FreeCAD Articles
Scritping Intro

Carlo Dormeletti

Version: 1.0

FreeCAD reference version: 0.21

Contents

1 Python Console Output

One of the most daunting thing is how to ”translate” **Python Console** output to a proper python script

Let’s see to start some output you could find in the Python Console when you perform some operations.

Create a new Document

```
### Begin command Std_New
# App.setActiveDocument("Unnamed")
# App.ActiveDocument=App.getDocument("Unnamed")
# Gui.ActiveDocument=Gui.getDocument("Unnamed")
# Gui.activeDocument().activeView().viewDefaultOrientation()
# Gui.ActiveDocument.ActiveView.setAxisCross(True)
### End command Std_New
# Gui.runCommand('Std_OrthographicCamera',1)
```

Activate Part WB

```
### Begin command Std_Workbench
# Gui.activateWorkbench("PartWorkbench")
### End command Std_Workbench
```

Create a Box in Part WB

```
### Begin command Part_Box
App.ActiveDocument.addObject("Part::Box","Box")
App.ActiveDocument.ActiveObject.Label = "Cube"
# Object created at document root.
App.ActiveDocument.recompute()
# Gui.SendMsgToActiveView("ViewFit")
### End command Part_Box
```

We could make some considerations:

- Many lines are commented out.
- There are some rationale in the commands, and we could note that there are some ”stanzas” enclosed in ### **Begin** and ### **End** comments.
- There are some regularities as many if not most of the lines are beginning with **App.** or **Gui.**

But how to make a proper script is not immediately visible.

First of all we have to make a translation between the namespace used by FreeCAD Python Console and the proper calls.

```
import FreeCAD as App
import FreeCADGui as Gui
```

Will make one of the possible translation that is a start to make a proper standalone script.

or simply remember that when you see **App** you should prepend the command with **FreeCAD** and when you see **Gui** you should prepend the command with **FreeCADGui**.

Let's examine the first "stanza":

```
### Begin command Std_New
# App.setActiveDocument("Unnamed")
# App.ActiveDocument=App.getDocument("Unnamed")
# Gui.ActiveDocument=Gui.getDocument("Unnamed")
# Gui.activeDocument().activeView().viewDefaultOrientation()
# Gui.ActiveDocument.ActiveView.setAxisCross(True)
### End command Std_New
# Gui.runCommand('Std_OrthographicCamera',1)
```

you could think that is enough to make this modifications:

```
import FreeCAD as App
import FreeCADGui as Gui

App.setActiveDocument("Unnamed")
App.ActiveDocument = App.getDocument("Unnamed")
Gui.ActiveDocument = Gui.getDocument("Unnamed")
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)
```

to have a new document created, but a simple test will reveal that is generating this error

```
Traceback (most recent call last):
  File "test1.py", line 4, in <module>
    App.setActiveDocument("Unnamed")
<class 'RuntimeError'>: Try to activate unknown document 'Unnamed
                    ↪ '
```

So it is almost clear that something is missing in the Python Console output that should be here.

In this case the remedy is to add a line.

```
import FreeCAD as App
import FreeCADGui as Gui

App.newDocument("Unnamed")

App.setActiveDocument("Unnamed")
App.ActiveDocument = App.getDocument("Unnamed")
Gui.ActiveDocument = Gui.getDocument("Unnamed")
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)
```

And everything is working as expected, a new document named ”Unnamed” is created, but this writing is not very Pythonic this writing is slightly better:

```
import FreeCAD as App
import FreeCADGui as Gui

doc_name = "mydocument"

App.newDocument(doc_name)

App.setActiveDocument(doc_name)
App.ActiveDocument = App.getDocument(doc_name)
Gui.ActiveDocument = Gui.getDocument(doc_name)
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)
```

And this is adding some flexibility when changing the new document name, it suffice to change it in only one place.

There are many other thing to note, but this is an introductory text so we must keep things short as much as possible.

we could expand the example adding:

```
import FreeCAD as App
import FreeCADGui as Gui

doc_name = "mydocument"

App.newDocument(doc_name)

App.setActiveDocument(doc_name)
App.ActiveDocument = App.getDocument(doc_name)
Gui.ActiveDocument = Gui.getDocument(doc_name)
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)

App.ActiveDocument.addObject("Part::Box","Box")
App.ActiveDocument.ActiveObject.Label = "Cube"
# Object created at document root.
App.ActiveDocument.recompute()
# Gui.SendMsgToActiveView("ViewFit")
```

And you effectively obtain a Cube, without too much hassle.

If you decomment the last line you will have the cube adapted to the 3d window.

But as said it is not very pythonic and a simple modification will make things more pythonic and hopefully more readable:

```
import FreeCAD as App
import FreeCADGui as Gui
```

```
doc_name = "mydocument"

App.newDocument(doc_name)

work_doc = App.getDocument(doc_name)

App.setActiveDocument(doc_name)
App.ActiveDocument = work_doc
Gui.ActiveDocument = Gui.getDocument(doc_name)
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)

box1 = work_doc.addObject("Part::Box","myBox")
box1.Label = "myCube"
# Object created at document root.
work_doc.recompute()
Gui.SendMsgToActiveView("ViewFit")
```

This will make it more pythonic, note that in this case you can change both the **Name** and the **Label** attributes of the created box to fit your needs

A side note is that you modify as example the **Length** property of the Box you will see in Python Console this line:

```
FreeCAD.getDocument('mydocument').getObject('myBox').Length = '11
                    ↪  mm'
```

We could make the some considerations, as FreeCAD in this case was telling us:

```
FreeCAD.getDocument('mydocument')
```

Is the proper way to refer to a document, not very different from that we have done when assigning the object to the variable **work_doc**.

```
FreeCAD.getDocument('mydocument').getObject('myBox')
```

This was telling us that the object "mydocument" hold some Objects and that we could with the function **getObject()** retrieve them by name in our case what we obtain is the same ting we have called **box1**.

So we could easily assign ours desired lengths to the box writing:

```
import FreeCAD as App
import FreeCADGui as Gui

doc_name = "mydocument"

App.newDocument(doc_name)

work_doc = App.getDocument(doc_name)
```

```
App.setActiveDocument(doc_name)
App.ActiveDocument = work_doc
Gui.ActiveDocument = Gui.getDocument(doc_name)
Gui.activeDocument().activeView().viewDefaultOrientation()
Gui.ActiveDocument.ActiveView.setAxisCross(True)

box1 = work_doc.addObject("Part::Box","myBox")
box1.Label = "myCube"
box1.Length = '2 in'
box1.Width = 30
box1.Height = 15

# Object created at document root.
work_doc.recompute()
Gui.SendMsgToActiveView("ViewFit")
```

A little note is that we could specify a value or an expression like **'2 in'** and this is accepted as valid but shown in the internal display units, in this case if you use mm you will the the value in inches displayed as mm.

But we are scripting so it is up to you to make experiments with your actual settings and see what is right or what is wrong for your needs and what is generating errors.

The big advantage of scripting thins is that you could write a very complex script that will generate a very complex object and change a couple of parameters and rerunning the script a new object is generated.

This little text has only scratched the surface about how to create a script taking as example the Python Console output, and how to make some translation from the "echoed" commands to a more proper Python script.

So for now bye and stay tuned for further articles.