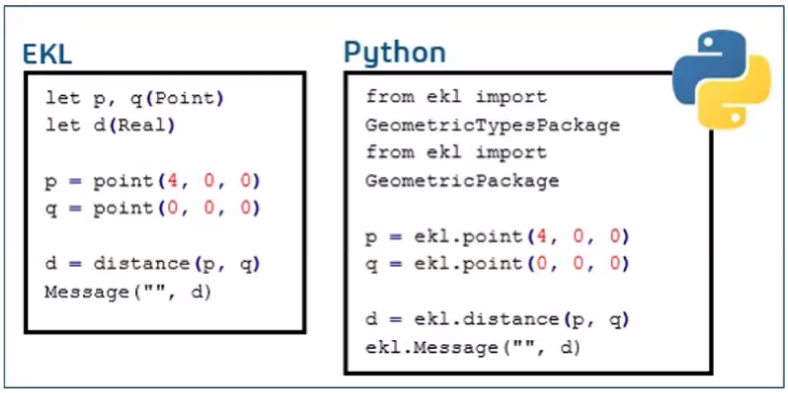
Python Automation

AUTOMATION | EKL & PYTHON OPENESS

R2022xFD

- Enhanced productivity for automation scenarios with the ability to add or retrieve URLs on parameters through Enterprise Knowledge Language (EKL)
- Increased usability and functional coverage for design automation scenarios by providing Python scripting capabilities.
  - Python is the most widely used language especially in STEM disciplines.
  - As a language, Python is a very complete, performant and easy to use.
  - It provides many discipline-specific libraries that can be readily referenced by users.



DASSAULT SYSTEMES

Introduction

Python has become one of the most popular scripting language in Science, Technology, Engineering and Mathematics disciplines.

Python is a very complete, performant and easy to use programing language. It provides many discipline-specific libraries that can be readily referenced by users.

Providing python scripting capabilities as a complement to our Enterprise Knowledge Language (EKL) scripting strengthens design automation capabilities on the 3DEXPERIENCE Platform.

Finally with Python, the access to scripting capabilities is easier and available to a larger audience democratizing design automation.

A simple demonstration illustrating some unique capabilities offered by the python language for design automation scenarios.

Aim

The aim of the python editor is to offer access to:

the standard Python modules (e.g. abc, collections, tkinter)

an EKL binding managing:

the automatic loading of the suitable EKL packages

the interoperability with EKL types
an access to all EKL functions and methods
interactions with the sessions features (relations and geometries in the specification tree, PLM objects)
extensions to external packages (e.g. Numpy, Maplotlib)
the capability to be invoked from EKL.

Syntax
Inside the Python editor, the syntax to use is the Python syntax:

comment (# and not // or /* */)
indentation is used to define blocks for conditions or loop
no variable declaration
The EKL native Types and Functions are integrated in the python syntax thanks to the EKL binding.
You can find more information in dedicated sections on this page.

Sample
The Python script below is computing the factorial:

```
#factorial method
def factorialf(f):
    prod=1
    for n in range(2,f+1):
        prod *= n
    return prod

#test computation
input_set = [1,4,5,8]
for item in input_set:
    fact = factorialf(item)
    print("Factorial of ",item," is ", fact)
```

EKL Variables in Python
The EKL language is a static and strongly typed language. All the variables have to be declared by using the keyword let.

However, as Python is dynamically typed language, variable declaration is not necessary.

The EKL script below is working with a Pad as input stored in the variable named myPad.

```
let name(String)
name = myPad.Name
Notify("Selected pad : #",name)
```

Its Python counterpart is :

```
name = myPad.Name
ekl.Notify("Selected pad : #",name)
```

EKL Types in Python

Python EKL type

The EKL language is based on two dictionaries: a type and a function dictionary. To generate their definition inside Python, the binding has kept this architecture and has created two Python namespaces: first one, ekl, stores all functions and the second one, ekl.types, stores all type definitions.

Let XXX be an EKL type, the associated Python object ekl.types.XXX is a class with the same inheritance that the EKL types have and with its EKL attributes and methods.

The Python type of the Python object ekl.type.XXX is: ‹class ekl.XXX›

Specific types

Basic types

The basic types are implicitly converted. Boolean, Integer, Real, String can be used as arguments of the EKL methods/functions without converting them in their EKL version.

Dimensional types

The dimension values have a specific construction. All the types of dimensions available in EKL as Length, Angle, Area, Volume are also available in Python thanks to the binding.

