

RUNNING PYTHON SCRIPTS **IN POSER**

Tutorial Version 1.2

Content

| | |
|-------------------------------------------------------------|---|
| 1. How to Run a Python Script in Poser..... | 1 |
| 2. How to Configure the Python Scripts Menu Palette..... | 1 |
| 3. How to Run Scripts from the Wacros Palette..... | 3 |
| 4. How to Run Scripts from the Scripts Menu..... | 3 |
| 5. How to Start Python Scripts together with Poser..... | 3 |
| 6. How to Run Python Scripts from a Poser Content File..... | 4 |

Copyright

This tutorial is © Copyright 2009-11 by Ralf Sessler. All rights reserved.

Disclaimer

There is no warranty in using this tutorial.

Ralf Sessler

Dimension 3D

E-Mail: d3d@sesseler.de

Internet: d3d.sesseler.de



1. How to Run a Python Script in Poser

The *Poser Python* scripting language allows to define additional functionality for Poser. It provides an interface to access and modify a large number of elements in a Poser scene controlled by the Python scripting language. Python scripts are text files with the extension *py* or compiled files with the extension *pyc*. They should be located in the main Runtime below the *Python\poserscripts* folder, for some older Poser versions Python scripts will not run otherwise.

There are several ways to run a Python script in Poser. The first one is to use *Run Python Script...* from the *File* menu. You locate the Python file in the file dialog and Poser executes it. This is simple but not practical if you want to use a script often. Also, in older Poser versions you have to change the file filter to have either uncompiled or compiled scripts listed.

The second way is the *Python Scripts* palette that is opened by *Python Scripts* from the *Window* menu. It has place for ten scripts, but you can configure it for any number of scripts with submenus (see 2). Poser 6 and up has a *Wacros* palette for ten Python scripts in the material room with a menu for user defined *Wacro* scripts (see 3). Poser 7 and up has a *Scripts* menu in the main menu bar (see 4). You can also start Python scripts together with Poser (see 5). Finally, you can run Python scripts from any Poser library file. E.g. you can define poses that contain one script call and then run the Python scripts from the pose library (see 6).

2. How to Configure the Python Scripts Menu Palette

The initial setting for the Python Scripts palette is defined in the Python script *mainButtons.py* that is located in *Runtime\Python\poserScripts*. When you open it in a text editor, you get something like this:

```
import poser

poser.DefineScriptButton(1, "...:CreateProps:propButtons.py", "Prop Samples")
poser.DefineScriptButton(2, "...:GeomMods:geomModButtons.py", "Geom Mods")
poser.DefineScriptButton(3, "...:Utility:utilityButtons.py", "Utility Funcs")
poser.DefineScriptButton(4, "...:SampleCallbacks:callbackButtons.py", "Sample Callbacks")
poser.DefineScriptButton(5, "...:RenderControl:renderButtons.py", "Render / IO")
poser.DefineScriptButton(6, "...:MaterialMods:materialModButtons.py", "Material Mods")
poser.DefineScriptButton(7, "...:PrintInfo:printInfoButtons.py", "Print Info")
poser.DefineScriptButton(8, "", "...")
poser.DefineScriptButton(9, "", "...")
poser.DefineScriptButton(10, "", "...")
```

(I replaced *:Runtime:Python:PoserScripts* by *..* to fit each command in one line.)

This looks like a list of menu buttons with a script and a label for each button. In fact, it is a Python script that assigns a pair script/label to each button of the Python Scripts palette, but to write your own menus, it's easier to just think of it as a list of menu buttons.

Because it is not just a list, but a Python script itself, you can use a file with such a list again as a script in a button list. When selecting such a button, it will assign new buttons to the *Python Scripts* palette. Indeed, all of the Python scripts in the original menu shown above are this kind of submenu that only change the buttons in the Python Scripts palette.

Creating Your Own Menus

If you want to create your own menu hierarchy of Python buttons, at best make a copy of the actual *mainButtons.py* first, maybe rename it to *poserButtons.py*.

Now, make a list of which Python scripts you want to access from the *Python Scripts* palette and how you want to arrange them. There is one *mainButtons.py* that defines the initial buttons, and you can define any number of additional lists with buttons. The only restriction is that you can have only up to 10 entries in each list and that you need some of the entries to navigate between the lists. Of course, all lists must be interconnected somehow and there must be a way to reach them from *mainButtons.py*.

