuse color.

**Arguments**

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
<NoneType> SetDiffuseColor(<FloatType> r, <FloatType> g, <FloatType> b)
```

**Example**

```
mat.SetDiffuseColor(0.46,0.57,0.33)
```

## SetNs

**Explanation**

Set the material's Ns to a specific value.  This is an exponent used in the calculation of the specular highlight.  A higher exponent results in a smaller highlight and vice-versa.

**Arguments**

Enter an exponent value.

**Syntax**

```
<FloatType> SetNs(<FloatType> value)
```

**Example**

```
mat.SetNs(0.5)
```

## SetReflectionColor

**Explanation**

Set the material's reflection color.

**Arguments**

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

**Syntax**

```
<NoneType> SetReflectionColor(<FloatType> r, <FloatType> g, <FloatType> b)
```

**Example**

```
mat.SetReflectionColor(0.46,0.57,0.33)
```

## SetReflectionMapFileName

**Explanation**

Set the material's reflection map filename.

**Arguments**

Enter a valid path and file name.

**Syntax**

```
<NoneType> SetReflectionMapFileName(<StringType> filePath)
```

**Example**

```
mat.SetReflectionMapStrength("C:\My Documents\myrefmap.bmp")
```

## SetReflectionStrength

**Explanation**

Set the material's reflection strength to a specific value.

**Arguments**

Enter a value between –1.0 and 1.0.

**Syntax**

```
<FloatType> SetReflectionStrength(<FloatType> value)
```

**Example**

```
mat.SetReflectionStrength(0.5)
```

## SetSelected

**Explanation**

Selects or deselects the material for inclusion in a material collection. A value of 1 selects the material, a value of 0 deselects. Use material selection in conjunction with the Actor. See the SaveMaterialCollection method in the Actor Methods section.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetSelected(<IntType> selected)
```

## SetSpecularColor

**Explanation**

Set the material's specular highlight color.

### Arguments

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

### Syntax

```
<NoneType> SetSpecularColor(<FloatType> r, <FloatType> g, <FloatType> b)
```

### Example

```
mat.SetSpecularColor(0.46,0.57,0.33)
```

## SetTextureColor

### Explanation

Set the material texture color in the format RBG.

### Arguments

- R: Enter the red value from 0.0 to 1.0.
- G: Enter the green value from 0.0 to 1.0.
- B: Enter the blue value from 0.0 to 1.0.

### Syntax

```
<NoneType> SetTextureColor(<FloatType> r, <FloatType> g, <FloatType> b,
<FloatType> )
```

### Example

```
mat.SetTextureColor(0.46,0.57,0.33)
```

## SetTextureMapFileName

**Explanation**

Set the material’s texture map filename.

**Arguments**

Enter the path and file name.

**Syntax**

```
<NoneType> SetTextureMapFileName(<StringType> filePath)
```

**Example**

```
mat.SetTexttureMapFileName(“C:\Files\Textures\ tex1.bmp”)
```

## SetTransparencyExpo

**Explanation**

Set the material’s tExpo to a specific value.  The tExpo parameter corresponds to the falloff of the the rate at which the transparency becomes opaque on the edges of an object.

**Arguments**

Enter a value between 0.0 and 10.0.

**Syntax**

```
<FloatType> SetTransparencyExpo(<FloatType> value)
```

**Example**

```
mat.SetTransparencyExpo(5.0).
```

## SetTransparencyMapFileName

**Explanation**

Set the material's transparency map filename.

**Arguments**

Enter the path and filename.

**Syntax**

```
<NoneType> SetTransparencyMapFileName(<StringType> filePath)
```

**Example**

```
mat.SetTransparencyMapFileName("C:\Files\ trans1.bmp")
```

## SetTransparencyMax :

**Explanation**

Set the material's tMax to a specific value. The tMax parameter is the maximum transparency allowable. A high TransparencyMax value makes the object very transparent in parts of the surface which face the eye.

**Arguments**

Enter a value between 0.0 and 1.0.

**Syntax**

```
<FloatType> SetTransparencyMax(<FloatType> value)
```

**Example**

```
mat.SetTransparencyMax(0.5)
```

## SetTransparencyMin

**Explanation**

Set the material's tMin to a specific value. The tMin parameter is the minimum transparency allowable. A high TransparencyMin value makes the objet very transparent on
its edges.

**Arguments**

Enter a value between 0.0 and 1.0.

**Syntax**

```
<FloatType> SetTransparencyMin(<FloatType> value)
```

**Example**

```
mat.SetTransparencyMin(0.5)
```

## ShaderTree

**Explanation**

Get the material's shader tree.

**Arguments**

None

**Syntax**

```
<ShaderTreeType> ShaderTree()
```

## SpecularColor

**Explanation**

**Get the material's specular highlight color in the format RGB (values between 0.0 and 1.0)Arguments**

None

**Syntax**

```
(<FloatType> r, <FloatType> g, <FloatType> b) SpecularColor()
```

## TextureColor

**Explanation**

Get the material's texture color in the format RG (values from 0.0 and 1.0).

**Arguments**

None

**Syntax**

```
(<FloatType> r, <FloatType> g, <FloatType> b) TextureColor()
```

## TextureMapFileName

**Explanation**

Get the material's texture map filename.

**Arguments**

None

**Syntax**

```
<StringType> TextureMapFileName()
```

## TransparencyExpo

### Explanation

Get the material's tExpo value.   The TransparencyExpo value determines the rate at which the transparency becomes opaque on the edges of an object.

### Arguments

None

### Syntax

```
<FloatType> TransparencyExpo()
```

## TransparencyMapFileName

### Explanation

Get the material's transparency map filename.

### Arguments

None

### Syntax

```
<StringType> TransparencyMapFileName()
```

## TransparencyMax

### Explanation

Get the tMax value for the material.   A high TransparencyMax value makes the object very transparent in parts of the surface which face the eye.

**Arguments**

None

**Syntax**

```
<FloatType> TransparencyMax()
```

## TransparencyMin

**Explanation**

Get the tMin value for the material. A high TransparencyMin value makes the object very transparent on its edges.

**Arguments**

None

**Syntax**

```
<FloatType> TransparencyMin()
```

# Parameter Methods

## Actor

**Explanation**

Return the actor which owns this parm.

**Arguments**

None

**Syntax**

```
<ActorType> Actor()
```

## AddKeyFrame

**Explanation**

Add a key frame for this parameter at the specified frame. If no frame is specified, a keyframe will be added at the current frame.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<NoneType> AddKeyFrame({<IntType> frame})
```

**Example**

```
AddKeyFrame(81)
```

## ApplyLimits

**Explanation**

Apply minimum and maximum limits to parameter.

**Arguments**

None

**Syntax**

```
<NoneType> ApplyLimits()
```

## ClearUpdateCallback

**Explanation**

Clear update callback for calculating this parameter value if it is set.

**Arguments**

None

**Syntax**

```
<NoneType> ClearUpdateCallback()
```

## ConstantAtFrame

**Explanation**

Query whether or not the given frame is within a constant range. If no frame is specified, the default frame is the current frame

**Arguments**

Optionally, enter a valid frame number.

**Syntax**

```
<IntType> ConstantAtFrame({<IntType> frame})
```

**Example**

```
parm.ConstantAtFrame(12)
```

## DeleteKeyFrame

**Explanation**

Delete a key frame for this actor at the specified frame.  If no frame is specified, a keyframe will be deleted at the current frame.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<NoneType> DeleteKeyFrame({<IntType> frame})
```

**Example**

```
parm.DeleteKeyFrame(30)
```

## Hidden

**Explanation**

Query whether or not the current parameter is hidden from the user interface (UI).  Returns 1 if hidden, 0 if visible.

**Arguments**

None

**Syntax**

```
<IntType> Hidden()
```

## InternalName

**Explanation**

Get the parameter's internal name.

**Arguments**

None

**Syntax**

```
<StringType> InternalName()
```

## IsMorphTarget

**Explanation**

Query whether or not this parameter is a morph target parameter.  Returns 1 if it is and 0 if it is not.

**Arguments**

None

**Syntax**

```
<IntType> IsMorphTargetParamter()
```

## IsValueParameter

**Explanation**

Query whether or not this parameter is a value parameter.  This type of parameter is not linked to Poser elements such as figures, props, etc.  Rather, it can be used to add user interface control to your Python scripts. Returns 0 if not a value parameter, 1 if yes.

**Arguments**

None

**Syntax**

```
<IntType> IsValueParamter()
```

## LinearAtFrame

**Explanation**

Query whether or not the given frame is within a linear range.  If no frame is specified, the default frame is the current frame.

**Arguments**

Optionally, enter a valid frame number.

**Syntax**

```
<IntType> LinearAtFrame({<IntType> frame})
```

**Example**

```
parm.LinearAtFrame(12)
```

## MaxValue

**Explanation**

Get the current parameter's maximum value.

**Arguments**

None

**Syntax**

```
<FloatType> MaxValue()
```

## MinValue

**Explanation**

Get the parameter's current minimum value.

