

Python Graphical User Interface

Today’s world is driven by fast-evolving technologies that require controlling complex instruments and developing sophisticated processes. This high demand leads to errors and challenges when mass-producing devices in a short time. In this sense, tools such as Python and the building of graphical interfaces (GUI) allow engineers and researchers to visualise data, manage experiments, and automate workflows with clarity and precision. Python’s clear syntax and extensive libraries encourage creativity while reducing development complexity, allowing ideas to move quickly from concept to application. Well-designed GUIs enhance user experience, improve productivity, and broaden the reach of technology. **BEAMER** enables Python GUI development within its Python module. In this application note, we will introduce the usage of this **BEAMER** tool.

INTRODUCTION

Python offers different tools for Graphical User Interface (GUI) development. Among the alternatives, **BEAMER** integrates PyQt and PySide, which are based on the Qt 5 framework, giving cross-platform compatibility and an extensive set of widgets. The pre-built widgets make it user-friendly for new and experienced users, warranting efficiency and robustness. Now, we will show users how to start using the Python Module with integrated GUI scripting.

PYTHON GUI SCRIPTING

The Python module, found under the Control modules, includes two tabs: the Python Script and the Python GUI Script (see Fig. 1).

To develop a GUI application, both tabs are necessary. The Python GUI Script, as its name indicates, determines the visual representation of the application while the Python Script defines the operational core of the application.

To exemplify the usage of both environments, let us create a GUI for drawing a LayoutPy-Circle where the user can change the radius without modifying the script of the shape. Additionally, users can find and use, from the Appendix section, all the codes displayed in the following images.

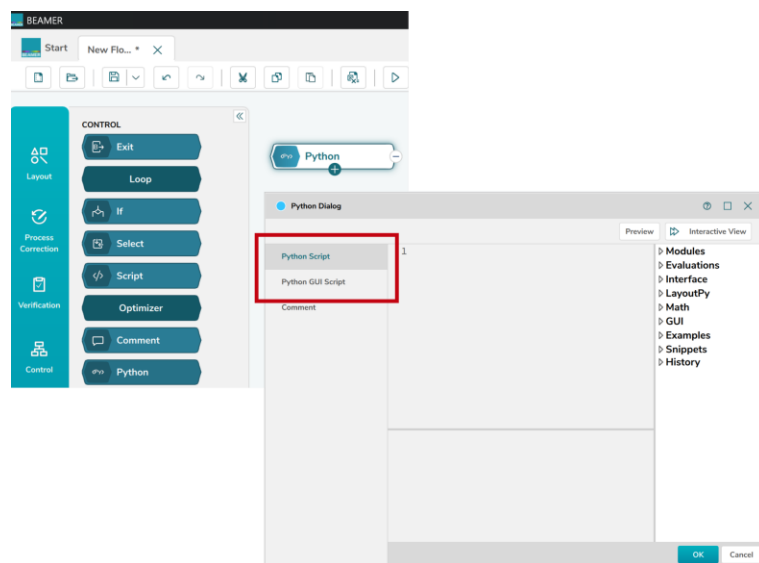


Figure 1. Python Module has two scripting environments, the Python Script and the Python GUI Script.

1. PYTHON SCRIPT TAB

We start dragging and dropping the script for a Circle found under the Examples dropdown panel (see Fig. 2, and script A1).