To illustrate, you will find below a EKL script as example:

```
let dim(LENGTH)
dim = 40mm
Notify("Length in mm: #",dim)
Notify("Length in cm: #",DimToString(dim,"cm"))
```

and its Python counterpart:

```
dim = ekl.types.Length(40,"mm")
ekl.Notify("Length in mm: #",dim)
ekl.Notify("Length in cm: #",ekl.DimToString(dim,"cm"))
```

Care must be taken for all EKL Functions/Methods that use dimensions as arguments as the units are not automatically detected as they are in EKL. For instance, the following EKL script:

```
let pt(Point)
pt = point(10mm,20mm,30mm)
let x,y,z(Length)
pt.coord(x,y,z)
Notify("Point : (#,#,#)",x,y,z)
```

has to be translated into Python either as follows:

```
xx=ekl.types.Length(10,"mm")
yy=ekl.types.Length(20,"mm")
zz=ekl.types.Length(30,"mm")
pt=ekl.point(xx,yy,zz)
x=ekl.types.Length(0,"mm")
y=ekl.types.Length(0,"mm")
z=ekl.types.Length(0,"mm")
pt.coord(x,y,z)
ekl.Notify("Point : (#,#,#)",x,y,z)
```

or as follows:

```
pt=ekl.point(0.01,0.02,0.03)
x=ekl.types.Length(0,"mm")
y=ekl.types.Length(0,"mm")
z=ekl.types.Length(0,"mm")
pt.coord(x,y,z)
ekl.Notify("Point : (#,#,#)",x,y,z)
```

List

The Python standard library provides several types of collections : list, array, tuple, dictionary.

On the other hand, the EKL language provides an unique collection called List that can handle lists, arrays and structures.

All the methods available in EKL to fill and manipulate collections such as GetItem, IndexOf were bound in Python as in the following Python script:

```
list=ekl.types.List([1,'test',5])
size=list.Size()
print(f"list.Size()={size}")
print(f"len(list)={len(list)}")
list[0]=list[0]+10
num=list[0]+10
list.SetItem(num,1)
list.Append(42)
index=list.IndexOf('test',1)
list.RemoveItem(index)
size = list.Size()
for ind in range(1,int(size+1)):
    print(f"elem {ind} : {list.GetItem(ind)}")
```

EKL Functions/Methods in Python

To understand how to call the function, first we need to know the difference between a function and a method:

A method is applied to the objects of the class they belong to. EKL and Python syntax are the same in that context

A function is not associated with any object and can be invoked just with its name. The EKL binding stores the EKL functions in the namespace ekl. So in the Python editor, the name of the EKL has to be preceded by the prefix "ekl.".

All the functions/methods available in EKL are available in the Python editor, like Message, Notify, GetEditorRoots, etc.

To illustrate, you will find below an EKL script with as input a Point stored in the variable named myPoint.

```
let xCoord(LENGTH)
xCoord = myPoint.coord(1)
Notify("#[1] = #",myPoint.Name,xCoord)
```

and its Python counterpart :

```
xCoord = myPoint.coord(1)
ekl.Notify("#[1] = #",myPoint.Name,xCoord)
```

New Python specific APIs

New API for the geometry validity: is_null

In the EKL actions, the user regularly navigate into product structure to retrieve a particular instance.

In that case, the user has to check at each step the validity of the returned object.

In the EKL language, the validity is performed by the following test : obj <> NULL

Due to the binding ,this check is not valid in Python. Instead a new function called is_null has been added to compensate it.

In the following example, the Param rep is specified as an VPMReference.

```
1    let geoSet (OpenBodyFeature)
2    geoSet = rep.Find("OpenBodyFeature", "x.Name=='EKL_GeoSet'", true)
3    if geoSet <> NULL {
4    geoSet.Delete()
5    Notify("EKL_GeoSet has been deleted") }
```

and its Python counterpart :

```
1    geoSet = rep.Find("OpenBodyFeature", "x.Name=='EKL_GeoSet'", True)
2    if not ekl.is_null(geoSet):
3        geoSet.Delete()
4        print("EKL_GeoSet has been deleted")
```

New API for the geometry creation: build_from
To create a geometry inside the part, the associated script can be split as follows:

declare a variable with type of the new geometry
create an occurence of the object
instantiate it via affecting the occurence to the new variable

In the EKL language, the instantiation is performed implicitly during the variable affectation.

In Python, the user has to define it explicitly by the method build_from.

