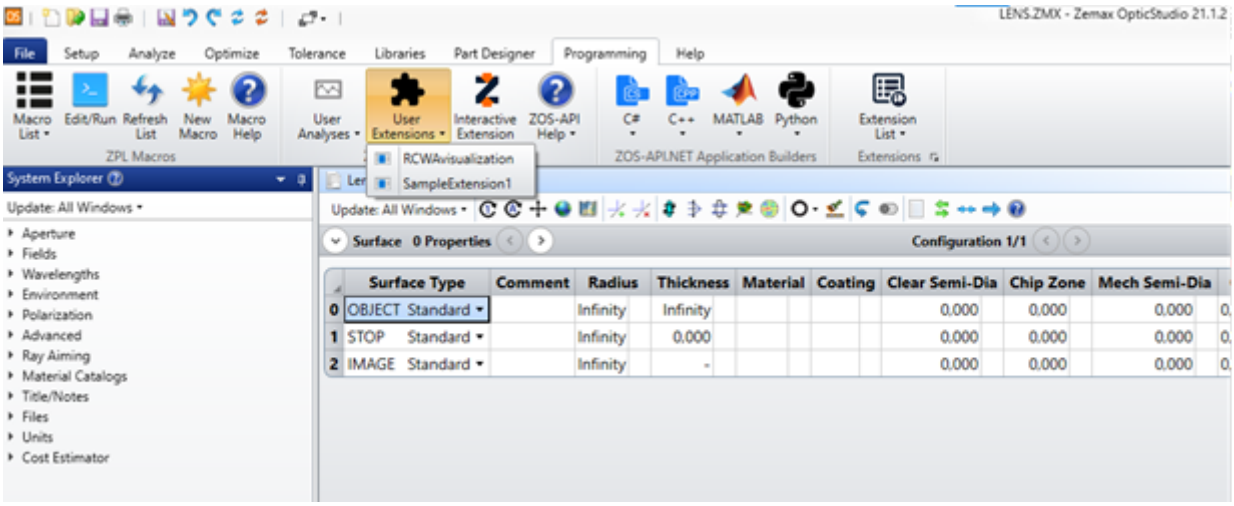
Abstract

This article will show how to make a ZOS-API user extension with a sample code written in Python.

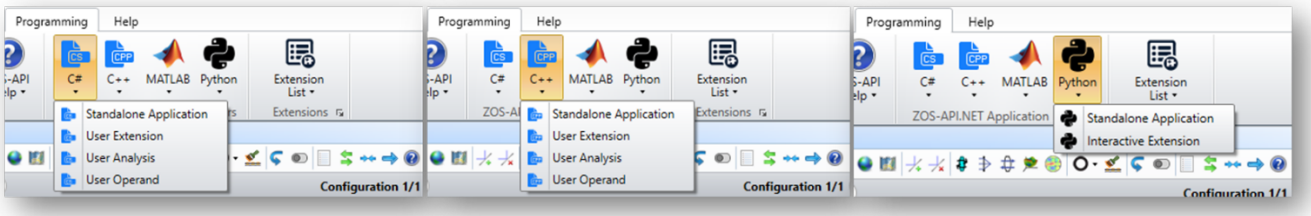
Contents

A brief description of the User Extension

Zemax Optic Studio® allows the user to integrate an application program using ZOS API as a User Extension. Basically, it's the user program bundled in a .exe file under *Zemax Folder* > *ZOS-API* > *Extensions* :



This is a useful feature because the program can work directly with the opened instance of Zemax, and the end user of the script doesn't have to set up a development environment. However, this feature is only available for C++ and C# languages. For Python, there are only samples of codes for the Interactive Extension or the Standalone Application :



Nevertheless, ZOS-API User Extension can be supported in Python with a small modification of the Standalone Application sample code.

Project Setup

The pre-requistes in order to have the support of the ZOS-API User Extension are the following :

- Have Zemax Optic Studio® **20.1 or higher** installed
- Set up ZOS-API with Python.NET (see this article for detailed explanations)

- Have the python module *numpy* installed (in the command prompt, type `python -m pip install numpy`)
- Have the python module *matplotlib* installed (in the command prompt, type `python -m pip install matplotlib==XX.XX.XX`)
- Have the python module *pyinstaller* (in the command prompt, type `python -m pip install pyinstaller`)
- Have the folder *PYTHON_FOLDER\Scripts* (for instance : *C:\Python38\Scripts*) declared as a PATH environment variable. This article explains how to add an environment variable for Windows 10.

