solidworks_python_lynkoa

November 15, 2019

0.1 Importing modules

[1]:
```
import win32com.client as win32
import pythoncom
import os
import pandas as pd
import re
```

0.2 Initializing application

[2]:
```
sw = win32.dynamic.Dispatch('SldWorks.Application')
```

0.3 Initializing custom Solidworks/Com32 var type

[3]:
```
VT_BYREF = win32.VARIANT(pythoncom.VT_BYREF | pythoncom.VT_I4, -1)
```

[4]:
```
VT_DISPATCH = win32.VARIANT(pythoncom.VT_DISPATCH, None)
```

Ressources : * https://joshuaredstone.blogspot.com/2015/02/solidworks-macros-via-python.html * https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.varenum?view=n 4.8

0.4 Creating new document

[3]:
```
# part
folder = 'C:\\ProgramData\\SolidWorks\\SOLIDWORKS 2019\\templates\MBD\\'
name = 'part.prtdot'
filename = os.path.join(folder, name)

model = sw.NewDocument(filename, 0, 0, 0) # params: template, ?, sheet_width,␣
 ↪sheet_height
```

[4]:
```
# assembly
folder = 'C:\\ProgramData\\SolidWorks\\SOLIDWORKS 2019\\templates\MBD\\'
name = 'assembly.asmdot'
filename = os.path.join(folder, name)
```

```
model = sw.NewDocument(filename, 0, 0, 0) # params: template, ?, sheet_width,␣
 ↪sheet_height
```

[32]:
```
# drawing
folder = 'C:\\ProgramData\\SolidWorks\\SOLIDWORKS 2019\\templates\MBD\\'
name = 'drawing.drwdot'
filename = os.path.join(folder, name)

model = sw.NewDocument(filename, 0, 0, 0) # params: template, ?, sheet_width,␣
 ↪sheet_height
# the parameters sheet_width & sheet_height are only usefull if using a␣
 ↪template with no
# predefined sheet sizes
```

Ressources: * NewDocument

0.5 Opening a document

[19]:
```
open_spec = sw.GetOpenDocSpec('cube.SLDPRT')

doc = sw.OpenDoc7(open_spec)
```

Ressources: * GetOpenDocSpec * IDocumentSpecification Interface * OpenDoc7

0.6 Retreiving an opened document

[5]:
```
# create a part and a document first

doc = None

opened = list()
meta = dict()

doc = sw.GetFirstDocument

while doc != None:
    meta['name'] = doc.GetTitle
    meta['type'] = doc.GetType
    meta['object'] = doc
    opened.append(meta)
    doc = doc.GetNext

print(opened)
```

```
[{'name': 'peigne_decoupe_laser.SLDPRT', 'type': 1, 'object': <COMObject
<unknown>>}]
```

[31]:
```
sw.ActivateDoc3('Pièce1', True, 0, VT_BYREF)
```

[31]:
```
<COMObject <unknown>>
```

Ressources : * ActivateDoc3

0.7 Creating a sketch

[41]:
```
swDoc = sw.ActiveDoc
boolstatus = swDoc.Extension.SelectByID2("Plan de face", "PLANE", 0, 0, 0,␣
 ↪False, 0, VT_DISPATCH, 0)
sketchMan = swDoc.SketchManager

sketchMan.InsertSketch(True)

swDoc.ClearSelection2(True)

# Create a sketch with "IModelDoc2.SketchManager.InsertSketch"
```

Ressources: * SelectByID2 * 'Name', 'Type', 'X', 'Y', 'Z', 'Append', 'Mark', 'Callout', 'SelectOption' * ClearSelection2

0.8 Drawing a line

[42]:
```
# Use "Creating a sketch"

d = 5/1000 # Units in meters

sketchMan.CreateLine(d, d, 0, -d, d, 0)
sketchMan.CreateLine(-d, d, 0, -d, -d, 0)
sketchMan.CreateLine(-d, -d, 0, d, -d, 0)
sketchMan.CreateLine(d, -d, 0, d, d, 0)

swDoc.ClearSelection2(True)
```

[44]:
```
swDoc.ViewZoomtofit2()
```

Ressources: * CreateLine * 'Xstart', 'Ystart', 'Zstart', 'Xstop', 'Ystop', 'Zstop' * ClearSelection2 * ViewZoomtofit2