In the following EKL script, the Param geoSet is specified as a Feature.

```
1    let pt1(Point)
2    pt1 = new("Point", "Pt1", geoSet)
3    pt1 = point(0mm,100mm, 0mm)
```

Its Python counterpart is :

```
pt1=ekl.new("Point","Pt1",geoSet)
pt1.build_from(ekl.point(0,0.100,0))
```

New API for the type import: import_type
Inside the Python editor, the EKL binding manages the automatic loading of the suitable EKL packages.

In some cases though, there may be ambiguities. To solve them, the user can explicitly load the associated library of a type by using the new function import_type.

The type and its associated EKL package will be loaded and defined in Python namespaces ekl and ekl.types.

```
ekl.import_type("Elec3DCable")
is_elec=Feat._IsSupporting("Elec3DCable")
print(f"is_elec={is_elec}")
```

Execution from an EKL Action
In the relations as the Actions/Reactions, the execution of a Python script can be implemented by applying the Run operator and specifying any input or output arguments to the Python script in the EKL Relation. In that case, the execution of the relation launches the execution of the included Python script. Notice that this is not possible from a VB script. In contrast, EKL or VB scripts cannot be run from a Python script.

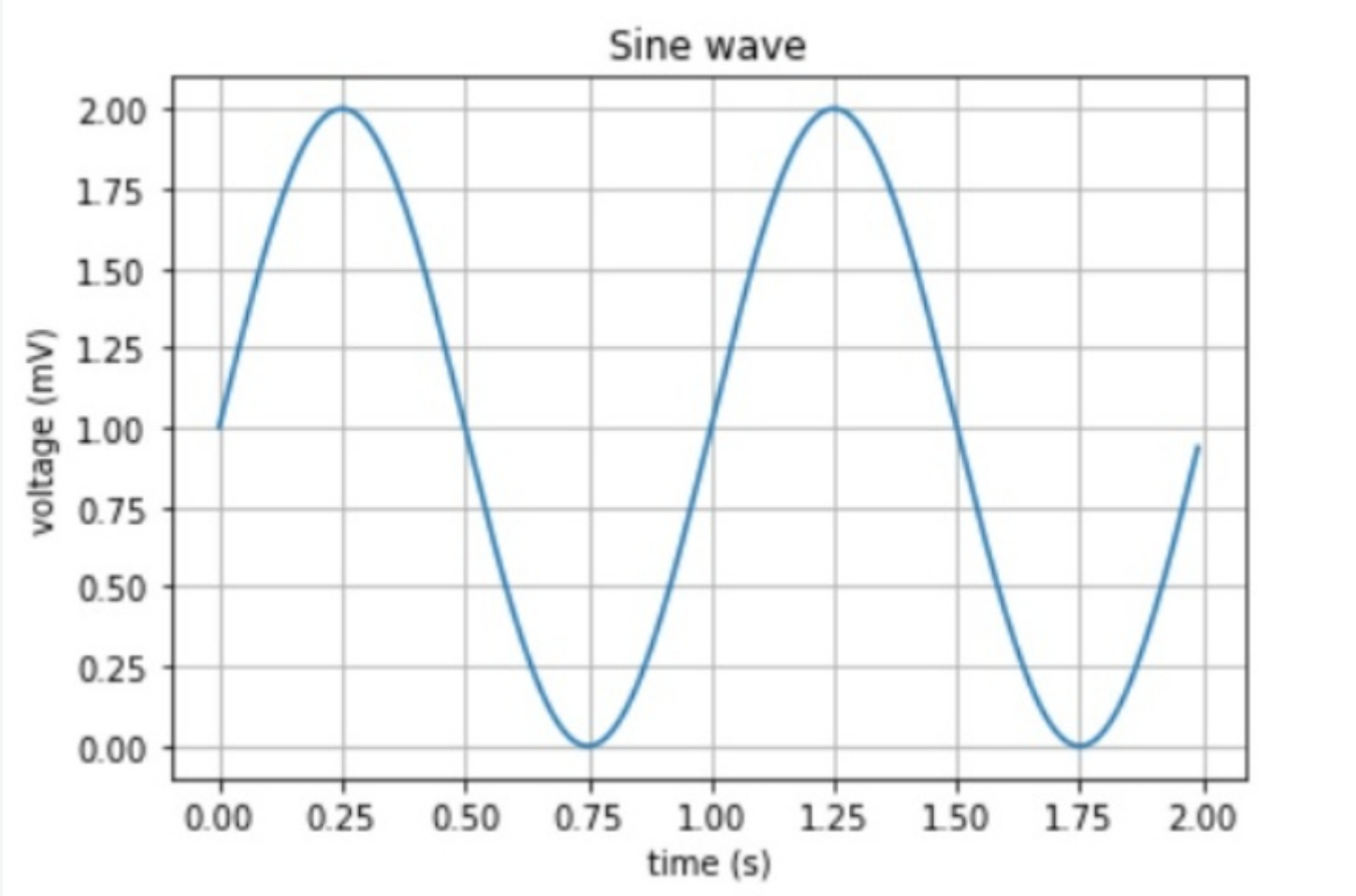
Let "Python Script.1" be a Python Script aggregated to the Relation Set "Relations"

```
'Relations\Python Script.1'->Run()
```