**Arguments**

None

**Syntax**

```
<FloatType> MinValue()
```

## MorphTargetDelta

**Explanation**

If this parameter is a morph target parameter, return the morph target “delta” associated with the specified vertex index. Note: Deltas are 3D differences between the original geometry and the morphed geometry.

**Arguments**

None

**Syntax**

```
(<FloatType> deltaX, <FloatType> deltaY, <FloatType> deltaZ)
MorphTargetDelta(<IntType> vertexIndex)
```

## Name

**Explanation**

Get the parameter’s external name.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## NextKeyFrame

**Explanation**

Get the next frame in which the parameter has a key frame set.

**Arguments**

None

**Syntax**

```
<IntType> NextKeyFrame()
```

## PrevKeyFrame

**Explanation**

Get the previous frame in which the parameter has a key frame set.

**Arguments**

None

**Syntax**

```
<IntType> PrevKeyFrame()
```

## Sensitivity

**Explanation**

Get the sensitivity of the mouse tracking (on the user interface).

**Arguments**

None

**Syntax**

```
<FloatType> Sensitivity()
```

## SetHidden

**Explanation**

Set the hidden status of this parameter.

**Arguments**

- 0: Set the parameter as visible.
- 1: Set the parameter as hidden.

**Syntax**

```
<NoneType> SetHidden(<IntType> hide)
```

**Example**

```
parm.SetHidden(1)
```

## SetInternalName

**Explanation**

Set the parameter's internal name.

**Arguments**

None

**Syntax**

```
<NoneType> SetInternalName()
```

## SetMaxValue

**Explanation**

Set the parameter’s maximum value.

**Arguments**

Enter a valid value for the current parameter.

**Syntax**

```
<FloatType> SetMaxValue(<FloatType> value)
```

**Example**

```
parm.SetMaxValue(100.00)
```

## SetMinValue

**Explanation**

Set the parameter’s minimum value.

**Arguments**

Enter a valid value for the current parameter.

**Syntax**

```
<FloatType> SetMinValue(<FloatType> value)
```

**Example**

```
parm.SetMinValue(1.35)
```

## SetMorphTargetDelta

**Explanation**

If this parameter is a morph target parameter, set the morph target "delta" value associated with the specified vertex index. Note: Deltas are 3D differences between the original geometry and the morphed geometry.

**Arguments**

- Vertex Index: Enter the array index that identifies the desired vertex.
- Delta X: Enter the change in the X component of the vertex.
- Delta Y: Enter the change in the Y component of the vertex.
- Delta Z: Enter the change in the Z component of the vertex.

**Syntax**

```
<NoneType> SetMorphTargetDelta(<IntType> vertexIndex, <FloatType> deltaX,
<FloatType> deltaY, <FloatType> deltaZ)
```

**Example**

```
parm.SetMorphTargetDelta( vertexIndex, 0.12, 0.34, 0.45)
```

## SetName

**Explanation**

Set the parameter's external name.

**Arguments**

Enter a valid name for the current parameter.

**Syntax**

```
<NoneType> SetName(<StringType> name)
```

### Example

```
parm.SetName("Test1")
```

## SetRangeSpline

### Explanation

Set the given frame range to have spline interpolation between key frames.  Automatically sets key frames at start and end of specified range.

### Arguments

Enter a valid starting and ending frame.

### Syntax

```
<NoneType> SetRangeSpline(<IntType> startFrame, <IntType> endFrame)
```

### Example

```
parm.SetRangeSpline(10,20)
```

## SetSensitivity

### Explanation

Set the sensitivity of the mouse tracking (on the user interface).

### Arguments

None

### Syntax

```
<NoneType> SetSensitivity(<FloatType> value)
```

## SetUpdateCallback

**Explanation**

Set a per-update callback for calculating this parameter value.

**Arguments**

The callback function should take the parameter and the parameters current value as callbacks.

**Syntax**

```
<NoneType> SetUpdateCallback(<FunctionType> newCB, {<Object> cbArgs})
```

**Example**

(See sample scripts)

## SetValue

**Explanation**

Set the parameter to a specific value.

**Arguments**

None

**Syntax**

```
<NoneType> SetValue(<FloatType> value)
```

## SplineAtFrame

**Explanation**

Query whether or not the given frame is within a spline range.  If no frame is specified, the default frame is the current frame.

**Arguments**

Optionally, enter a valid frame number.

**Syntax**

```
<IntType> SplineAtFrame({<IntType> frame})
```

**Example**

```
parm.SplineAtFrame(32)
```

## SetRangeConstant

**Explanation**

et the given frame range to have constant (step) interpolation between keyframes.  Automatically sets key frames at start and end of specified.

**Arguments**

Enter valid start and end frame numbers.

**Syntax**

```
<NoneType> SetRangeConstant(<IntType> startFrame, <IntType> endFrame)
```

**Example**

```
parm.SetRangeConstant(12,32)
```

## SetRangeLinear

**Explanation**

Set the given frame range to have linear interpolation between key frames.  Automatically sets key frames at start and end of specified range.

### Arguments

Enter valid start and end frame numbers.

### Syntax

```
<NoneType> SetRangeLinear(<IntType> startFrame, <IntType> endFrame)
```

### Example

```
parm.SetRangeLinear(12,32)
```

## SetSplineBreak

### Explanation

Break spline interpolation at the given frame. If the frame is not a keyframe, no action will be taken.  If no frame is specified, the default frame is the current frame.  A broken spline can be un-broken by passing a 0 as the second argument to this method.

### Arguments

Enter a valid frame number.  Optionally, add 0 to u-break the spline.

### Syntax

```
<NoneType> SetSplineBreak({<IntType> frame, <IntType> on})
```

### Example

```
parm.SetSplineBreak(12,0)
```

## SplineBreakAtFrame

### Explanation

Query whether or not the given frame is a spline-break. If the frame is not a keyframe, no action will be taken.  If no frame is specified, the default frame is the current frame.

**Arguments**

Enter a valid frame number.

**Syntax**

```
<IntType> SplineBreakAtFrame({<IntType> frame})
```

**Example**

```
parm. SplineBreakAtFrame(12)
```

## TypeCode

**Explanation**

Get the type code for this parameter. Type codes are enumerated values, such as `poser.kParmCodeXROT`.

**Arguments**

None

**Syntax**

```
<IntType> TypeCode()
```

## Value

**Explanation**

Get the parameter's current value.

**Arguments**

None

**Syntax**

```
<FloatType> Value()
```

# Geometry Methods

## AddGeneralMesh

### Explanation

Add a general mesh to existing geometry. Arguments are numerical Python arrays specifying polygons, sets, and vertices, as well as optionally texture-polygons, texture-sets, and texture-vertices. (See Numerical Python documentation for more details).

### Arguments

Required:

- Polygons: Enter the Numerical array specifying the polygonal connectivity of the sets. Each polygon stores the starting set index and the number of vertices it contains.
- Sets: Enter the Numerical array containing the IDs of the vertices. Each ID is an integer corresponding to the position of the vertex in the vertex array.
- Vertices: Enter the Numerical array containing the actual positions of the vertices. Each vertex has an X, Y, and Z component.

Optional:

- Texture Polygons: Enter the Numerical array specifying the polygonal connectivity of the texture-sets. Each polygon stores the starting set index and the number of vertices it contains.
- Texture Sets: Enter the Numerical array containing the IDs of the texture vertices. Each ID is an integer corresponding to the position of the vertex in the vertex array.
- Texture Vertices: Enter the Numerical array containing the actual positions of the texture vertices. Each vertex has an X, Y, and Z component.

### Syntax

```
<NoneType> AddGeneralMesh(<IntType nx2 Numeric.Array> polygons, <IntType nx1
Numeric.Array> sets, <FloatType nx3 Numeric.Array> vertices, {<IntType nx2
Numeric.Array> texPolygons, <IntType nx1 Numeric.Array> texSets, <FloatType nx3
Numeric.Array> texVertices})
```

**Example**

See sample scripts.

## AddMaterialName

**Explanation**

Adds a material name to the geometry material name list and returns its index.

**Arguments**

Enter the name for the new material.

**Syntax**

```
<IntType> AddMaterialName(<StringType> name)
```

**Example**

```
j = geom.AddMaterialName("Chartreux")
```

## AddPolygon

**Explanation**

Add a polygon to existing geometry by providing a 2-dimensional nx3 numerical python array of vertices.  (See Numerical Python documentation for more details).  Returns the newly added polygon object.

**Arguments**

Enter a Numerical array containing the actual positions of the vertices for the new polygon.  Each vertex has an X, Y, and Z component.

**Syntax**

```
<PolygonType> AddPolygon(<FloatType nx3 Numeric.Array> vertices)
```

**Example**

```
poly = geom.AddPolygon(newVerts)
```

## AddTriangle

**Explanation**

Add a triangle to existing geometry by specifying 3 points.

**Arguments**

Enter a tuple of tuples specifying the vertices of the new triangle to add.

**Syntax**

```
<StringType list> Groups()
```

**Example**

```
poly = geom.AddTriangle((1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0))
```

## Groups

**Explanation**

Get the list of group names for this geometry. Groups can be created using the grouping tool in the Poser GUI.