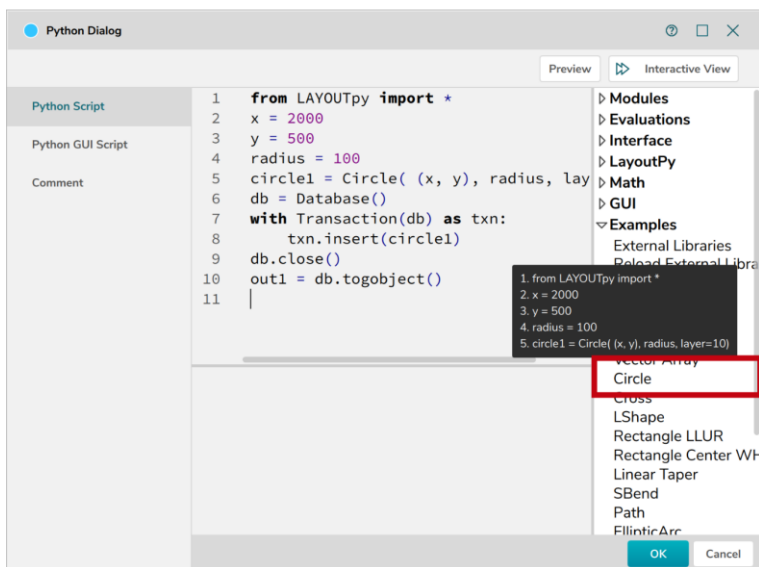


Figure 2. Script under the Examples tab for drawing a Circle.

Since we want users to only change the radius using the graphical interface, we modify the script as follows:

1. We remove the x , y , and $radius$ variables.
2. We set the x and y values to 0 within the *Circle* command (shape centred in the origin).
3. We change the $radius$ within the *Circle* command for `get_gui_parameter('radius', '20')` which creates the communication between the script and the GUI. In this case, we set that the radius starting value is 20 nm .
4. We define that the input value is either an integer `int()` or a floating `float()`. For this example, we write `float(get_gui_parameter('radius', '20'))`.

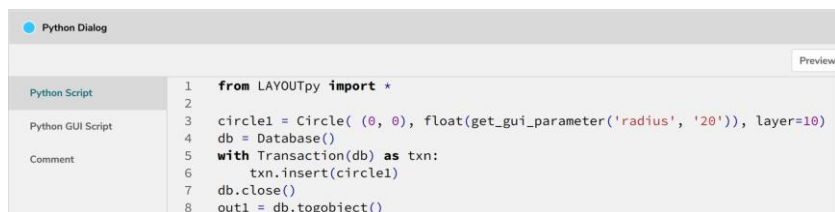


Figure 3. Modified script to create communication with graphical interface.

The modified script (see Fig. 3, and script A2) is a functional code that runs and creates a circle with the desired starting values (see Fig. 4).

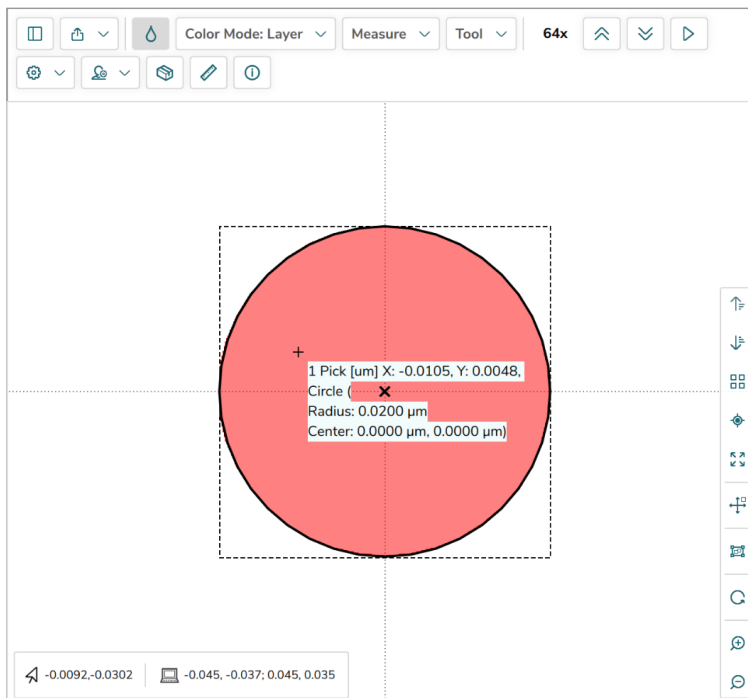


Figure 4. Circle using $x = y = 0$ and radius = 20 nm.