External Python package
To use external Python libraries the user has to:

Allow referencing external libs
Specify the location of the file directory in the python scripts section of the preference pages : 3D Modeling > Multidiscipline Automated Engineering > Basics
The Python script below is plotting a Sine wave using Matplotlib and Numpy.

```python
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2*np.pi*t)
plt.plot(t, s)
plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('Sine wave')
plt.grid(True)
plt.show()
```



Quick notes on installing external libs

One needs to take care to install the compatible version of the library with the Python interpreter provided in 3DEXPERIENCE. To know the current python version, use print(sys.version) in the editor. To install Python packages into a specified directory using pip install, the -t, --target option can be used. By default this will not replace existing files/folders in the specified directory. Add the -U, --upgrade option to replace existing packages with new versions.

Detailed instructions

(Thanks to @Thomas PERRET & @Guillaume VIRY for this information.)

Prerequisite

In order to install additional Python packages to be used in scripts, you will need to have a separate Python environment. The one coming with the 3DEXPERIENCE is partial and does not come with all the required tools.

There is no real constraint, the environment can be installed via the Microsoft Store, the official Python installer, or a tier Python distribution like Anaconda or Miniforge.

From a terminal (cmd.exe, or PowerShell), you need to launch the pip command, bundled with Python.

Here we will for instance install the “pandas” package in a specific folder.

If your Python environment is 3.7.x (recommended), you can do:

“path/to/pip.exe” install pandas –t “path/to/my/folder”

If it has a different version (like 3.10.x), you need to specify the target intended version (3.7 in R2022x FD01), and probably force binary packages

“path/to/pip.exe” install pandas –t “path/to/my/folder” –-python-version “3.7” --only-binary=:all:

You will see that the requested package and its dependencies have been installed in your folder.

Extra Tips

To check the python version of CATIA (3.7.9 as of 2022x FD01), run a Python Script in Engineering Rules Capture with:

print(sys.version)

If you already have a Python 3.7.x environment setup on your device, it is probably a good idea to use its external package directory

Disclaimer

If you change of device, your python scripts may not work anymore. You will need to install or copy the necessary packages.

Temporary Limitations

Even though, the EKL integration to Python is almost complete in R2022x FD01, two restrictions exist:

the access to the Local and Data Setup resources
the EKL variable initialization in Python.
Both will be fixed in FD02.

Limitation 1 :

In FD01, the vast majority of EKL functions are integrated to Python (Geometry/ Measure/Search functions, Message and Notify functionalities, Product Structure/Parameter management). However,

aspects related to Local and Data Setup resources are not yet integrated. i.e. the following capabilties are not available yet :

add Python Script features to local resource tables or to Data Setup tables.
have access from another script or action to a Python Script included in a local resource table or to a Data Setup table.
have access from a Python script to objects from local resource tables or Data Setup tables.
For example, Template Instantiation from a python script will not be possible in FD01.

```
root = ekl.GetEditorRoots("VPMReference").GetItem(1)
#Resource table replace
root.ManageInstance("ToReplace", "", "ToReplace.1")
#Search and instantiate
myQuery = ekl.CreatePLMQuery("VPMReference")
myQuery.AddCriterion("V_Name","Hub_TopPlate*")
results = myQuery.RunQuery()
currentResult = results.GetItem(1)
currentResultLoaded = currentResult.LoadResult()
tempInstance = ekl.new("VPMInstance", "", root, currentResultLoaded)
```

Limitation 2 :

In Python, a variable doesn’t exist unless it has a value, so the variable is only created the first time a value is assigned to it.

In FD01, unless you know how to explicitly initialize an EKL object, you can’t create this variable and use it as an argument in a function later.

