# Tutorial 4: How to Write Script Plugin

**Download the example scripts:** https://github.com/raemorris/VRED-Python-Tutorial-Examples-Scripts/blob/main/tutorial4.zip?raw=true

## Sample Python Code

Here are the accompanying example Python scripts for the *Tutorial 4: How to Write Script Plugin* video.

- To view the video, visit: https://www.youtube.com/watch?v=ueUd9pLpDRQ
- To review the video caption, see *Video Captions*.

## Simple Plugin

There is the sample code for **render_viewpoints_simple.py**:

```
import vrFileIO
import vrFileDialog
import vrRenderSettings
from PySide2 import QtCore, QtWidgets
from shiboken2 import wrapInstance

def vredMainWindow():
    main_window_ptr = getMainWindow()
    return wrapInstance(int(main_window_ptr), QtWidgets.QMainWindow)

class CustomDialog(QtWidgets.QDialog):
    def __init__(self, parent=None):
        super(MyDialog, self).__init__(parent)

        boxlayout = QtWidgets.QVBoxLayout(self)

        self.lineedit = QtWidgets.QLineEdit()
        boxlayout.addWidget(self.lineedit)
```

```
        self.button = QtWidgets.QPushButton("Set Label")
        self.button.clicked.connect(self.buttonClicked)
        boxlayout.addWidget(self.button)

        self.label = QtWidgets.QLabel()
        boxlayout.addWidget(self.label)

        self.setLayout(boxlayout)

    def buttonClicked(self):
        self.label.setText(self.lineedit.text())
        self.lineedit.setText("")



def renderViewpoints():
    renderDirectory = vrFileDialog.getExistingDirectory("Select a render
directory:", vrFileIO.getFileIOBaseDir())

    if not renderDirectory:
        print("No directory where to save the renderings!")
        return

    viewpoints = vrCameraService.getAllViewpoints()
    for viewpoint in viewpoints:
        name = viewpoint.getName()
        vrRenderSettings.setRenderFilename("{}.jpg".format(name))
        vrRenderSettings.startRenderToFile(False)



dialog = CustomDialog(vredMainWindow())
dialog.show()
```

## Viewpoint Plugin

There are three example files:

- icon_random_viewpoint.png
- viewpoint_plugin.py
- viewpoint_plugin.ui

## viewpoint_plugin.py

```
from PySide2 import QtCore, QtGui, QtWidgets


import os

import random

import threading


# Import vred modules in a try-catch block to prevent any errors

# Abort plugin initialization when an error occurs

importError = False

try:

    import vrController

    import vrFileIO

    import vrMovieExport

    import vrFileDialog

    import vrRenderSettings

except ImportError:

    importError = True

    pass


import uiTools


# Load a pyside form and the widget base from a ui file that describes the
layout

form, base = uiTools.loadUiType('viewpoint_plugin.ui')
```

```
class vrViewpointPlugin(form, base):
    """
    Main plugin class
    Inherits from fhe form and the widget base that was generated from the
ui-file
    """

    def __init__(self, parent=None):
        """Setup and connect the plugins user interface"""

        super(vrViewpointPlugin, self).__init__(parent)
        parent.layout().addWidget(self)
        self.parent = parent
        self.setupUi(self)
        self.setupUserInterface()

        # Initialize some class variables that we need for our loop function
        self.loopCounter = 0
        self.loopRunning = False



    def setupUserInterface(self):
        """Setup and connect the plugins user interface"""

        self._render_all.clicked.connect(self.renderViewpoints)

        self._random_viewpoint.clicked.connect(self.selectRandomViewpoint)
self._random_viewpoint.setIcon(QtGui.QIcon("icon_random_viewpoint.png"))
        self._random_viewpoint.setIconSize(QtCore.QSize(32,32))

        self._loop_viewpoints.clicked.connect(self.loopViewpoints)
```

```
    def renderViewpoints(self):
        """
        Open a directory dialog and then render all viewpoints to that
directory
        """

        print("[Viewpoint Plugin] Render all viewpoints...")
        renderDirectory = vrFileDialog.getExistingDirectory("Select a render
directory:", vrFileIO.getFileIOBaseDir())
        if not renderDirectory:
            print("No directory where to save the renderings!")
            return

        viewpoints = vrCameraService.getAllViewpoints()
        for viewpoint in viewpoints:
            name = viewpoint.getName()
            viewpoint.activate()
            print("{}/{}.jpg".format(renderDirectory, name))

vrRenderSettings.setRenderFilename("{}/{}.jpg".format(renderDirectory, name))
            vrRenderSettings.startRenderToFile(False)

        msgBox = QtWidgets.QMessageBox()
        msgBox.setWindowTitle("Finished Rendering Viewpoints")
        msgBox.setInformativeText("Finished Rendering Viewpoints. Do you want
to open the render directory?")
        msgBox.setStandardButtons(QtWidgets.QMessageBox.Ok |
QtWidgets.QMessageBox.Cancel)
        ret = msgBox.exec_()

        if ret == QtWidgets.QMessageBox.Ok:
            os.startfile(renderDirectory)
```

```
    def selectRandomViewpoint(self):
        """ Select a random viewpoint from all viewpoints """

        print("[Viewpoint Plugin] Select a random viewpoint...")

        viewpoints = vrCameraService.getAllViewpoints()
        randomViewpoint = random.choice(viewpoints)
        randomViewpoint.activate()



    def loopViewpoints(self):
        """
        Loops through all viewpoints once. Stops looping when the button is
pressed again
        """

        # When a loop is already running, then cancel the loop
        if self.loopRunning:
            print("[Viewpoint Plugin] Stop loop...")
            self.__setLoopViewpointLabel("Loop Viewpoints")

            self.loopRunning = False
            return

        # Otherwise start a new loop
        if self.loopCounter == 0 and not self.loopRunning:
            print("[Viewpoint Plugin] Loop all viewpoints...")
            self.__setLoopViewpointLabel("Stop Loop")

            viewpoints = vrCameraService.getAllViewpoints()
```

```
            self.loopCounter = len(viewpoints) - 1
            self.loopRunning = True
            threading.Timer(1.0, self.__loopNextViewpoint).start()


    def __loopNextViewpoint(self):
        """
        Loops through all viewpoints once. Stops looping when the button is
pressed again
        """

        if self.loopCounter == 0 or not self.loopRunning:
            self.loopRunning = False
            self.loopCounter = 0
            return

        viewpoints = vrCameraService.getAllViewpoints()
        viewpoints[self.loopCounter].activate()
        self.loopCounter = self.loopCounter - 1
        threading.Timer(2.0, self.__loopNextViewpoint).start()


    def __setLoopViewpointLabel(self, labelText):
        """ Change the text of the "loop" button """

        self._loop_viewpoints.setText(labelText)

# Actually start the plugin
if not importError:
    viewpointPlugin = vrViewpointPlugin(VREDPluginWidget)
```

## viewpoint_plugin.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>vrViewpointPlugin</class>
 <widget class="QWidget" name="vrViewpointPluginGUI">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>300</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Viewpoint Plugin</string>
  </property>

  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="QPushButton" name="_render_all">
     <property name="text">
      <string>Render All Viewpoints</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QPushButton" name="_random_viewpoint">
    <property name="text">
      <string>Render All Viewpoints</string>
     </property>
    </widget>
```

```
   </item>
   <item>
    <widget class="QPushButton" name="_loop_viewpoints">
     <property name="text">
      <string>Loop Viewpoints</string>
     </property>
    </widget>
   </item>
  </layout>