**Arguments**

None

**Syntax**

```
<StringType list> Groups()
```

## Materials

**Explanation**

Get a list of material objects of the geometry.

**Arguments**

None

**Syntax**

```
<MaterialType List> Materials()
```

## Normals

**Explanation**

Get a list of vertex normals.  Each normal is a vertex object.

**Arguments**

None

**Syntax**

```
<VertType List> Normals()
```

## NumMaterials

**Explanation**

Get the number of materials in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumMaterials()
```

## NumNormals

**Explanation**

Get the number of normals in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumNormals()
```

## NumPolygons

**Explanation**

Get the number of polygons in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumPolygons()
```

## NumSets

**Explanation**

Get the number of sets in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumSets()
```

## NumTexPolygons

**Explanation**

Get the number of texture polygons in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumTexPolygons()
```

## NumTexSets

**Explanation**

Get the number of texture sets in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumTexSets()
```

## NumTexVertices

**Explanation**

Get the number of texture vertices in the geometry..

**Arguments**

None

**Syntax**

```
<IntType> NumTexVertices()
```

## NumVertices

**Explanation**

Get the number of vertices in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumVertices()
```

## Polygon

**Explanation**

Get a polygon object by specifying its index.

**Arguments**

Enter a valid polygon index.

**Syntax**

```
<PolygonType> Polygon(<IntType> index)
```

**Example**

```
geom.Polygon(3)
```

## Polygons

**Explanation**

Get a list of polygon objects.  Each polygon object can be queried for a start set index as well as the number of vertices in the polygon.

**Arguments**

None

**Syntax**

```
<PolygonType List> Polygons()
```

## Sets

**Explanation**

Get a set list for the geometry.  Each set value is an index into the vertex list.

**Arguments**

None

**Syntax**

```
<IntType List> Sets()
```

## TexPolygons

**Explanation**

Get a list of texture polygon objects.  Each texture polygon object can be queried for a start set index as well as the number of vertices in the polygon.

**Arguments**

None

**Syntax**

```
<TexPolygonType List> TexPolygons()
```

## TexSets

**Explanation**

Get a texture set list for the geometry.  Each texture set value is an index into the texture vertex list.

**Arguments**

None

**Syntax**

```
<IntType List> TexSets()
```

## TexVertices

**Explanation**

Get a list of texture vertex objects.  Each vertex object can be queried for U and V values.

**Arguments**

None

**Syntax**

```
<TexVertType List> TexVertices()
```

## Vertex

**Explanation**

Get a (model space) vertex object by specifying its index.

**Arguments**

None

**Syntax**

```
<VertType> Vertex(<IntType> index)
```

## Vertices

**Explanation**

Get a list of vertex objects.  Each vertex object can be queried for x,y,z values.

**Arguments**

None

**Syntax**

```
<VertType List> Vertices()
```

## Weld

**Explanation**

Share similar vertices.  This call smoothes objects by homogenizing normals for coincident vertices.

**Arguments**

None

**Syntax**

```
<NoneType> Weld()
```

## WorldNormals

**Explanation**

Get a list of vertex normals in world space if possible.  If no world space data is available, the function will return None. Each normal is a vertex object.

**Arguments**

None

**Syntax**

```
<VertType List> WorldNormals()
```

### WorldVertex

**Explanation**

If possible, get a (world space) vertex object by specifying its index.  If no world space data is available, the function will return None.

**Arguments**

None

**Syntax**

```
<VertType> WorldVertex(<IntType> index)
```

### WorldVertices

**Explanation**

Get the vertices of the geometry in world space if possible.  If no world space data is available, the function will return None.

**Arguments**

None

**Syntax**

```
<VertType List> WorldVertices()
```

## Vertex Methods

### SetX

**Explanation**

Set X coordinate.

**Arguments**

Enter a valid coordinate.

**Syntax**

```
<NoneType> SetX(<FloatType> value)
```

**Example**

```
vert.SetX(4.11)
```

## SetY

**Explanation**

Set Y coordinate.

**Arguments**

Enter a valid coordinate.

**Syntax**

```
<NoneType> SetY(<FloatType> value)
```

**Example**

```
vert.SetY(2.25)
```

## SetZ

**Explanation**

Set Z coordinate.

**Arguments**

Enter a valid coordinate.

**Syntax**

```
<NoneType> SetZ(<FloatType> value)
```

**Example**

```
vert.SetZ(6.52)
```

## X

**Explanation**

Get X coordinate.

**Arguments**

None

**Syntax**

```
<FloatType> X()
```

## Y

**Explanation**

Get Y coordinate.

**Arguments**

None

**Syntax**

```
<FloatType> Y()
```

### Z

**Explanation**

Get Z coordinate.

**Arguments**

None

**Syntax**

```
<FloatType> Z()
```

## Polygon Methods

### Groups

**Explanation**

Return a list of groups in which this polygon is included. Groups can be created using the grouping tool in the Poser GUI."

**Arguments**

None

**Syntax**

```
<StringType list> Groups()
```

### InGroup

**Explanation**

Determine if the polygon is in the specified group.  Groups can be created using the grouping tool in the Poser GUI.

**Arguments**

Enter a valid group name.

**Syntax**

```
<IntType> InGroup(<StringType> groupName)
```

**Example**

```
poly.InGroup("MyNewGroup")
```

## IncludeInGroup

**Explanation**

Include the polygon in the specified group.  Groups can be created using the grouping tool in the Poser GUI.

**Arguments**

Enter a valid group name.

**Syntax**

```
<NoneType> IncludeInGroup(<StringType> groupName)
```

**Example**

```
poly.IncludeInGroup("MyNewGroup")
```

## MaterialIndex

**Explanation**

Get the material index of the element.  This is an index into the list of materials of this geometry object.

**Arguments**

None

**Syntax**

```
<IntType> MaterialIndex()
```

## MaterialName

**Explanation**

Get the element's material name.

**Arguments**

None

**Syntax**

```
<StringType> MaterialName()
```

## NumVertices

**Explanation**

Get the number of vertices in the polygon.

**Arguments**

None

**Syntax**

```
<IntType> NumVertices()
```

## RemoveFromGroup

**Explanation**

Remove the polygon from the specified group.  Groups can be created using the grouping tool in the Poser GUI.

**Arguments**

Enter a valid group name.

**Syntax**

```
<NoneType> RemoveFromGroup(<StringType> groupName)
```

**Example**

```
poly.RemoveFromGroup("MyNewGrouMethod Name
```

## SetMaterialIndex

**Explanation**

Set the polygon's material index.  This is an index into the list of materials of this geometry object.

**Arguments**

Enter the index of the desired material.

**Syntax**

```
<NoneType> SetMaterialIndex(<IntType> index)
```

**Example**

```
poly.SetMaterialIndex(3)
```

## SetMaterialName

**Explanation**

Set the material name of the polygon, returns material index.

**Arguments**

Enter a name for the polygon's material.

**Syntax**

```
<IntType> SetMaterialName(<StringType> name)
```

**Example**

```
poly.SetMaterialName("cotton")
```

## Start

**Explanation**

Get the starting set index of the element.  Using this value to index into the set list, one can get the index of the associated vertex.

**Arguments**

None

**Syntax**

```
<IntType> Start()
```

## Vertices

**Explanation**

Get a list of vertex objects for the polygon.

**Arguments**

None

**Syntax**

```
<VertType List> Vertices()
```

## TexPolygon Methods

### NumTexVertices

**Explanation**

Get the number of texture vertices in the geometry.

**Arguments**

None

**Syntax**

```
<IntType> NumTexVertices()
```

### Start

**Explanation**

Get the starting set index of the polygon. Using this value to index into the set list, one can get the index of the associated texture vertex.

**Arguments**

None

**Syntax**

```
<IntType> Start()
```

### TexVertices

**Explanation**

Get a list of texture vertex objects for the polygon.

**Arguments**

None

**Syntax**

```
<TexVertType List> TexVertices()
```

## TexVertex Methods

### SetU

**Explanation**

Set U coordinate.

**Arguments**

Enter a U coordinate.

**Syntax**

```
<NoneType> SetU(<FloatType> value)
Geom.SetU(.96)
```

## SetV

**Explanation**

Set V coordinate.

**Arguments**

Enter a V coordinate.

**Syntax**

```
<NoneType> SetU(<FloatType> value)
```

**Example**

```
geom.SetV(.96)
```

## U

**Explanation**

Get U coordinate.

**Arguments**

None

**Syntax**

```
<FloatType> U()
```

## V

**Explanation**

Get V coordinate.

**Arguments**

None

**Syntax**

```
<FloatType> V()
```

## Shader Tree Methods

This class of methods was introduced in Poser 6.0.0.

### AttachTreeNodes

**Explanation**

Connect a ShaderNode's output to another ShaderNode's input.

**Arguments**

Specify the input to which you wish to connect (parameter), the node on which that input resides (node1), and the node whose output you are connecting (node2).

**Syntax**