From FD02, if the user doesn’t want or know how to preinitialize its value, the can use the keyword None.

In the following example, the input ref is specified as an VPMReference.

```
path = ""
repRef = ref.Find("VPMRepReference", "x.Name like '3D Shape*'", true)
matName = repRef.GetAttributeString("Material")
mat = ekl.AccessResource(matName, "CATMATReference")
matCnx=None
ekl.SetMaterialCore(ref,path ,mat ,matCnx )
```

To help users implement Python Scripts with EKL objects / APIs, in R2022x FD02, a Getting Started guide and the EKL Language Browser will be available directly from the Python Editor.

EKL scripts and their equivalents in Python
For reference, here are some basic examples of EKL scripts for different use cases and their equivalent Python versions.

Basic Object Search and Attribute Modification

EKL

```
let root, curRef(VPMReference)
root = GetEditorRoots("VPMReference").GetItem(1)

let armList (list)

armList = root.Query("VPMReference", "x.Name like 'Arm*'")

let curRep (VPMRepReference)

for curRef inside armList
{
        curRef.V_description = "This description is changed at " + DateFormat("%c")
        curRep = curRef.Find("VPMRepReference", "x.IsASortOf('3DShape')", true)
        if curRep.HasAttribute("Date") == True { curRep.SetAttributeString("Date", DateFormat("%c")) }
}
```

Python

```
# BasicObjectAndAttributeSearch
root = ekl.GetEditorRoots("VPMReference").GetItem(1)

armList = root.Query("VPMReference", "x.Name like 'Arm*'")
for curRef in armList :
    curRef.V_description = "This description is changed at "+str(ekl.DateFormat("%c"))
    curRep = curRef.Find("VPMRepReference", "x.IsASortOf('3DShape')", True)

    if curRep.HasAttribute("Date"):
        curRep.SetAttributeString("Date", ekl.DateFormat("%c"))
```

Create and Remove Instances

EKL

```
let root, newPrt (VPMReference)
let oldInst, newInst (VPMInstance)
root = GetEditorRoots("VPMReference").GetItem(1)
oldInst = root.Find("VPMInstance","x.Name like 'NewPrd*'", true)
if oldInst <> NULL
{
        Message("Old Instance: "+oldInst.Name + " deleted !")
        root.RemoveInstance(oldInst.Name)
}
newInst = new("VPMReference", "NewPrd_"+DateFormat("%c"), root)
newPrt = new("3DPart", "NewPart_" + DateFormat("%c"), newInst.Reference)
```

Python

```
# CreateAndRemoveInstances :
root = ekl.GetEditorRoots("VPMReference").GetItem(1)
oldInst = root.Find("VPMInstance","x.Name like 'NewPrd*'", True)
if not ekl.is_null(oldInst):
    ekl.Message("Old Instance: # deleted !",oldInst.Name)
    root.RemoveInstance(oldInst.Name)
newInst = ekl.new("VPMReference", "NewPrd_"+str(ekl.DateFormat("%c")), root)
newPrt = ekl.new("3DPart", "NewPart_"+str(ekl.DateFormat("%c")), newInst.Reference)
```

Get and Set Parameter values

EKL

```
let root, geoRef (VPMReference)
root = GetEditorRoots("VPMReference").GetItem(1)
geoRef = root.Find("VPMReference", "x.Name == 'EKL_Geometry'", true)
let rep (VPMRepReference)
rep = geoRef.Find("VPMRepReference", "", true)
//to test finding the owner
set geoRef = rep.AggregatingReference
geoRef.V_description = DateFormat("%c")
//create parameter set
let advParamSet (AdvisorParameterSet)
advParamSet  = rep.Find("AdvisorParameterSet", "", true)
advParamSet = new("AdvisorParameterSet", "ParamSet_"+DateFormat("%c"), advParamSet)
//create parameter
advParamSet.SetAttributeDimension("Length1", 0.01, "Length")
//value pointer
let vp, vp2 (ValuePointer)
vp = advParamSet.GetAttributeValuePointer("Length1")
vp.AuthorizedValues = List (10mm, 20mm, 30mm)
```