2. PYTHON GUI SCRIPT TAB

Now, we move to the Python GUI Script to create the objects for the application. For this, we use the class `QDialog` which is a fundamental widget for representing a pop-up window that includes commands for decision-making actions (for instance, *Ok*, *Save*, and *Cancel* buttons).

Figure 5 shows the script to create an *Ok* and a *Cancel* buttons on a pop-up window named *Draw Circle*. The script creates a horizontal layout to place the buttons side by side; moreover, the code uses two definitions to state the action to perform when hitting either of the buttons (see script A3).

```

Python Dialog
Python Script
Python GUI Script
Comment
Execute

1 class ExampleDialog(QDialog):
2     def __init__(self):
3         super().__init__()
4
5         self.setWindowTitle('Draw Circle')
6
7         layout = QVBoxLayout()
8
9         # OK and CANCEL buttons
10        button_confirm = QPushButton('Ok')
11        button_confirm.pressed.connect(self.on_confirm)
12
13        button_cancel = QPushButton('Cancel')
14        button_cancel.pressed.connect(self.on_cancel)
15
16        buttonBox = QDialogButtonBox(Qt.Horizontal)
17        buttonBox.addButton(button_confirm, QDialogButtonBox.AcceptRole)
18        buttonBox.addButton(button_cancel, QDialogButtonBox.RejectRole)
19
20        h1 = QHBoxLayout()
21        h1.addWidget(buttonBox)
22        layout.addLayout(h1)
23
24        self.setLayout(layout)
25        self.setModal(True)
26
27    def on_confirm(self):
28        gui_confirm()
29        self.close()
30
31    def on_cancel(self):
32        gui_cancel()
33        self.close()
34
35    # Run the dialog
36    dialog = ExampleDialog()
37    dialog.exec_()

```

Figure 5. Script using the class `QDialog` to create a pop-up window that uses an *Ok* and a *Cancel* buttons.

Click *Ok* on the Python Dialog to test the current script, and double left click on the Python module. The user should notice that before starting coding in the Python GUI Script tab, the double left click opens the Python Dialog; however, the GUI scripting transforms the Python module into an application. Accordingly, the double left click opens the pop-up window with the title *Draw Circle* that includes the *Ok* and *Cancel* actions (see Fig. 6).

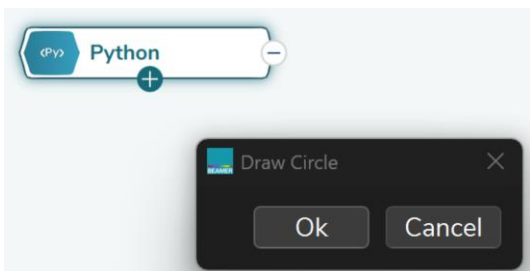


Figure 6. Python Module opening a pop-up window with action *Ok* and *Cancel*.

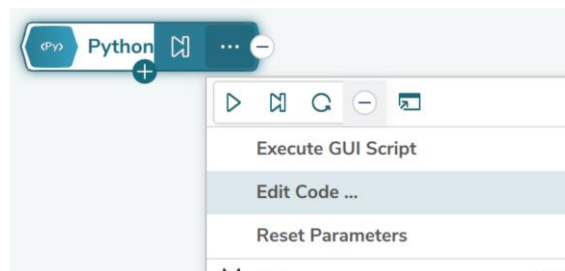


Figure 7. 'Edit Code...' option using the three dots menu of the Python module.

To return to the Python Dialog, click on the three dots on the module and select the *Edit Code...* option (see Fig. 7). The next step, it is to include the control over the radius mentioned in Fig. 3. Therefore, we add code for controlling the parameter radius (see Fig. 8 left) and an action to edit and use new values (see Fig. 8 right).

```

7 layout = QVBoxLayout()
8
9 # Parameter control
10 label = QLabel('radius')
11 self.test_edit = QLineEdit()
12 self.test_edit.setText(get_gui_parameter('radius', '20'))
13
14 layout.addWidget(label)
15 layout.addWidget(self.test_edit)
16
17 # OK and CANCEL buttons

```

```

35 def on_confirm(self):
36     set_gui_parameter('radius', self.test_edit.text())
37     gui_confirm()
38     self.close()

```

Figure 8. (Left) Script for controlling the radius and (Right) Action on the *Ok* button to store the new values.