```
<NoneType> AttachTreeNodes(<ShaderNodeType> node1, <StringType> parameter,
<ShaderNodeType> node2)
```

### CreateNode

**Explanation**

Create a new ShaderNode.

#### Arguments

Specify the type of ShaderNode you wish to create.

#### Syntax

```
<ShaderNodeType> CreateNode(<StringType> type)
```

## DeleteNode

#### Explanation

Delete a node from the ShaderTree.

#### Arguments

Specify the number of the node you wish to delete.

#### Syntax

```
<NoneType> DeleteNode(<IntType> i)
```

## DetachTreeNode

#### Explanation

Detach a node from a specified input on a specified node.

#### Arguments

Specify the node from which you want to detach (node), and the specific input on that node from which you are detaching (parameter).

#### Syntax

```
<NoneType> DetachTreeNode(<ShaderNodeType> node, <StringType> parameter)
```

## Node

**Explanation**

Get the node number " i " in this ShaderTree.

**Arguments**

Specify the node number you wish to get.

**Syntax**

```
<ShaderNodeType> Node(<IntType> i)
```

## NodeByInternalName

**Explanation**

Get a ShaderNode by using its internal name.

**Arguments**

Specify the internal name of the ShaderNode you wish to get.

**Syntax**

```
<ShaderNodeType> NodeByInternalName(<StringType> name)
```

## Nodes

**Explanation**

Get a list of all nodes in this ShaderTree.

**Arguments**

None

**Syntax**

```
<ListType> Nodes()
```

### NumNodes

**Explanation**

Get the number of nodes in this ShaderTree.

**Arguments**

None

**Syntax**

```
<IntType> NumNodes()
```

### UpdatePreview

**Explanation**

Tell Poser that this ShaderTree has been modified.

**Arguments**

None

**Syntax**

```
<NoneType> UpdatePreview()
```

## Shader Node Methods

This class of methods was introduced in Poser 6.0.0.

## ConnectToInput

**Explanation**

Connect the output of this node to the given input

**Arguments**

Specify a node input

**Syntax**

```
<NoneType> ConnectToInput(<ShaderNodeInputType> Input)
```

## Delete

**Explanation**

Delete this node.  (You must not use this ShaderNode object after deleting it.)

**Arguments**

None

**Syntax**

```
<NoneType> Delete()
```

## Input

**Explanation**

Get an input by means of its index.

**Arguments**

Enter an input index number.

**Syntax**

```
<ShaderNodeInputType> Input(<IntType> Index)
```

## InputByInternalName

**Explanation**

Get an input by means of its internal name.

**Arguments**

Enter the internal name of the input you wish to access.

**Syntax**

```
<ShaderNodeInputType> InputByInternalName(<StringType> inputName)
```

## Inputs

**Explanation**

Get a list of all inputs for the current node (the node upon which this method is called).

**Arguments**

None

**Syntax**

```
<ShaderNodeInputType list> Inputs()
```

## InputsCollapsed

**Explanation**

Query whether the node's inputs are collapsed in the UI.

**Arguments**

None

**Syntax**

```
<IntType> InputsCollapsed()
```

## InternalName

**Explanation**

Get the internal name for the node.

**Arguments**

None

**Syntax**

```
<StringType> InternalName()
```

## Location

**Explanation**

Get the UI position of this node in pixels.

**Arguments**

None

**Syntax**

```
(<IntType> x, <IntType> y) Location()
```

## Name

**Explanation**

Get the (external) name of the node.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## NumInputs

**Explanation**

Get the number of this ShaderNode's inputs.

**Arguments**

None

**Syntax**

```
<IntType> NumInputs()
```

## PreviewCollapsed

**Explanation**

Query whether the node's preview is collapsed in the UI.

**Arguments**

None

**Syntax**

```
<IntType> PreviewCollapsed()
```

## SetInputsCollapsed

**Explanation**

Set whether the node's inputs are collapsed in the UI.

**Arguments**

Enter 1 to collapse inputs, and 0 to disable input collapse.

**Syntax**

```
<NoneType> SetInputsCollapsed(<IntType> Collapsed)
```

## SetLocation

**Explanation**

Set the UI position of this node in pixels.

**Arguments**

The x and y Arguments specify the coordinates of the node location on the Material palette.

**Syntax**

```
<NoneType> SetLocation(<IntType> x, <IntType> y)
```

## SetName

**Explanation**

Set the (external) name of the node.

**Arguments**

Enter a string for the name

**Syntax**

```
<NoneType> SetName(<StringType> Name)
```

## SetPreviewCollapsed

**Explanation**

Set whether the preview is collapsed in the UI.

**Arguments**

Enter 1 to collapse the preview, or 0 to disable preview collapse.

**Syntax**

```
<NoneType> SetPreviewCollapsed(<IntType> Collapsed)
```

### Type

**Explanation**

Get the Type of this node.

**Arguments**

None

**Syntax**

```
<StringType> Type()
```

## Shader Node Input Methods

This class of methods was introduced in Poser 6.0.0.

### Animated

**Explanation**

Returns 1 if the input is animated, 0 if it is not animated.

**Arguments**

None

**Syntax**

```
<IntType> Animated()
```

## CanBeAnimated

**Explanation**

Returns 1 if the input can be animated, 0 if the input cannot be animated.

**Arguments**

None

**Syntax**

```
<IntType> CanBeAnimated()
```

## Disconnect

**Explanation**

Disconnects the node that is plugged into this input.

**Arguments**

None

**Syntax**

```
<NoneType> Disconnect()
```

## InNode

**Explanation**

Returns the shader node that is plugged into this input.  Returns none if no shader node is plugged into this input.

**Arguments**

None

**Syntax**

```
<ShaderNodeType> InNode()
```

## InternalName

**Explanation**

Get the internal name of the shader node input.

**Arguments**

None

**Syntax**

```
<StringType> InternalName()
```

## ItsNode

**Explanation**

Returns the shader node to which this input belongs.

**Arguments**

None

**Syntax**

```
<ShaderNodeType> ItsNode()
```

## Name

**Explanation**

Get the (external) name of the shader node input.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## Parameters

**Explanation**

Returns the parameter(s) if the input is animated. Depending on the type of input, the latter two return values may be set to None.

**Arguments**

None

**Syntax**

```
(<ParmType> r, <ParmType> g, <ParmType> b) Parameters()
```

## SetAnimated

**Explanation**

Set the animation status of this input. Set it to 1 for the input to be animated, 0 for it not to be animated.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetAnimated(<IntType> animated)
```

## SetColor

**Explanation**

Set the value of a color or vector input.

**Arguments**

Specify the RGB value of the desired color.

**Syntax**

```
<NoneType> SetColor(<FloatType> r, <FloatType> g, <FloatType> b)
```

## SetFloat

**Explanation**

Set the value of a float, integer, Boolean or menu input.

**Arguments**

Enter the value you wish to set.

**Syntax**

```
<NoneType> SetFloat(<FloatType> value)
```

## SetName

**Explanation**

Set the (external) name.

**Arguments**

Enter the desired name.

**Syntax**

```
<NoneType> SetName(<StringType> Name)
```

## SetString

**Explanation**

Set the string value.  In this version of Poser, this is the path of a texture or movie file.

**Arguments**

Enter the path name for the desired texture or movie file.

**Syntax**

```
<NoneType> SetString(<StringValue> file)
```

## Type

**Explanation**

Get the type of data accepted by the current input.  The types are defined as Poser member variables such as poser.kNodeInputCodeCOLOR.

**Arguments**

None

**Syntax**

```
<IntType> Type()
```

## Value

**Explanation**

Get the current value at the selected input. Depending on the type of input, the return value can be a float, a tuple of three floats, or a string.

**Arguments**

None

**Syntax**

```
<FloatType> Value()
(<FloatType>, <FloatType>, <FloatType>) Value()
<StringType> Value()
```

# FireFly Options Methods

This class of methods was introduced in Poser 6.0.0.

## AutoValue

**Explanation**

Get the value of the automatic render settings slider.

**Arguments**

None

**Syntax**

```
<IntType> AutoValue()
```

## BucketSize

**Explanation**

Get the bucket size.

**Arguments**

None

**Syntax**

```
<IntType> BucketSize()
```

## DepthOfField

**Explanation**

Query whether or not depth of field is enabled.  A return value of 1 indicates that depth of field is enabled; a return value of 0 indicates that depth of field is disabled.

**Arguments**

None

**Syntax**

```
<IntType> DepthOfField()
```

## Displacement

**Explanation**

Query whether or not displacement is enabled.  A return value of 1 indicates that displacement is enabled; a return value of 0 indicates that displacement is disabled.

**Arguments**

None

**Syntax**

```
<IntType> Displacement()
```

## DisplacementBounds

**Explanation**

Get the size of the displacement bounds.

**Arguments**

None

**Syntax**

```
<FloatType> DisplacementBounds()
```

## DrawToonOutline

**Explanation**

Query whether or not toon outlines are being drawn.  A return value of 1 indicates that drawing is enabled; a return value of 0 indicates that toon outlines are turned off.

**Arguments**

None

**Syntax**

```
<IntType> DrawToonOutline()
```

## FilterSize

**Explanation**

Get the post filter size.