Let me give you a simple example. Say you have 4 scripts that you use often and 8 more scripts. Finally, you want to have access to the original Poser scripts. Your *menuButtons.py* would look like this:

```
poser.DefineScriptButton(1, "..:script1.py", "Script 1")
poser.DefineScriptButton(2, "..:script2.py", "Script 2")
poser.DefineScriptButton(3, "..:script3.py", "Script 3")
poser.DefineScriptButton(4, "..:script4.py", "Script 4")
poser.DefineScriptButton(5, "", "...")
poser.DefineScriptButton(6, "", "...")
poser.DefineScriptButton(7, "", "...")
poser.DefineScriptButton(8, "", "...")
poser.DefineScriptButton(9, "..:moreButtons.py", ">> More Scripts")
poser.DefineScriptButton(10, "..:poserButtons.py", ">> Poser Scripts")
```

The additional scripts are in *moreButtons.py*:

```
poser.DefineScriptButton(1, "..:script5.py", "Script 5")
poser.DefineScriptButton(2, "..:script6.py", "Script 6")
poser.DefineScriptButton(3, "..:script7.py", "Script 7")
poser.DefineScriptButton(4, "..:script8.py", "Script 8")
poser.DefineScriptButton(5, "..:script9.py", "Script 9")
poser.DefineScriptButton(6, "..:script10.py", "Script 10")
poser.DefineScriptButton(7, "..:script11.py", "Script 11")
poser.DefineScriptButton(8, "..:script12.py", "Script 12")
poser.DefineScriptButton(9, "", "...")
poser.DefineScriptButton(10, "..:moreButtons.py", "<< Main")
```

I used >> and << in front of those button labels that are used to navigate between the menus.

As you can see, you can create arbitrary complex menu hierarchies in this way. You can have more than one submenu like *moreButtons.py* in *mainButtons.py*, and you can have further submenus also in those submenus. Just remember to have a path to each of your menu button lists from the initial *mainButtons.py* and to always include a button to go back to the previous list or to the top list.

3. How to Run Scripts from the Wacros Palette

A *Wacro* is a Python script that is applied to the actual material in the material room or to all materials of the current object. The material room has its own palette with buttons for Python scripts. These buttons are defined in *mainWacros.py* in *Runtime\Python\poserScripts\Wacros*. Again, this is a Python script that contains a kind of list of button definitions. The only difference is that *DefineMaterialWacroButton* is used instead of *DefineScriptButton*. You can define the same kind of menu structure as for the *Python Scripts* palette (see 2). Of course, you can use any kind of Python scripts here, not just Wacros, but you should not run Wacros outside of the material room.

An alternative way to access Python scripts in the material room is the *User Defined* menu located below the Wacro buttons. It lists all Python scripts that are located in the folder *Runtime\Python\poserScripts\Wacros\UserDefined*.

4. How to Run Scripts from the Scripts Menu

All Python scripts that are located in the folder *Runtime\Python\poserScripts\ScriptsMenu* or any sub-folder are shown in the *Scripts* menu of the main menu bar of Poser 7. You can start the script simply by selecting it from the menu.

Note: You have to restart Poser to update the *Scripts* menu after you added or removed scripts in this folder.

5. How to Start Python Scripts together with Poser

If you want to run a Python script each time when Poser starts, you can do so from the file *poserStartup.py*, which is located in *Runtime\Python\poserScripts*. This is a Python script that is executed once while Poser is starting.

To start a Python script from *poserStartUp.py*, add the following line to this file:

```
poser.ExecFile(" :Runtime:Python:poserScripts:script.py")
```

The path may be an absolute path (use `\\` instead of `\` as path separator in that case) or a relative path starting with `:Runtime` as in the example above.

However, Poser may not yet be fully initialized when this start-up script runs, and some scripts may fail because of this. In that case, you may start a script with a delay for a few seconds. In Poser 8 and up, this can be done as follows:

```

import wx

class Delay(wx.Timer):
    def __init__(self, ms):
        wx.Timer.__init__(self)
        self.Start(ms, True)
    def Notify(self):
        poser.ExecFile(":Runtime:Python:poserScripts:script.py")
        global delay
        del delay

delay = Delay(1000)

```

The 1000 in Delay(1000) is the number of milliseconds for the delay. Depending on your system and the script, other delay times may be reasonable.

6. How to Run Python Scripts from a Poser Content File

You can even include Python scripts in any kind of Poser content file. Just add the line

```
runPythonScript ...:script.py
```

to your Poser file at the top level, i.e. not inside of any block with braces. The script will be executed when the file is loaded from the library. If you define Poser files that contain nothing but this kind of script call, you can access Python scripts from the library.

```

{
    version
    {
        number 4
    }
    runPythonScript ...:script.py
}

```

Note: For pose files, you have to add the *runPythonScript* line to the second part of the pose, or it will be executed twice.

```

{
    version
    {
        number 4
    }
}
{
    version
    {
        number 4
    }
    runPythonScript ...:script.py
}

```

Startup Scripts

For scenes and figures, you can define a startup script. In a scene, the script is called when the scene is loaded. In a figure, the script is called when the figure is added to the scene, but also when a scene that contains the figure is loaded. The following line has to be added to the end of the *doc* part of a scene or the *figure* part of a figure to be used as startup script:

```
pythonStartupScript 0 0 ...:script.py
```

(I don't know what the 0 0 is for, but it must be there.)