0.9 Applying a function to a sketch

[46]:
```
start_cond = {
    'swStartOffset': 3,
    'swStartSketchPlane': 0,
    'swStartSurface': 1,
    'swStartVertex': 2,
}
```

[47]:
```
stop_cond = {
    'swEndCondBlind': 0,
```

```
    'swEndCondMidPlane': 6,
    'swEndCondOffsetFromSurface': 5,
    'swEndCondThroughAll': 1,
    'swEndCondThroughAllBoth': 9,
    'swEndCondThroughNext': 2,
    'swEndCondUpToBody': 7,
    'swEndCondUpToNext': 11,
    'swEndCondUpToSelection': 10,
}
```

[52]:
```
# Use "Creating a sketch" & "Drawing a line" first

featMan = swDoc.FeatureManager

featMan.FeatureExtrusion3(True, False, False, stop_cond['swEndCondMidPlane'],␣
↪stop_cond['swEndCondMidPlane'],
                          10/1000, 0, False, False, False, False, 0, 0, False,␣
↪False, False, False, True, False, True,
                          start_cond['swStartSketchPlane'], 0, False)
```

[52]:
```
<COMObject <unknown>>
```

Ressources: * FeatureExtrusion3 * swStartConditions_e * swEndConditions_e

```
Function FeatureExtrusion3( _
   ByVal Sd As System.Boolean, _
   ByVal Flip As System.Boolean, _
   ByVal Dir As System.Boolean, _
   ByVal T1 As System.Integer, _
   ByVal T2 As System.Integer, _
   ByVal D1 As System.Double, _
   ByVal D2 As System.Double, _
   ByVal Dchk1 As System.Boolean, _
   ByVal Dchk2 As System.Boolean, _
   ByVal Ddir1 As System.Boolean, _
   ByVal Ddir2 As System.Boolean, _
   ByVal Dang1 As System.Double, _
   ByVal Dang2 As System.Double, _
   ByVal OffsetReverse1 As System.Boolean, _
   ByVal OffsetReverse2 As System.Boolean, _
   ByVal TranslateSurface1 As System.Boolean, _
   ByVal TranslateSurface2 As System.Boolean, _
   ByVal Merge As System.Boolean, _
   ByVal UseFeatScope As System.Boolean, _
   ByVal UseAutoSelect As System.Boolean, _
   ByVal T0 As System.Integer, _
   ByVal StartOffset As System.Double, _
   ByVal FlipStartOffset As System.Boolean _
) As Feature
```

0.10 All features

IFeatureManager Interface Members

0.11 Creating a BOM from the feature manager

[56]:
```
panes = {
    'swFeatMgrPaneBottom': 1,
    'swFeatMgrPaneBottomHidden': 3,
    'swFeatMgrPaneFlyout': 4,
    'swFeatMgrPaneTop': 0,
    'swFeatMgrPaneTopHidden': 2,
}
```

[132]:
```
def traverse_node(node, index=1):

    if not node == None:

        if node.ObjectType == 2:
            with open('out.txt', 'a') as f:
                f.write(index * '---' + node.Text + f' type{node.ObjectType} '␣
 ↪+ '\n')

        sub_node = node.GetFirstChild
        if sub_node != None:
            traverse_node(sub_node, index+1)

        node = node.GetNext
        if node != None:
            traverse_node(node, 1)
```

[133]:
```
swDoc = sw.ActiveDoc
featMan = swDoc.FeatureManager

root_item = featMan.GetFeatureTreeRootItem2(panes['swFeatMgrPaneBottom'])
with open('out.txt', 'w') as f:
    f.write(root_item.Text + '\n')
traverse_node(root_item.GetFirstChild)
print('finished!')
```

```
finished!
```

Ressources: * GetFeatureTreeRootItem2 * ITreeControlItem * ITreeControlItem Interface Members * swFeatMgrPane_e * Traverse FeatureManager Design Tree Example (VBA)

0.12 Exporting BOM as Pandas DataFrame

[63]:
```
panes = {
    'swFeatMgrPaneBottom': 1,
    'swFeatMgrPaneBottomHidden': 3,
    'swFeatMgrPaneFlyout': 4,
    'swFeatMgrPaneTop': 0,
    'swFeatMgrPaneTopHidden': 2,
}
```