**Arguments**

None

**Syntax**

```
<IntType> FilterSize()
```

## FilterType

**Explanation**

Get the post filter type.

**Arguments**

None

**Syntax**

```
<IntType> FilterType()
```

## GIINtensity

**Explanation**

Get the intensity of indirect light. A value of 0 indicates that global illumination is turned off. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

None

**Syntax**

```
<FloatType> GIIntensity()
```

## GIMaxError

**Explanation**

```
Get the Max Error of the irradiance cache. Valid values are in the range between
0 and 1.
```

**Arguments**

None

**Syntax**

```
<FloatType> GIMaxError()
```

## GIOnlyRender

**Explanation**

Queries whether or not only indirect light is being rendered. A return value of 1 stands for indirect light only, 0 indicates a regular render.

**Arguments**

None

**Syntax**

```
<IntType> GIOnlyRender()
```

## GINumBounces

**Explanation**

Get the number of bounces used for indirect light, Higher values result in better quality and longer render times. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

None

**Syntax**

```
<IntType> GINumBounces()
```

## GINumSamples

**Explanation**

Get the number of rays used for global illumination, Higher values result in better quality and longer render times. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

None

**Syntax**

```
<IntType> GINumSamples()
```

## LoadPreset

**Explanation**

Load options from a render preset (.prp file).

**Arguments**

Specify the full path for the preset file.

**Syntax**

```
<NoneType> LoadPreset(<StringType> presetFilePath)
```

## Manual

**Explanation**

Query whether manual render settings apply.  A return value of 1 indicates that manual settings apply; a return value of 0 indicates that automatic settings apply.

**Arguments**

None

**Syntax**

```
<IntType> Manual()
```

## MaxError

**Explanation**

Get the Maximum Error of the occlusion cache.  Valid values are in the range between 0 and 1.

**Arguments**

None

**Syntax**

```
<FloatType> MaxError()
```

## MaxRayDepth

**Explanation**

Get the maximum number of raytrace bounces.

**Arguments**

None

**Syntax**

```
<IntType> MaxRayDepth()
```

## MaxSampleSize

**Explanation**

Get the maximum distance between two irradiance samples.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

None

**Syntax**

```
<FloatType> MaxSampleSize()
```

## MaxTextureRes

**Explanation**

Get the max texture resolution.

**Arguments**

None

**Syntax**

```
<IntType> MaxTextureRes()
```

## MinShadingRate

**Explanation**

Get the minimum shading rate.

**Arguments**

None

**Syntax**

```
<FloatType> MinShadingRate()
```

## MotionBlur

**Explanation**

Query whether or not motion blur is enabled.  A return value of 1 indicates that motion blur is enabled; a return value of 0 indicates that motion blur is disabled.

**Arguments**

None

**Syntax**

```
<IntType> MotionBlur()
```

## PixelSamples

**Explanation**

Get the number of samples per pixel.

**Arguments**

None

**Syntax**

```
<IntType> PixelSamples()
```

## RayAccelerator

**Explanation**

Get the current ray accelerator. Return value is a constant such as kRayAcceleratorCodeKDTREE (see FireFly Options Codes section for more possible return values). Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

None

**Syntax**

```
<IntType> RayAccelerator()
```

## RayTracing

**Explanation**

Query whether or not raytracing is enabled.  A return value of 1 indicates that raytracing is enabled; a return value of 0 indicates that raytracing is disabled.

**Arguments**

None

**Syntax**

```
<IntType> RayTracing()
```

## RemoveBackfacing

**Explanation**

Query whether or not remove backfacing polygons is enabled.  A return value of 1 indicates that backfacing polygons will be removed; a return value of 0 indicates that they will not be removed.

**Arguments**

None

**Syntax**

```
<IntType> RemoveBackfacing()
```

## SavePreset

**Explanation**

Save the current render options to a preset (.prp file).

**Arguments**

Specify the full path for the new .prp file.

**Syntax**

```
<NoneType> SavePreset(<StringType> presetFilePath)
```

## SetAutoValue

**Explanation**

Set the value of the automatic render settings slider.  Values from 0 to 8 (inclusive) are valid.

**Arguments**

Specify the slider value as an integer between 0 and 8 (inclusive).

**Syntax**

```
<NoneType> SetAutoValue(<IntType> value)
```

## SetBucketSize

**Explanation**

Set the bucket size.

**Arguments**

Enter the bucket size value.

**Syntax**

```
<NoneType> BucketSize(<IntType> size)
```

## SetDepthOfField

**Explanation**

Set whether depth of field is enabled.  A value of 1 enables depth of field, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetDepthOfField(<IntType> depthoffield)
```

## SetDisplacement

**Explanation**

Set whether displacement is enabled.  A value of 1 enables displacement, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetDisplacement(<IntType> displacement)
```

## SetDisplacementBounds

**Explanation**

Set the size of the displacement bounds.

**Arguments**

Enter a floating-point value that represents the displacement bounds.

**Syntax**

```
<NoneType> SetDisplacmentBounds(<FloatType> bounds)
```

## SetDrawToonOutline

**Explanation**

Set whether toon outlines are being drawn.  A value of 1 enables toon outlines, and 0 disables them.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetDrawToonOutline(<IntType> drawoutlines)
```

## SetFilterSize

**Explanation**

Set the post filter size.

**Arguments**

Enter an integer value to represent the post filter size.

**Syntax**

```
<NoneType> SetFilterSize(<IntType> size)
```

## SetFilterType

**Explanation**

Set the post filter type.

**Arguments**

Enter the constant defined for the desired post filter type.

**Syntax**

```
<NoneType> SetFilterType(<IntType> type)
```

## SetGIIntensity

**Explanation**

Set the intensity of indirect light.  A value of 0 turns off global illumination.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

Specify the indirect light intensity as a floating-point number.

**Syntax**

```
<NoneType> SetGIIntensity(<FloatType> intensity)
```

## SetGIMaxError :

**Explanation**

Set the Max Error of the irradiance cache. Valid values are in the range between 0 and 1.

**Arguments**

Enter a valid value to represent the maximum acceptable error.

### Syntax

```
<NoneType> SetGIMaxError(<FloatType> maxError)
```

## SetGINumBounces

### Explanation

Set the number of bounces for indirect light.  Higher values result in better quality and longer render times.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

### Arguments

Enter an integer value to represent the number of bounces.

### Syntax

```
<NoneType> SetGINumBounces(<IntType> bounces)
```

## SetGINumSamples

### Explanation

Set the number of rays used for global illumination.  Higher values result in better quality and longer render times.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

### Arguments

Enter an integer value to represent the number of rays.

### Syntax

```
<NoneType> SetGINumSamples(<IntType> samples)
```

## SetGIOnlyRender :

**Explanation**

Set if only indirect light are being rendered. A value of 0 enables regular renders, 1 enables indirect light only rendering.

**Arguments**

Enter either 0 for regular rendering, or 1 for indirect light only rendering.

**Syntax**

```
<NoneType> SetGIOnlyRender(<IntType> gionly)
```

## SetManual

**Explanation**

Set whether manual render settings should apply.  Enter a value of 1 for manual settings, or 0 for automatic settings.

**Arguments**

Enter either 0 or 1 to specify automatic or manual settings.

**Syntax**

```
<NoneType> SetManual(<IntType> manual)
```

## SetMaxError

**Explanation**

Set the Maximum Error of the occlusion cache.  Valid values are in the range between 0 and 1.

**Arguments**

Specify the Maximum Error as a floating-point number between 0 and 1.

**Syntax**

```
<NoneType> SetMaxError(<FloatType> maxError)
```

## SetMaxRayDepth

**Explanation**

Set the maximum number of raytrace bounces.

**Arguments**

Enter the desired maximum number.

**Syntax**

```
<NoneType> SetMaxRayDepth(<IntType> depth)
```

## SetMaxSampleSize

**Explanation**

**Set the maximum distance between two irradiance samples.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availabiArguments**

Specify the maximum distance as a floating-point number.

**Syntax**

```
<NoneType> SetMaxSampleSize(<FloatType> maxSize)
```

## SetMaxTextureRes

**Explanation**

Set the maximum texture resolution.

**Arguments**

Specify the maximum x and y resolution at which Poser will load figures.  Both x and y share a single value.

**Syntax**

```
<NoneType> SetMaxTextureRes(<IntType> resolution)
```

## SetMinShadingRate

**Explanation**

Set the minimum shading rate.

**Arguments**

Specify the desired minimum shading rate.

**Syntax**

```
<NoneType> SetMinShadingRate(<FloatType> shadingrate)
```

## SetMotionBlur

**Explanation**

Set whether motion blur is enabled.  A value of 1 enables motion blur, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetMotionBlur(<IntType> enabled)
```

## SetPixelSamples

**Explanation**

Set the number of samples per pixel.

**Arguments**

Enter the desired sample value.

**Syntax**

```
<NoneType> SetPixelSamples(<IntType> numsamples)
```

## SetRayAccelerator

**Explanation**

Set the ray accelerator. The value should be a constant such as kRayAcceleratorCodeKDTREE (see FireFly Options Codes for more possible ray accelerator constants). Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability. Use at your own risk.