A Basic Example

Source Example

We will use one of the code samples provided in the ZOS-API Documentation : **Example 02 : NSC Ray Trace**. Hereafter the python code sample, using the *Standalone Application* (Source file : *PythonStandalone_02_NSC_ray_trace.py* in the *ZOS-API Sample Code* folder) :

```
import clr, os, winreg
from itertools import islice

import matplotlib.pyplot as plt
import numpy as np

class PythonStandaloneApplication(object):
    class LicenseException(Exception):
        pass
    class ConnectionException(Exception):
        pass
    class InitializationException(Exception):
        pass
    class SystemNotPresentException(Exception):
        pass

    def __init__(self, path=None):
        # determine location of ZOSAPI_NetHelper.dll & add as reference
        aKey = winreg.OpenKey(winreg.ConnectRegistry(None,
winreg.HKEY_CURRENT_USER), r"Software\Zemax", 0, winreg.KEY_READ)
        zemaxData = winreg.QueryValueEx(aKey, 'ZemaxRoot')
        NetHelper = os.path.join(os.sep, zemaxData[0], r'ZOS-
API\Libraries\ZOSAPI_NetHelper.dll')
        winreg.CloseKey(aKey)
        clr.AddReference(NetHelper)
        import ZOSAPI_NetHelper

        # Find the installed version of OpticStudio
        if path is None:
            isInitialized = ZOSAPI_NetHelper.ZOSAPI_Initializer.Initialize()
        else:
            # Note -- uncomment the following line to use a custom initialization
path
            isInitialized = ZOSAPI_NetHelper.ZOSAPI_Initializer.Initialize(path)
```

```
        # determine the ZOS root directory
        if isInitialized:
            dir = ZOSAPI_NetHelper.ZOSAPI_Initializer.GetZemaxDirectory()
        else:
            raise PythonStandaloneApplication.InitializationException("Unable to
locate Zemax OpticStudio.  Try using a hard-coded path.")

        # add ZOS-API referencecs
        clr.AddReference(os.path.join(os.sep, dir, "ZOSAPI.dll"))
        clr.AddReference(os.path.join(os.sep, dir, "ZOSAPI_Interfaces.dll"))
        import ZOSAPI

        # create a reference to the API namespace
        self.ZOSAPI = ZOSAPI

        # create a reference to the API namespace
        self.ZOSAPI = ZOSAPI

        # Create the initial connection class
        self.TheConnection = ZOSAPI.ZOSAPI_Connection()

        if self.TheConnection is None:
            raise PythonStandaloneApplication.ConnectionException("Unable to
initialize .NET connection to ZOSAPI")

        self.TheApplication = self.TheConnection.CreateNewApplication()
        if self.TheApplication is None:
            raise PythonStandaloneApplication.InitializationException("Unable to
acquire ZOSAPI application")

        if self.TheApplication.IsValidLicenseForAPI == False:
            raise PythonStandaloneApplication.LicenseException("License is not
valid for ZOSAPI use")

        self.TheSystem = self.TheApplication.PrimarySystem
        if self.TheSystem is None:
            raise PythonStandaloneApplication.SystemNotPresentException("Unable to
acquire Primary system")

    def __del__(self):
        if self.TheApplication is not None:
            self.TheApplication.CloseApplication()
            self.TheApplication = None

        self.TheConnection = None

    def OpenFile(self, filepath, saveIfNeeded):
        if self.TheSystem is None:
            raise PythonStandaloneApplication.SystemNotPresentException("Unable to
acquire Primary system")
        self.TheSystem.LoadFile(filepath, saveIfNeeded)

    def CloseFile(self, save):
```

```
        if self.TheSystem is None:
            raise PythonStandaloneApplication.SystemNotPresentException("Unable to
acquire Primary system")
        self.TheSystem.Close(save)

    def SamplesDir(self):
        if self.TheApplication is None:
            raise PythonStandaloneApplication.InitializationException("Unable to
acquire ZOSAPI application")

        return self.TheApplication.SamplesDir

    def ExampleConstants(self):
        if self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusType.PremiumEdition:
            return "Premium"
        elif self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusTypeProfessionalEdition:
            return "Professional"
        elif self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusTypeStandardEdition:
            return "Standard"
        else:
            return "Invalid"