Python

```
#Parameters
root = ekl.GetEditorRoots("VPMReference").GetItem(1)
geoRef = root.Find("VPMReference", "x.Name == 'EKL_Geometry'", True)
rep = geoRef.Find("VPMRepReference", "", True)
print(f"[Parameters] rep= {rep.Name}")
#to test finding the owner
geoRef = rep.AggregatingReference
geoRef.V_description = ekl.DateFormat("%c")
#create parameter set
advParamSet  = rep.Find("AdvisorParameterSet", "", True)
advParamSet = ekl.new("AdvisorParameterSet", "ParamSet_"+str(ekl.DateFormat("%c")), advParamSet)
print(f"[Parameters] advParamSet= {advParamSet.Name}")
#create parameter
advParamSet.SetAttributeDimension("Length1", 0.01, "Length")
#value pointer
vp = advParamSet.GetAttributeValuePointer("Length1")
dim1=ekl.types.Length(10,"mm")
dim2=ekl.types.Length(20,"mm")
dim3=ekl.types.Length(40,"mm")
auth_val = ekl.types.List([dim1, dim2, dim3])
if not ekl.is_null(vp):
  vp.AuthorizedValues = auth_val
  vp.Value = dim2
```

Create Geometry

EKL

```
let root, geoRef (VPMReference)
root = GetEditorRoots("VPMReference").GetItem(1)
geoRef = root.Find("VPMReference", "x.Name == 'EKL_Geometry'", true)
let rep (VPMRepReference)
rep = geoRef.Find("VPMRepReference", "", true)
let partF (PartFeature)
partF = rep.Find("PartFeature", "", true)

let geoSet (OpenBodyFeature)
geoSet = rep.Find("OpenBodyFeature", "x.Name=='EKL_GeoSet'", true)
if geoSet <> NULL { geoSet.Delete() }
geoSet = new("OpenBodyFeature", "EKL_GeoSet", partF)

let pt1, pt2 (Point)
pt1 = new("Point", "Pt1", geoSet)
pt1 = point(0mm,100mm, 0mm)
pt2 = new("Point", "Pt2", geoSet)
pt2 = point(100mm,100mm, 0mm)

let l (line)
l = new("Line", "Line1",geoSet)
l = line(pt1, pt2)

let dist (length)
dist = distance(pt1, pt2)

let s, v (surface)
s = new("Surface", "Surface1", geoSet)
s = extrude(l, direction(0,0,1), 0mm, dist, true)
s.Color = "Blue"

v = new("VolumeGeo", "Vol1", geoSet)
set v = extrude(s, direction(0,1,0), 0mm, dist, true)
v.Color = "Green"
v.Transparency = 100

let surfaceArea (AREA)
let vol (volume)
surfaceArea = area(s)
vol = volume(v)

rep.Update()

Message("Area of the surface = #|Volume of the solid = #", surfaceArea, vol)
```

Python

```
root = ekl.GetEditorRoots("VPMReference").GetItem(1)
geoRef = root.Find("VPMReference", "x.Name == 'EKL_Geometry'", True)
rep = geoRef.Find("VPMRepReference", "", True)
partF = rep.Find("PartFeature", "", True)

geoSet = rep.Find("OpenBodyFeature", "x.Name=='EKL_GeoSet'", True)
if not ekl.is_null(geoSet) :
  geoSet.Delete()
else :
  geoSet = ekl.new("OpenBodyFeature", "EKL_GeoSet", partF)

pt1 = ekl.new("Point", "Pt1", geoSet)
pt1.build_from(ekl.point(0,0.100, 0))
pt2 = ekl.new("Point","Pt2", geoSet)
pt2.build_from(ekl.point(0.100,0.100, 0))

l = ekl.new("Line", "Line1",geoSet)
l.build_from(ekl.line(pt1, pt2))
dist = ekl.distance(pt1, pt2)

s = ekl.new("Surface", "Surface1", geoSet)
s.build_from(ekl.extrude(l, ekl.direction(0,0,1), 0, dist, True))
s.Color = "Blue"

v = ekl.new("VolumeGeo", "Vol1", geoSet)
v.build_from(ekl.extrude(s, ekl.direction(0,1,0), 0, dist, True))
v.Color = "Green"
v.Transparency = 100

surfaceArea = ekl.area(s)
vol = ekl.volume(v)
rep.Update()
ekl.Message("Area of the surface = #|Volume of the solid = #", surfaceArea, vol)
```

Python Learning Resources

Python is a hugely popular language and it is very easy to find learning resources and answers to many questions online. It is also possible to find many courses like on LinkedIn Learning, Udemy, etc.

The Python Essential Training course by Bill Weinman is a good one to quickly get familiar with the language.

"Think Python" - well written and pretty thorough in covering different aspects of Python. The PDF is freely available here Think Python .