**Arguments**

Specify the ray accelerator constant.

**Syntax**

```
<NoneType> SetRayAccelerator(<IntType> acceleratorType)
```

## SetRayTracing

**Explanation**

Set whether raytracing is enabled. A value of 1 enables raytracing, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetRayTracing(<IntType> raytracing)
```

## SetRemoveBackfacing

**Explanation**

Set whether remove backfacing polygons is enabled. A value of 1 specifies that backfacing polygons will be removed; a value of 0 specifies that they will not be removed.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetRemoveBackfacing(<IntType> enabled)
```

## SetShadowOnlyRender

**Explanation**

Set whether only shadows are being rendered. A value of 1 enables regular renders, and 0 enables shadow only renders.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetShadowsOnlyRender(<IntType> shadowsonly)
```

## SetShadows

**Explanation**

Set whether shadows are being rendered. A value of 1 enables shadow rendering, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetShadows(<IntType> doshadows)
```

## SetSmoothPolys

**Explanation**

Set whether polygons are being smoothed. A value of 1 renders polygons as smooth surfaces, and 0 renders polygons as flat surfaces. Note that this can be set per actor; see the description of Actor Type.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetSmoothPolys(<IntType> dosmoothing)
```

## SetTextureCacheCompression

**Explanation**

Set the compression scheme FireFly uses for disk-based textures.  The return value is a constant such as kTextureCompressorCodeZIP (see FireFly Options Codes for more possible compression codes).  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

Specify the constant for the desired compression scheme.

**Syntax**

```
<NoneType> SetTextureCacheCompression(<IntType> compression)
```

## SetTextureCacheSize

**Explanation**

Set the texture cache size (in KB).  This setting determines how much RAM FireFly will reserve to cache disk-based textures.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.  Use at your own risk.

**Arguments**

Enter an integer to represent the cache size in KB.

**Syntax**

```
<NoneType> SetTextureCacheSize(<IntType> cacheSize)
```

## SetTextureFiltering

**Explanation**

Set whether texture filtering is enabled.  A value of 1 enables texture filtering, and 0 disables it.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetTextureFiltering(<IntType> filtering)
```

## SetToneExposure :

**Explanation**

Set the tone mapping exposure.

**Arguments**

Enter exposure as floating point value.

**Syntax**

```
<NoneType> SetToneExposure(<FloatType> exposure)
```

## SetToneGain :

**Explanation**

Set the tone mapping gain.

**Arguments**

Enter gain as floating point value.

**Syntax**

```
<NoneType> SetToneGain(<FloatType> gain)
```

## SetToneMapper :

**Explanation**

Set the tone mapping operator. Poser supports 0 = no tone mapping and 1 = exponential tone mapping.

**Arguments**

Enter 0 to disable tone mapping, and 1 to enable exponential tone mapping.

**Syntax**

```
<IntType> ToneMapper()
```

## SetToonOutlineStyle

**Explanation**

Set the toon outline style.

**Arguments**

Enter a constant representing the desired toon outline style.

**Syntax**

```
<NoneType> SetToonOutlineStyle(<IntType> outlinestyle)
```

## SetUseOcclusionCulling

**Explanation**

Set whether FireFly performs occlusion culling to improve performance.  A value of 1 enables occlusion culling; a value of 0 disables it.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetUseOcclusionCulling(<IntType> useOcclusionCulling)
```

## SetUseTextureCache

**Explanation**

Set whether FireFly uses cached disk-based textures instead of loading the entire texture into RAM.  A value of 1 enables texture caching; a value of 0 disables it.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetUseTextureCache(<IntType> useCache)
```

## ShadowOnlyRender

**Explanation**

Queries whether or not only shadows are being rendered.  A return value of 1 indicates a shadows only render, and a value of 0 indicates a regular render.

**Arguments**

None

**Syntax**

```
<IntType> ShadowOnlyRender()
```

## Shadows

**Explanation**

Queries whether or not shadows are being rendered.  A return value of 1 indicates that shadows are being rendered, and a value of 0 indicates shadow rendering is disabled.

**Arguments**

None

**Syntax**

```
<IntType> Shadows()
```

## SmoothPolys

**Explanation**

Queries whether or not polygons are being smoothed.  A value of 1 indicates polygons are being smoothed, and a value of 0 indicates that polygons are being rendered as flat surfaces.  Note that this can be set per actor; see the description of Actor Type.

#### Arguments

None

#### Syntax

```
<IntType> SmoothPolys()
```

## TextureCacheCompression

#### Explanation

Get the compression scheme FireFly uses for disk-based textures.  The return value will be a constant such as kTextureCompressorCodeZIP (see FireFly Options Codes for more possible compression schemes).  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

#### Arguments

None

#### Syntax

```
<IntType> TextureCacheCompression()
```

## TextureCacheSize

#### Explanation

Get the texture cache size (in KB).  This value determines how much RAM FireFly will reserve to cache disk-based textures.  Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

#### Arguments

None

**Syntax**

```
<IntType> TextureCacheSize()
```

## TextureFiltering

**Explanation**

Queries whether or not texture filtering is enabled. A return value of 1 indicates that texture filtering is enabled, and a value of 0 indicates texture filtering is disabled.

**Arguments**

None

**Syntax**

```
<IntType> TextureFiltering()
```

## ToneExposure

**Explanation**

Get the tone mapping exposure.

**Arguments**

None

**Syntax**

```
<FloatType> ToneExposure()
```

## ToneGain

**Explanation**

Get the tone mapping gain.

**Arguments**

None

**Syntax**

```
<FloatType> ToneGain()
```

## ToneMapper

**Explanation**

Get the tone mapping operator. Poser 8 supports 0 = no tone mapping and 1 = exponential tone mapping.

**Arguments**

None

**Syntax**

```
<IntType> ToneMapper()
```

## ToonOutlineStyle

**Explanation**

Queries for toon outline style.

**Arguments**

None

### Syntax

```
<IntType> ToonOutlineStyle()
```

## UseOcclusionCulling

### Explanation

Query whether FireFly performs occlusion culling to improve performance. A return value of 1 indicates that occlusion culling is enabled; a return value of 0 indicates that it is disabled. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

### Arguments

None

### Syntax

```
<IntType> UseOcclusionCulling()
```

## UseTextureCache

### Explanation

Query whether FireFly uses cached disk-based textures instead of loading the entire texture into RAM. A return value of 1 indicates that texture caching is enabled; a return value of 0 indicates that it is disabled. Note: This method is not officially supported by Smith Micro Software, and we make no guarantees as to its functionality or its future availability.

### Arguments

None

### Syntax

```
<IntType> UseTextureCache()
```

# Hair Methods

This class of methods was introduced in Poser 6.0.0.

## AirDamping

**Explanation**

Get the air damping value.

**Arguments**

None

**Syntax**

```
<FloatType> AirDamping()
```

## BendResistance

**Explanation**

Get the bend resistance.

**Arguments**

None

**Syntax**

```
<FloatType> BendResistance()
```

## CalculateDynamics

**Explanation**

Calculate this group's hair dynamics.  Note that this may take quite some time, depending on the complexity of the hair group, the scene geometry and the animation length.

**Arguments**

None

**Syntax**

```
<ActorType> CalculateDynamics()
```

## Clumpiness

**Explanation**

Get the clumpiness value.

**Arguments**

None

**Syntax**

```
<FloatType> Clumpiness()
```

## CollisionsOn

**Explanation**

Determine whether this hair group reacts to collisions.  A return value of 1 indicates collision detection is on, and a value of 0 indicates collision detection is off.

**Arguments**

None

**Syntax**

```
<IntType> CollisionsOn()
```

## Delete

**Explanation**

Delete the hair group and its associated hair prop.

**Arguments**

None

**Syntax**

```
<ActorType> Delete()
```

## Density

**Explanation**

Get the density of populated hairs.

**Arguments**

None

**Syntax**

```
<FloatType> Density()
```

## Gravity

**Explanation**

Get the gravity value.

**Arguments**

None

**Syntax**

```
<FloatType> Gravity()
```

## GrowHair

**Explanation**

Grow guide hairs.

**Arguments**

None

**Syntax**

```
<NoneType> GrowHair()
```

## HairProp

**Explanation**

Get the prop that represents this hair group.

**Arguments**

None

**Syntax**

```
<ActorType> HairProp()
```

## KinkDelay

**Explanation**

Get the kink delay value.

**Arguments**

None

**Syntax**

```
<FloatType> KinkDelay()
```

## KinkScale

**Explanation**

Get the kink scale value.

**Arguments**

None

**Syntax**

```
<FloatType> KinkScale()
```

## KinkStrength

**Explanation**

Get the kink strength value.

**Arguments**

None

**Syntax**

```
<FloatType> KinkStrength()
```

## LengthMax

**Explanation**

Get the maximum hair length.

**Arguments**

None

**Syntax**

```
<FloatType> LengthMax()
```

## LengthMin

**Explanation**

Get the minimum hair length.