    def reshape(self, data, x, y, transpose = False):
        """Converts a System.Double[,] to a 2D list for plotting or post
processing

        Parameters
        ----------
        data      : System.Double[,] data directly from ZOS-API
        x         : x width of new 2D list [use var.GetLength(0) for dimension]
        y         : y width of new 2D list [use var.GetLength(1) for dimension]
        transpose : transposes data; needed for some multi-dimensional line series
data

        Returns
        -------
        res       : 2D list; can be directly used with Matplotlib or converted to
                    a numpy array using numpy.asarray(res)
        """
        if type(data) is not list:
            data = list(data)
        var_lst = [y] * x;
        it = iter(data)
        res = [list(islice(it, i)) for i in var_lst]
        if transpose:
            return self.transpose(res);
        return res

    def transpose(self, data):
        """Transposes a 2D list (Python3.x or greater).
```

```
        Useful for converting mutli-dimensional line series (i.e. FFT PSF)

        Parameters
        ----------
        data      : Python native list (if using System.Data[,] object reshape
first)

        Returns
        -------
        res       : transposed 2D list
        """
        if type(data) is not list:
            data = list(data)
        return list(map(list, zip(*data)))

if __name__ == '__main__':
    zos = PythonStandaloneApplication()

    #use http://matplotlib.org/ to plot 2D graph
    # need to install this package before running this code

    # load local variables
    ZOSAPI = zos.ZOSAPI
    TheApplication = zos.TheApplication
    TheSystem = zos.TheSystem
    sampleDir = TheApplication.SamplesDir

    #! [e02s01_py]
    # Open file
    testFile = os.path.join(os.sep, sampleDir,  r'Non-
sequential\Miscellaneous\Digital_projector_flys_eye_homogenizer.zmx')
    TheSystem.LoadFile(testFile, False)
    #! [e02s01_py]

    # ! [e02s02_py]
    # Create ray trace
    NSCRayTrace = TheSystem.Tools.OpenNSCRayTrace()
    NSCRayTrace.SplitNSCRays = True
    NSCRayTrace.ScatterNSCRays = False
    NSCRayTrace.UsePolarization = True
    NSCRayTrace.IgnoreErrors = True
    NSCRayTrace.SaveRays = False
    NSCRayTrace.Run()
    # ! [e02s02_py]

    lastValue = []
    lastValue.append(0)
    print('Beginning ray trace:')
    while NSCRayTrace.IsRunning:
        currentValue = NSCRayTrace.Progress
        if currentValue % 2 == 0:
            if lastValue[len(lastValue) - 1] != currentValue:
                lastValue.append(currentValue)
                print(currentValue)
```

```
    NSCRayTrace.WaitForCompletion()
    NSCRayTrace.Close()

    # Non-sequential component editor
    TheNCE = TheSystem.NCE

    DetObj = 4
    obj = TheSystem.NCE.GetObjectAt(DetObj);
    numXPixels = obj.ObjectData.NumberXPixels;
    numYPixels = obj.ObjectData.NumberYPixels;
    pltWidth   = 2 * obj.ObjectData.XHalfWidth;
    pltHeight  = 2 * obj.ObjectData.YHalfWidth;

    pix = 0

    #! [e02s03_py]
    # Get detector data
    detectorData = [[0 for x in range(numYPixels)] for x in range(numXPixels)]
    for x in range(0,numYPixels,1):
        for y in range(0,numXPixels,1):
            ret, pixel_val = TheNCE.GetDetectorData(DetObj, pix, 1, 0)
            pix += 1
            if ret == 1:
                detectorData[y][x] = pixel_val
            else:
                detectorData[x][y] = -1
    #! [e02s03_py]

    # end of default code
    # everything below here is based on numpy/matplotlib and is not supported by
ZOSAPI or Zemax
    # https://docs.scipy.org/doc/numpy/index.html
    # http://matplotlib.org/

    # This will clean up the connection to OpticStudio.
    # Note that it closes down the server instance of OpticStudio, so you for
maximum performance do not do
    # this until you need to.

    detectorData = zos.transpose(detectorData)

    del zos
    zos = None

    # text output & FOR loops for OpticStudio will invert the vertical image
    # place plt.show() after clean up to release OpticStudio from memory
    plt.imshow(detectorData)
    plt.show()
```

Basically, this code opens the Zemax File located in *ZEMAX_FOLDER\Samples\Non-sequential\Miscellaneous\Digital_projector_flys_eye_homogenizer.zmx*, launches a Ray Tracing operation, and displays the result on the detector, using the *matplotlib* library.