 </widget>
 <resources/>
 <connections/>
</ui>
```

## Video Captions

Hello and welcome to our tutorial series Scripting with Python for VRED Pro. My name is Christopher and today I want to talk about Script Plugins and how you can write your own.

As we learned in the last tutorial, VRED offers many different interfaces where you can add your own Python scripts to customize your scene. With a few simple scripts, we can implement user interactions and tools that help us during production. When you want to offer specialized tools that should be available to all of your colleagues or be delivered to customers, you should consider implementing your tools as a script plugin.

Script plugins are a special kind of Python script. They are independent from the VRED scene and must be installed in a special script plugin directory in order to be used. You can access script plugins by opening the “Script” entry in the menu bar in VRED where every installed script plugin is listed. Each plugin can be enabled and disabled by clicking on the corresponding entry. Script plugins usually provide a user interface that is integrated into VRED and feels if it is just a part of VRED itself. That’s because the PySide widgets use the VRED window styles by default. In this tutorial, we will implement a script plugin together and, along the way, learn how to build a user interface and implement new tools.

For this video, I want to implement a simple plugin that helps me working with viewpoints. First, I want to implement a tool that will help me to render all viewpoints to a directory that I can specify. Secondly, I want to randomly select a viewpoint and set my camera to it.

To implement these functions, we first have to create our plugin class. This class setup will basically be the same for each plugin you will develop. I start by creating a new directory that is located under my user documents, Autodesk, VRED13.3, ScriptPlugins. This directory is called “ViewpointPlugin” and will

contain any plugin files we create. When this directory does not exist on your machine, you can just create it and VRED will try to read script plugins from here.

For the following examples, I work with Visual Studio Code with the Python extensions installed. I can open Visual Studio Code at the newly created directory and create two files: viewpoint_plugin.py and viewpoint_plugin.ui. The first file will contain our Python code and the second one will contain our user interface layout. At the top of our script, we import the modules we need from the PySide2 namespace, that is Qt Core, Qt GUI, and Qt Widgets. We also import UI tools that help us generate the plugin widget.

Next, we can load a form and the widget base, using the UI tools. Here we reference the UI file we just created. Then we can add our actual plugin class that inherits from the form and the base we just generated. The start of the __init__ constructor is always the same and you can just copy it for every plugin you write.

Below our class, we can instantiate our plugin, using the “VRED plugin widget” as the parameter of the constructor. Don’t worry if this is going a bit too fast and you don’t understand every bit of it. This is a lot of special plugin code that is always the same. You can copy this plugin structure from the plugin example from VRED when you want to develop your own. And, of course, you can always rewind this tutorial or hit the Pause button.

Ok, so on the Python side, we are done for now, but we also need to define our user interface. This is done in theviewpoint_plugin.ui file. The layout file uses an xml notation to define a user interface. In this tutorial, I will just add some buttons, so nothing too complicated. The important thing here is, that I add two Push buttons and give them a name. This name is later used in the Python script to reference the buttons. I recommend you have a look at the “QT designer” that offers an editor to create and export such QT layout files. Doing this by hand is fine for this tutorial, but very tedious for larger projects.

When we save our files, we can switch back to VRED and reload our Script plugins. We can see that there is another entry in the “Script” menu and when we click on it we see our new plugin interface. We already have an UI that we can open and close, but right now the functionality is still missing.

Let's implement our first tool that renders all of our viewpoints. We define a new function called “render viewpoints”. At first, we can use the file dialog module to request a directory from the user. When the user selects a directory, we are fine. But when they cancel the operation, we have to catch this and return from the function with doing nothing. In the next step, we iterate over a list of all available viewpoints and for every viewpoint, we activate it. After that, we generate a filename and render the viewpoint to a file. From the directory, the user selected and the name of the viewpoint, we can generate a file path that tells VRED where to store the renderings. When our script finished, it would also be nice to show a message dialog and ask our user if they want to open the directory where the files have been saved. This can be achieved with a “Q message box” that can return a user choice for you. It presents an OK and CANCEL button, and when a user clicks OK, we use the default Python way to open the directory.

When we would test it right now, nothing really would happen because we forgot a vital part of the script. We still have to connect our render function to the Push button. This is done by connecting to the clicked signal of our button that we reference with the name we gave it in the ui layout. And now, we are good to go. We can test our implementation by reloading the script plugins and starting our new

plugin. As expected, all viewpoints get rendered and we also are asked if we want to open the render directory right after the rendering finished.

Time to implement our second function. This time we want to add an icon to our button, instead of text. So we delete the text node from our second push button in the ui layout file and switch back to our Python file. We can add icons by setting them directly on the button with the setIcon function. This function needs a QIcon parameter as an input that we create with our icon image. Our icon is just a .png image that we store in our plugin directory.

We also have to set the size of the icon, so that it's not too large and fits in our user interface. Of course, we also have to connect this push button to a function. This time we do it beforehand and call the function selectRandomViewpoint. We add this function to our plugin class and start writing an implementation for it. This tool is much shorter, as we only have to select a random viewpoint and set it active. When we reload our plugin, we see that our text button changed into an image button and when we click it, a random viewpoint is selected. Nice!

From here on, you are free to implement whatever functionality you can think of! You can design you own tools and give it a unique look and feel. You could even implement a script plugin that uses a screenplate with an HTML user interface as an input. A thing you have to be careful about is, that VRED treats every Python script as a single script plugin. This makes it difficult to split your plugin into multiple files. You can work around this by moving the additional Python modules you use to the internal VRED Python library directory. Every Python module that is in this directory can be loaded from any script in VRED.

Another note: You might have noticed that we had to explicitly import some VRED modules in our script. You have this for all modules from the API version 1. Modules from the API version 2 are automatically added by VRED itself.

After this tutorial you should now be able to implement your own tools using script plugins. If you are just starting out, I would recommend you use the VRED plugin example as a starting point and fill it with your own implementations. Then you can start creating your own user interface with the Qt designer tools.

That's it for today! Thanks for watching and see you next time!