**Arguments**

none

**Syntax**

```
<FloatType> LengthMin()
```

## Name

**Explanation**

Get the name of this Hair.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## NumbPopHairs

**Explanation**

Get the total number of Hairs.

**Arguments**

None

**Syntax**

```
<IntType> NumbPopHairs()
```

## NumbVertsPerHair

**Explanation**

Get the number of vertices per hair.

**Arguments**

None

**Syntax**

```
<IntType> NumbVertsPerHair()
```

## PositionForce

**Explanation**

Get the internal PositionForce simulation parameter.

**Arguments**

None

**Syntax**

```
<FloatType> PositionForce()
```

## PullBack

**Explanation**

Get the pull back parameter value.

**Arguments**

None

**Syntax**

```
<FloatType> PullBack()
```

## PullDown

**Explanation**

Get the pull down parameter value.

**Arguments**

None

**Syntax**

```
<FloatType> PullDown()
```

## PullLeft

**Explanation**

Get the pull left parameter value.

**Arguments**

None

**Syntax**

```
<FloatType> PullLeft()
```

## RootStiffness

**Explanation**

Get the root stiffness.

**Arguments**

None

**Syntax**

```
<FloatType> RootStiffness()
```

## RootStiffnessFalloff

**Explanation**

Get the root stiffness falloff.

**Arguments**

None

**Syntax**

```
<FloatType> RootStiffnessFalloff()
```

## RootWidth

**Explanation**

Get the hair root width.

**Arguments**

None

**Syntax**

```
<FloatType> RootWidth()
```

## SetAirDamping

**Explanation**

Set the air damping.

**Arguments**

Specify the air damping as a floating-point number.

**Syntax**

```
<NoneType> SetAirDamping(<FloatType> value)
```

## SetBendResistance

**Explanation**

Set the bend resistance.

**Arguments**

Specify the bend resistance as a floating-point number.

**Syntax**

```
<NoneType> SetBendResistance(<FloatType> value)
```

## SetClumpiness

**Explanation**

Set the hair clumpiness.

**Arguments**

Specify the clumpiness as a floating-point number.

**Syntax**

```
<NoneType> SetClumpiness(<FloatType> value)
```

## SetCollisionsOn

**Explanation**

Set whether or not this hair group reacts to collisions.

**Arguments**

Enter 1 to enable collision detection, and 0 to disable it.

**Syntax**

```
<NoneType> SetCollisionsOn(<IntType> value)
```

## SetDensity

**Explanation**

Set the density of populated hairs.

**Arguments**

Specify the hair density as a floating-point number.

**Syntax**

```
<NoneType> SetDensity(<FloatType> value)
```

## SetGravity :

**Explanation**

Set the gravity.

**Arguments**

Specify gravity in g as a floating point number.

**Syntax**

```
<NoneType> SetGravity(<FloatType> value)
```

## SetKinkDelay

**Explanation**

Set the kink delay.

**Arguments**

Specify the kink delay as a floating-point number.

**Syntax**

```
<NoneType> SetKinkDelay(<FloatType> value)
```

## SetKinkScale

**Explanation**

Set the kink scale.

**Arguments**

Specify the kink scale as a floating-point number.

**Syntax**

```
<NoneType> SetKinkScale(<FloatType> value)
```

## SetKinkStrength

**Explanation**

Set the kink strength.

**Arguments**

Specify the kink strength  as a floating-point number.

**Syntax**

```
<NoneType> SetKinkStrength(<FloatType> value)
```

## SetLengthMax

**Explanation**

Set the maximum length.

**Arguments**

Enter the desired maximum length as a floating-point number.

**Syntax**

```
<NoneType> SetLengthMax(<FloatType> value)
```

## SetLengthMin

**Explanation**

Set the minimum length.

**Arguments**

Enter the desired minimum length as a floating-point number.

**Syntax**

```
<NoneType> SetLengthMin(<FloatType> value)
```

## SetName

**Explanation**

Set the name of this Hair.

**Arguments**

Specify the desired name.

**Syntax**

```
<StringType> SetName(<StringType> name)
```

## SetNumbPopHairs

**Explanation**

Set the total number of hairs.

**Arguments**

Enter the total hair number value.

**Syntax**

```
<NoneType> SetNumbPopHairs(<IntType> value)
```

## SetNumbVertsPerHair

**Explanation**

Set the number of vertices per hair.

**Arguments**

Enter the value for the number of vertices.

**Syntax**

```
<NoneType> SetNumbVertsPerHair(<IntType> value)
```

## SetPositionForce

**Explanation**

Set the internal PositionForce simulation parameter.

**Arguments**

Specify the PositionForce value as a floating-point number.

**Syntax**

```
<NoneType> SetPositionForce(<FloatType> value)
```

## SetPullBack

**Explanation**

Set the pull back parameter.

**Arguments**

Specify the pull back value as a floating-point number.

**Syntax**

```
<NoneType> SetPullBack(<FloatType> value)
```

## SetPullDown

**Explanation**

Set the pull down parameter.

**Arguments**

Specify the pull down value as a floating-point number.

**Syntax**

```
<NoneType> SetPullDown(<FloatType> value)
```

## SetPullLeft

**Explanation**

Set the pull left parameter.

**Arguments**

Specify the pull left value as a floating-point number.

**Syntax**

```
<NoneType> SetPullLeft(<FloatType> value)
```

## SetRootStiffness

**Explanation**

Set the root stiffness.

**Arguments**

Specify the root stiffness as a floating-point number.

**Syntax**

```
<NoneType> SetRootStiffness(<FloatType> value)
```

## SetRootStiffnessFalloff

**Explanation**

Set the root stiffness falloff.

**Arguments**

Specify the root stiffness falloff as a floating-point number.

**Syntax**

```
<NoneType> SetRootStiffnessFalloff(<FloatType> value)
```

## SetRootWidth

**Explanation**

Set the hair root width.

**Arguments**

Specify the root width as a floating-point number.

**Syntax**

```
<NoneType> SetRootWidth(<FloatType> value)
```

## SetShowPopulated

**Explanation**

Set whether populated hair is shown.  A value of 1 indicates that it is shown, and 0 indicates that it is not shown.

**Arguments**

Enter a value of either 1 or 0.

**Syntax**

```
<NoneType> SetShowPopulated(<IntType> value)
```

## SetSpringDamping

**Explanation**

Set the spring damping value.

**Arguments**

Specify the spring damping value as a floating-point number.

**Syntax**

```
<NoneType> SetSpringDamping(<FloatType> value)
```

## SetSpringStrength

**Explanation**

Set the spring strength value.

**Arguments**

Specify the spring strength value as a floating-point number.

**Syntax**

```
<NoneType> SetSpringStrength(<FloatType> value)
```

## SetTipWidth

**Explanation**

Set the hair tip width.

**Arguments**

Specify the hair tip width as a floating-point number.

**Syntax**

```
<NoneType> SetTipWidth(<FloatType> value)
```

## ShowPopulated

**Explanation**

Determine whether populated hair is shown. A return value of 1 indicates that it is shown, and a value of 0 indicates that it is not shown.

**Arguments**

None

**Syntax**

```
<IntType> ShowPopulated()
```

## SpringDamping

**Explanation**

Get the spring damping value.

**Arguments**

None

**Syntax**

```
<FloatType> SpringDamping()
```

## SpringStrength

**Explanation**

Get the spring strength value.

**Arguments**

None

**Syntax**

```
<FloatType> SpringStrength()
```

## TipWidth

**Explanation**

Get the hair tip width.

**Arguments**

None

**Syntax**

```
<FloatType> TipWidth()
```

# Cloth Simulator Methods

This class of methods was introduced in Poser 6.0.0.

## AddClothActor

**Explanation**

Add a clothified actor to this simulation.

**Arguments**

Specify the name of the actor you wish to add

**Syntax**

```
<NoneType> AddClothActor(<ActorType> actor)
```

## AddCollisionActor

**Explanation**

Add an actor as a collision object to this simulation.

**Arguments**

Specify the name of the actor you wish to add

**Syntax**

```
<NoneType> AddCollisionActor(<ActorType> actor)
```

## AddCollisionFigure

**Explanation**

Add a figure as a collision object to this simulation, excluding the group names in the list.

**Arguments**

Specify the name of the actor you wish to add, plus the list of group names you wish to exclude.

**Syntax**

```
<NoneType> AddCollisionFigure(<FigureType> figure, <StringType list> excludeList)
```

## CalculateDynamics

**Explanation**

Start the simulation calculation.

**Arguments**

None

**Syntax**

```
<NoneType> CalculateDynamics()
```

## ClearDynamics

**Explanation**

Clear the simulation dynamics cache.

**Arguments**

None

**Syntax**

```
<NoneType> ClearDynamics()
```

## CurrentClothActor

**Explanation**

Get the current cloth actor.

**Arguments**

None

**Syntax**

```
<ActorType> CurrentClothActor()
```

## Delete

**Explanation**

Removes this simulation from the scene.

**Arguments**

None

**Syntax**

```
<NoneType> Delete()
```

## DrapingFrames

**Explanation**

Get the number of draping frames in this simulation.