We can transpose this Standalone Application code into a User Extension code following these steps, described in the sections below :

1. Create a Python class `PythonUserExtension`
2. Use this new class within the code sample, and create a .exe file using the *pyinstaller* library
3. Open in a Zemax Optic Studio® instance the *Digital_projector_flys_eye_homogenizer.zmx* and execute the generated User Extension

The PythonUserExtension class

The following class has to be used in a ZOS-API User Extension code written in Python :

```
# This boilerplate requires the 'pythonnet' module.
# The following instructions are for installing the 'pythonnet' module via pip:
#    1. Ensure you are running Python 3.4, 3.5, 3.6, or 3.7. PythonNET does not
work with Python 3.8 yet.
#    2. Install 'pythonnet' from pip via a command prompt
#    (type 'cmd' from the start menu or press Windows + R and type 'cmd' then
enter)
#
#        python -m pip install pythonnet

class PythonUserExtension(object):
    class LicenseException(Exception):
        pass

    class ConnectionException(Exception):
        pass

    class InitializationException(Exception):
        pass

    class SystemNotPresentException(Exception):
        pass

    def __init__(self, path=None):
        # determine location of ZOSAPI_NetHelper.dll & add as reference
        a_key = winreg.OpenKey(winreg.ConnectRegistry(None,
winreg.HKEY_CURRENT_USER), r"Software\Zemax", 0,
                               winreg.KEY_READ)
        zemax_data = winreg.QueryValueEx(a_key, 'ZemaxRoot')
        net_helper = os.path.join(os.sep, zemax_data[0], r'ZOS-
API\Libraries\ZOSAPI_NetHelper.dll')
        winreg.CloseKey(a_key)
        clr.AddReference(net_helper)
        import ZOSAPI_NetHelper

        # Find the installed version of OpticStudio
        # if len(path) == 0:
        if path is None:
            is_initialized = ZOSAPI_NetHelper.ZOSAPI_Initializer.Initialize()
        else:
```

```
            # Note -- uncomment the following line to use a custom initialization
path
            is_initialized = ZOSAPI_NetHelper.ZOSAPI_Initializer.Initialize(path)

        # determine the ZOS root directory
        if is_initialized:
            zemax_dir = ZOSAPI_NetHelper.ZOSAPI_Initializer.GetZemaxDirectory()
        else:
            raise PythonUserExtension.InitializationException(
                "Unable to locate Zemax OpticStudio.  Try using a hard-coded
path.")

        # add ZOS-API references
        clr.AddReference(os.path.join(os.sep, zemax_dir, "ZOSAPI.dll"))
        clr.AddReference(os.path.join(os.sep, zemax_dir, "ZOSAPI_Interfaces.dll"))
        import ZOSAPI

        # create a reference to the API namespace
        self.ZOSAPI = ZOSAPI

        # create a reference to the API namespace
        self.ZOSAPI = ZOSAPI

        # Create the initial connection class
        self.TheConnection = ZOSAPI.ZOSAPI_Connection()

        if self.TheConnection is None:
            raise PythonUserExtension.ConnectionException("Unable to initialize
.NET connection to ZOSAPI")

        self.TheApplication = self.TheConnection.ConnectToApplication()
        if self.TheApplication is None:
            raise PythonUserExtension.InitializationException("Unable to acquire
ZOSAPI application")

        if self.TheApplication.Mode != ZOSAPI.ZOSAPI_Mode.Plugin:
            raise PythonUserExtension. \
                InitializationException("User plugin was started in the wrong
mode: expected Plugin, found ",

ZOSAPI.ZOSAPI_Mode.GetName(ZOSAPI.ZOSAPI_Mode, self.TheApplication.Mode))

        if not self.TheApplication.IsValidLicenseForAPI:
            raise PythonUserExtension.LicenseException("License is not valid for
ZOSAPI use")

        self.TheSystem = self.TheApplication.PrimarySystem
        if self.TheSystem is None:
            raise PythonUserExtension.SystemNotPresentException("Unable to acquire
Primary system")

        self.TheApplication.ProgressPercent = 0
        self.TheApplication.ProgressMessage = 'Running Extension...'
```

```
    def __del__(self):
        if self.TheApplication is not None:
            self.TheApplication.ProgressMessage = 'Complete'
            self.TheApplication.ProgressPercent = 100

    def OpenFile(self, filepath, save_if_needed):
        if self.TheSystem is None:
            raise PythonUserExtension.SystemNotPresentException("Unable to acquire
Primary system")
        self.TheSystem.LoadFile(filepath, save_if_needed)

    def CloseFile(self, save):
        if self.TheSystem is None:
            raise PythonUserExtension.SystemNotPresentException("Unable to acquire
Primary system")
        self.TheSystem.Close(save)

    def SamplesDir(self):
        if self.TheApplication is None:
            raise PythonUserExtension.InitializationException("Unable to acquire
ZOSAPI application")

        return self.TheApplication.SamplesDir

    def ExampleConstants(self):
        if self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusType.PremiumEdition:
            return "Premium"
        elif self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusTypeProfessionalEdition:
            return "Professional"
        elif self.TheApplication.LicenseStatus ==
self.ZOSAPI.LicenseStatusTypeStandardEdition:
            return "Standard"
        else:
            return "Invalid"

    def reshape(self, data, x, y, transpose = False):
        """Converts a System.Double[,] to a 2D list for plotting or post
processing

        Parameters
        ----------
        data      : System.Double[,] data directly from ZOS-API
        x         : x width of new 2D list [use var.GetLength(0) for dimension]
        y         : y width of new 2D list [use var.GetLength(1) for dimension]
        transpose : transposes data; needed for some multi-dimensional line series
data

        Returns
        -------
        res       : 2D list; can be directly used with Matplotlib or converted to
                    a numpy array using numpy.asarray(res)
        """
```

```
        if type(data) is not list:
            data = list(data)
        var_lst = [y] * x;
        it = iter(data)
        res = [list(islice(it, i)) for i in var_lst]
        if transpose:
            return self.transpose(res);
        return res

    def transpose(self, data):
        """Transposes a 2D list (Python3.x or greater).

        Useful for converting mutli-dimensional line series (i.e. FFT PSF)

        Parameters
        ----------
        data      : Python native list (if using System.Data[,] object reshape
first)

        Returns
        -------
        res       : transposed 2D list
        """
        if type(data) is not list:
            data = list(data)
        return list(map(list, zip(*data)))
```