Finally, testing the new script shows the pop-up window including the *radius* parameter with the initial value of *20 nm* (see Fig. 9, and scrip A4).

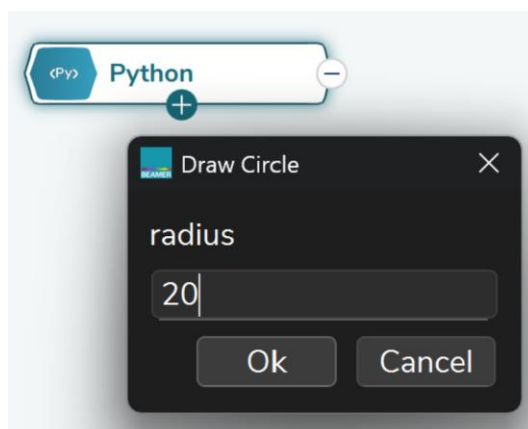


Figure 9. Draw Circle application.

EXAMPLES AND RESOURCES

BEAMER's Python module includes GUI scripting examples with useful widgets as starting point. To access these instances, drag and drop the desired code from the GUI dropdown menu on the right panel into the command line of the Python Script tab and the Python GUI Script tab (see Fig. 10). The user must select the codes with the same name for the different scripting tabs.

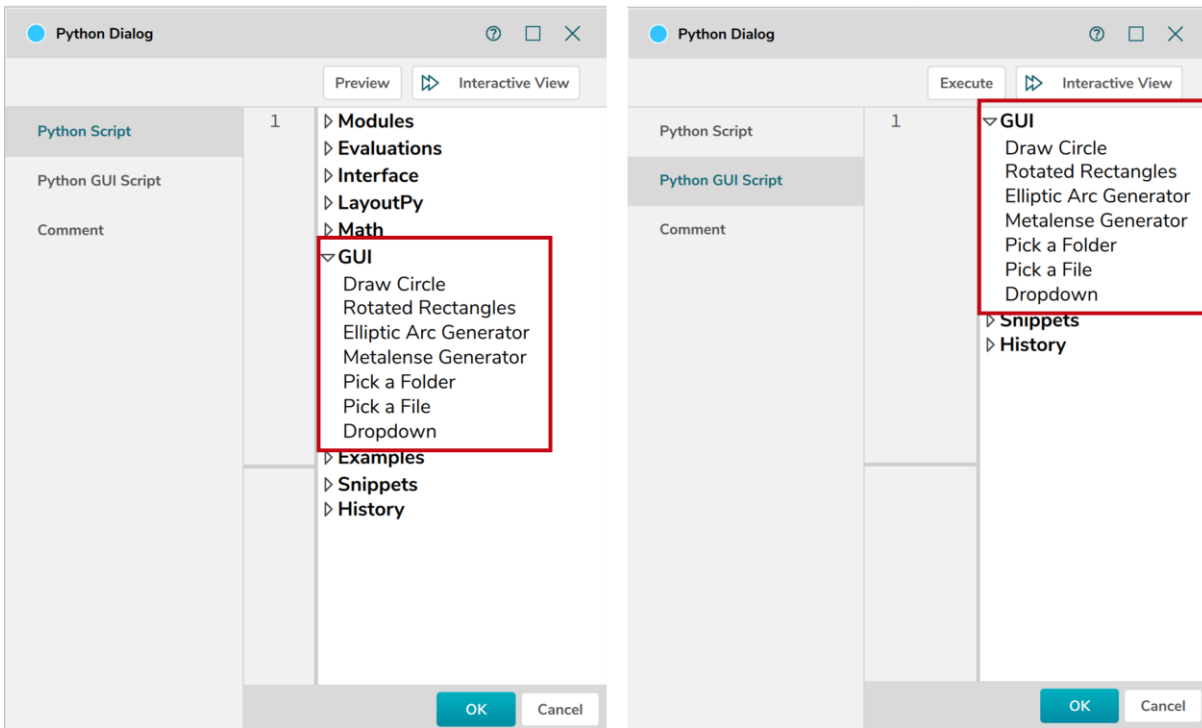


Figure 10. Location of the GUI examples for the (Left) Python Script tab and (Right) Python GUI Script tab.

Finally, there are numerous resources to find available widgets within the *QDialog*, the users can get more information in <https://doc.qt.io/archives/qt-5.15/qdialog.html> and <https://doc.qt.io/archives/qt-5.15/qwidget.html>.

APPENDIX

Here, the user will find the scripts explained throughout this document as such that can be copied for testing on the **BEAMER** Python module.

A1. FIGURE 2 SCRIPT – LAYOUTPY CIRCLE

```

from LAYOUTpy import *
x = 2000
y = 500
radius = 100
circle1 = Circle( (x, y), radius, layer=10)
db = Database()
with Transaction(db) as txn:
    txn.insert(circle1)
db.close()
out1 = db.togobject()

```

A2. FIGURE 3 SCRIPT – MODIFIED CIRCLE

```

from LAYOUTpy import *

circle1 = Circle( (0, 0), float( get_gui_parameter('radius', '20') ), layer=10)
db = Database()
with Transaction(db) as txn:
    txn.insert(circle1)
db.close()
out1 = db.togobject()

```

A3. FIGURE 5 SCRIPT – USING THE CLASS QDIALOG

```

class ExampleDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('Draw Circle')

        layout = QVBoxLayout()