**Arguments**

None

**Syntax**

```
<IntType> DrapingFrames()
```

## DynamicsProperty

**Explanation**

Get the value of a named property.  Property names are defined as Poser member variables such as poser.kClothParmDENSITY.

**Arguments**

Specify the property for which you want the value.

**Syntax**

## <FloatType> DynamicsProperty(<StringType> NameEndFrame

**Explanation**

Get the end frame of this simulation.

**Arguments**

None

**Syntax**

```
<IntType> EndFrame()
```

## IsClothActor

**Explanation**

Query whether this actor is a cloth actor in this simulation.  A value of 1 indicates that it is a cloth actor, and a value of 0 indicates that it is not a cloth actor.

**Arguments**

Specify the name of the actor.

**Syntax**

```
<IntType> IsClothActor(<ActorType> actor)
```

## IsCollisionActor

**Explanation**

Query whether this actor is a collision actor in this simulation. A value of 1 indicates that it is a collision actor, and a value of 0 indicates that it is not a collision actor.

**Arguments**

Specify the name of the actor.

**Syntax**

```
<IntType> IsCollisionActor(<ActorType> actor)
```

## IsCollisionFigure

**Explanation**

Query whether this actor is a collision figure in this simulation. A value of 1 indicates that it is a collision figure, and a value of 0 indicates that it is not a collision figure.

**Arguments**

Specify the name of the actor.

**Syntax**

```
<IntType> IsCollisionFigure(<FigureType> figure)
```

## Name

**Explanation**

Get the name of the simulation.

**Arguments**

None

**Syntax**

```
<StringType> Name()
```

## RemoveClothActor

**Explanation**

Remove a clothified actor from this simulation.

**Arguments**

Specify the name of the actor to remove.

**Syntax**

```
<NoneType> RemoveClothActor(<ActorType> actor)
```

## RemoveCollisionActor

**Explanation**

Remove this actor as a collision object from this simulation.

**Arguments**

Specify the name of the actor to remove.

**Syntax**

```
<NoneType> RemoveCollisionActor(<ActorType> actor)
```

## RemoveCollisionFigure

**Explanation**

Remove this figure as a collision object from this simulation.

**Arguments**

Specify the name of the figure to remove.

**Syntax**

```
<NoneType> RemoveCollisionFigure(<FigureType> figure)
```

## SetCurrentClothActor

**Explanation**

Set the current cloth actor.

**Arguments**

Specify the actor.

### Syntax

```
<NoneType> SetCurrentClothActor(<ActorType> Actor)
```

## SetDrapingFrames

### Explanation

Set the number of draping frames in this simulation.

### Arguments

Specify the number of frames.

### Syntax

```
<NoneType> SetDrapingFrames(<IntType> frames)
```

## SetDynamicsProperty

### Explanation

Set the value of a named property.

### Arguments

Specify the property name and value.

### Syntax

```
<NoneType> SetDynamicsProperty(<StringType> Name, <FloatType> Value)
```

## SetEndFrame

**Explanation**

Set the end frame of this simulation.

**Arguments**

Enter the frame number.

**Syntax**

```
<NoneType> SetEndFrame(<IntType> frame)
```

## SetName

**Explanation**

Set the name of the simulation.

**Arguments**

Enter the desired name.

**Syntax**

```
<StringType> Name(<StringType> name)
```

## SetStartFrame

**Explanation**

Set the starting frame of this simulation.

**Arguments**

Enter the frame number.

**Syntax**

```
<NoneType> SetStartFrame(<IntType> frame)
```

## StartFrame

**Explanation**

Get the starting frame of this simulation.

**Arguments**

None

**Syntax**

```
<IntType> StartFrame()
```

## DialogSimple Methods

This class of methods was introduced in Poser 6.0.0.

### AskActor

**Explanation**

Ask the user to select an actor.

**Arguments**

Enter the request message.

**Syntax**

```
<NoneType> AskActor(<StringType> message)
```

## AskFloat

**Explanation**

Ask the user for a floating-point number.

**Arguments**

Enter the request message.

**Syntax**

```
<FloatType> AskFloat(<StringType> message)
```

## AskInt

**Explanation**

Ask the user for an integer value.

**Arguments**

Enter the request message.

**Syntax**

```
<FloatType> AskInt(<StringType> message)
```

## AskMenu

**Explanation**

Ask the user to select an item in a menu.

**Arguments**

Enter the menu title, the request message, and each of the subsequent items in the menu.

### Syntax

```
<StringType> AskMenu(<StringType> title, <StringType> message, <StringType>
item1, <StringType> item2, ...)
```

## DialogSimple

### Explanation

Creates an object of the DialogSimple class type – in other words, a simple dialog.

### Arguments

none

### Syntax

```
<NoneType> DialogSimple()
```

## MessageBox

### Explanation

Show a message box with the message and an OK button.

### Arguments

Enter the message.

### Syntax

```
<NoneType> MessageBox(<StringType> message)
```

### PickImage

**Explanation**

Bring up the Texture Manager and let the user pick an image for this input.

**Arguments**

Specify the input to which the new image node will be attached.

**Syntax**

```
<NoneType> PickImage(<ShaderNodeInputType> inputInput)
```

### YesNo

**Explanation**

Show a dialog with the message and a Yes and a No button.  The function returns 1 if the user clicks Yes, and 0 if the user clicks No.

**Arguments**

Enter the message.

**Syntax**

```
<IntType> YesNo(<StringType> message)
```

## Dialog Methods

This class of methods was introduced in Poser 7.0.0.

## AddButtonCallback

### Explanation

Assigns a method callback function to a button click.

### Arguments

Enter the button to which you wish to assign a callback, and the function you wish to call when the button is clicked.

### Syntax

```
<NoneType> AddButtonCallback(<StringType> buttonName, <FunctionType> function)
```

## Dialog

### Explanation

Implements a message dialog callable from Poser's Python interface.

### Arguments

Enter the path for the of the XML file that defines the dialog layout, the title of the dialog, the message the dialog should contain, and the height and width of the dialog.

### Syntax

```
<DialogType> Dialog(<StringType> layoutXMLPath, <StringType> title, <StringType>
message, <IntType> width, <IntType> height)
```

## SetButtonValue

### Explanation

Specify a numerical value for a button's label.

**Arguments**

Specify the name of the buttona nd the value with which you wish to label it.

**Syntax**

```
<NoneType> SetButtonValue(<StringType> buttonName, <IntType> value)
```

### SetText

**Explanation**

Specify the text for a dialog widget.

**Arguments**

Specify the widget name and the text you wish to accompany the widget.

**Syntax**

```
<NoneType> SetText(<StringType> widgetName, <StringType> text)
```

## DialogFileChooser Methods

This class of methods was introduced in Poser 7.0.0.

### DialogFileChooser

**Explanation**

Implements a file chooser callable from Poser's Python interface.

**Arguments**

This method requires 4 Arguments:

- Type: Enter a Poser Dialog constant specifying either a File Open or File Save dialog (such as kDialogFileChooserOpen).
- Parent: Specify the window to which the file chooser will be parented.
- Message: Enter the message to be displayed in the dialog.
- Start Directory: Specify the file that will be selected by default in the dialog.

**Syntax**

```
<DialogFileChooserType> DialogFileChooser(<IntType> type, <DialogType>
parentDialog, <StringType> message, <StringType> startDir)
```

## Path

**Explanation**

Get the path specified by the user in the dialog.

**Arguments**

None

**Syntax**

```
<StringType> Path()
```

## Show

**Explanation**

Brings up the File Chooser modal dialog.

**Arguments**

None

**Syntax**

```
<NoneType> Show()
```

## DialogDirChooser Methods

This class of methods was introduced in Poser 7.0.0.

### DialogDirChooser

**Explanation**

Implements a directory chooser callable from Poser's Python interface.

**Arguments**

Specify the window to which the dialog will be parented, the specific message text, and the directory that will be selected by default in the dialog.

**Syntax**

```
<DialogDirChooserType> DialogDirChooser(<DialogType> parentDialog, <StringType>
message, <StringType> startDir)
```

### Path

**Explanation**

Get the path specified by the user in the dialog.

**Arguments**

None

**Syntax**

```
<StringType> Path()
```

### Show

**Explanation**

Brings up the Directory Chooser dialog.

**Arguments**

None

**Syntax**

```
<NoneType> Show()
```

## DialogTextEntry Methods

This class of methods was introduced in Poser 7.0.0.

### DialogTextEntry

**Explanation**

Implements a simple (one-field) text entry dialog.

**Arguments**

Specify the window to which the dialog will be parented, and the message to be displayed in the dialog.

**Syntax**

```
<DialogTextEntry> DialogTextEntry(<DialogType> parentDialog, <StringType>
message)
```

## Show

**Explanation**

Brings up a text entry dialog.

**Arguments**

None

**Syntax**

```
<NoneType> Show()
```

## Text

**Explanation**

Get the text entered by the user in the dialog.

**Arguments**

None

**Syntax**

```
<StringType> Text()
```

# Index