In the main code, we now have to call this class (instead of the `PythonStandaloneApplication` class). Moreover, there is no need to keep the code that loads the Zemax File (the User Extension will be called in the opened instance). Hereafter the reworked main part of the example :

```
if __name__ == '__main__':
    zos = PythonUserExtension()

    #use http://matplotlib.org/ to plot 2D graph
    # need to install this package before running this code

    # load local variables
    ZOSAPI = zos.ZOSAPI
    TheApplication = zos.TheApplication
    TheSystem = zos.TheSystem

    # ! [e02s02_py]
    # Create ray trace
    NSCRayTrace = TheSystem.Tools.OpenNSCRayTrace()
    NSCRayTrace.SplitNSCRays = True
    NSCRayTrace.ScatterNSCRays = False
    NSCRayTrace.UsePolarization = True
    NSCRayTrace.IgnoreErrors = True
    NSCRayTrace.SaveRays = False
    NSCRayTrace.Run()
```

```
    # ! [e02s02_py]

    lastValue = []
    lastValue.append(0)
    print('Beginning ray trace:')
    while NSCRayTrace.IsRunning:
        currentValue = NSCRayTrace.Progress
        if currentValue % 2 == 0:
            if lastValue[len(lastValue) - 1] != currentValue:
                lastValue.append(currentValue)
                print(currentValue)
    NSCRayTrace.WaitForCompletion()
    NSCRayTrace.Close()

    # Non-sequential component editor
    TheNCE = TheSystem.NCE

    DetObj = 4
    obj = TheSystem.NCE.GetObjectAt(DetObj);
    numXPixels = obj.ObjectData.NumberXPixels;
    numYPixels = obj.ObjectData.NumberYPixels;
    pltWidth   = 2 * obj.ObjectData.XHalfWidth;
    pltHeight  = 2 * obj.ObjectData.YHalfWidth;

    pix = 0

    #! [e02s03_py]
    # Get detector data
    detectorData = [[0 for x in range(numYPixels)] for x in range(numXPixels)]
    for x in range(0,numYPixels,1):
        for y in range(0,numXPixels,1):
            ret, pixel_val = TheNCE.GetDetectorData(DetObj, pix, 1, 0)
            pix += 1
            if ret == 1:
                detectorData[y][x] = pixel_val
            else:
                detectorData[x][y] = -1
    #! [e02s03_py]

    # end of default code
    # everything below here is based on numpy/matplotlib and is not supported by
ZOSAPI or Zemax
    # https://docs.scipy.org/doc/numpy/index.html
    # http://matplotlib.org/

    # This will clean up the connection to OpticStudio.
    # Note that it closes down the server instance of OpticStudio, so you for
maximum performance do not do
    # this until you need to.

    detectorData = zos.transpose(detectorData)

    del zos
    zos = None
```

```
    # text output & FOR loops for OpticStudio will invert the vertical image
    # place plt.show() after clean up to release OpticStudio from memory
    plt.imshow(detectorData)
    plt.show()
```