        # OK and CANCEL buttons
        button_confirm = QPushButton('Ok')
        button_confirm.pressed.connect(self.on_confirm)

```

```

button_cancel = QPushButton('Cancel')
button_cancel.pressed.connect(self.on_cancel)

buttonBox = QDialogButtonBox(Qt.Horizontal)
buttonBox.addButton(button_confirm, QDialogButtonBox.AcceptRole)
buttonBox.addButton(button_cancel, QDialogButtonBox.RejectRole)

hl = QHBoxLayout()
hl.addWidget(buttonBox)
layout.addLayout(hl)

self.setLayout(layout)
self.setModal(True)

def on_confirm(self):
    gui_confirm()
    self.close()

def on_cancel(self):
    gui_cancel()
    self.close()

# Run the dialog
dialog = ExampleDialog()
dialog.exec_()

```

A4. FIGURE 6 SCRIPT – COMPLETE GUI APPLICATION

```

class ExampleDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('Draw Circle')

        layout = QVBoxLayout()

        # Parameter control
        label = QLabel('radius')
        self.test_edit = QLineEdit()
        self.test_edit.setText(get_gui_parameter('radius', '20'))

        layout.addWidget(label)
        layout.addWidget(self.test_edit)

```

```
# OK and CANCEL buttons
button_confirm = QPushButton('Ok')
button_confirm.pressed.connect(self.on_confirm)

button_cancel = QPushButton('Cancel')
button_cancel.pressed.connect(self.on_cancel)

buttonBox = QDialogButtonBox(Qt.Horizontal)
buttonBox.addButton(button_confirm, QDialogButtonBox.AcceptRole)
buttonBox.addButton(button_cancel, QDialogButtonBox.RejectRole)

hl = QHBoxLayout()
hl.addWidget(buttonBox)
layout.addLayout(hl)

self.setLayout(layout)
self.setModal(True)

def on_confirm(self):
    set_gui_parameter('radius', self.test_edit.text())
    gui_confirm()
    self.close()

def on_cancel(self):
    gui_cancel()
    self.close()

# Run the dialog
dialog = ExampleDialog()
dialog.exec_()
```