Generate the executable

We now have a reworked python code sample, *nscraytrace_user_extension.py* (attached to this article), and we have to generate a .exe file in order to use it as an User Extension.

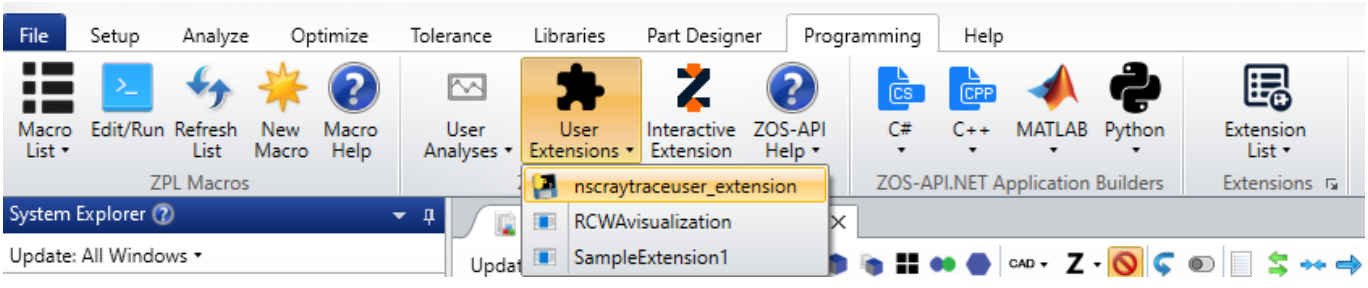
For this purpose, the python library pyinstaller can be used. In your working folder, open the command prompt and type :

```
pyinstaller --onefile nscraytrace_user_extension.py
```

This will generate a dist folder, and within it a .exe file *nscraytrace_user_extension.exe* that can be used as an User Extension. The *pyinstaller* module will also generate a *__pycache__* and a *build* subfolders : these folders and their content can be deleted.
Note that the `--onefile` option bundle the script and all its dependencies into a single executable. This is the simplest solution, considering that the executable has to be launched from a Zemax Optic Studio® instance, but pyinstaller provides other bundling possibilities.

Use the extension

All there is to do now is to copy the executable file *nscraytrace_user_extension.exe* into *ZEMAX_FOLDER\ZOS-API\Extensions* folder. Then, open a Zemax Optic Studio® instance and load the file *ZEMAX_FOLDER\Samples\Non-sequential\Miscellaneous\Digital_projector_flys_eye_homogenizer.zmx*. Our program should be in the User Extension list :



This extension can be launched, and works exactly as the standalone application code.

Conclusion

This article has shown how to make a ZOS-API User Extension with a Python script. It was illustrated with a basic example, but with the Python script boilerplate attached to this article (*PythonUserExtension.py*), it can be used with any Python script using ZOS-API.