[64]:
```
def get_children(bom, parent, level):
    local_level = level + 1
    elem = parent.GetFirstChild
    while elem != None:
        if elem.ObjectType == 2:
            bom.append({
                'level':local_level,
                'quantity':1,
                'reference':elem.text,
                'revision':'',
                'designation':'',
                'provider':'',
            })
        if not elem.text in ('Historique', 'Capteurs', 'Annotations'):
            get_children(bom, elem, local_level)
        elem = elem.GetNext
```

[65]:
```
swDoc = sw.ActiveDoc
featMan = swDoc.FeatureManager

root_item = featMan.GetFeatureTreeRootItem2(panes['swFeatMgrPaneBottom'])

bom = [{
    'level':0,
    'quantity':1,
    'reference':root_item.text,
    'revision':'',
    'designation':'',
    'provider':'',
}]

get_children(bom, root_item, 0)

print('Finished!')
```

```
Finished!
```

[66]:
```
df = pd.DataFrame(bom)
```

[80]:
```
df.to_csv('bom_out.csv')
```

Ressources:

- ..

0.13 Parsing BOM as class

[5]:
```
panes = {
    'swFeatMgrPaneBottom': 1,
    'swFeatMgrPaneBottomHidden': 3,
    'swFeatMgrPaneFlyout': 4,
    'swFeatMgrPaneTop': 0,
    'swFeatMgrPaneTopHidden': 2,
}
```

[30]:
```
class bom():

    prefix = None
    quantity = None
    level = None
    reference = None
    revision = None
    designation = None
    provider = None

    parent = None
    sub_parts = None

    def __init__(self, ft_element, parent=None):
        self.quantity = 1
        if ft_element.text:
            self.reference = ft_element.text
        if parent:
            self.parent = parent
        self.level = self.get_level()
        self.sub_parts = list()
        child = ft_element.GetFirstChild
        while child != None:
            if child.ObjectType == 2 or 'Répétition' in child.text:
                self.sub_parts.append(bom(child, self))
            child = child.GetNext

    def __str__(self):
        s = ''
        s += f'|{"level":^5}|{"qty":^5}|{"reference":^50}|\n'
        s += f'|{"-"*5}|{"-"*5}|{"-"*50}|\n'
        s += self.str_element()
```

```
        s += f'|{"-"*5}|{"-"*5}|{"-"*50}|\n'
        return(s)

    def str_element(self):
        s = ''
        s += f'|{self.level:^5}|{self.quantity:^5}|{self.reference[:50]:50}|\n'
        for p in self.sub_parts:
              s += p.str_element()
        return s

    def get_level(self):
        current_parent = self.parent
        level = 0
        while current_parent != None:
            level += 1
            current_parent = current_parent.parent
        return level

    def pack(self):
        new_sub_parts = dict()
        while len(self.sub_parts) > 0:
            part = self.sub_parts.pop()
            if part.reference in new_sub_parts.keys():
                new_sub_parts[part.reference].quantity += 1
            else:
                new_sub_parts[part.reference] = part
        self.sub_parts = new_sub_parts.values()
        for part in self.sub_parts:
            part.pack()

    def unpack(self):
        pass # TBDone

    def clean(self):
        new_sub_parts = list()
        while len(self.sub_parts) > 0:
            part = self.sub_parts.pop()
            if 'Répétition' in part.reference:
                new_sub_parts.append(part.sub_parts)
            else:
                new_sub_parts.append(part)
        self.sub_parts = new_sub_parts
        for part in self.sub_parts:
            part.level = part.get_level()
        patterns = [
            '\([^()]*\)$',
            '<[0-9]+> ?',
```

```
        ]
        for pattern in patterns:
              self.reference = re.sub(pattern, '', self.reference)
        for part in self.sub_parts:
              part.clean()
```

[31]:
```
swDoc = sw.ActiveDoc
featMan = swDoc.FeatureManager
ft_element = featMan.GetFeatureTreeRootItem2(panes['swFeatMgrPaneBottom'])
ft_bom = bom(ft_element)

print('Finished!')
```

```
Finished!
```

[ ]:
```
ft_bom.clean()
ft_bom.pack()
print(ft_bom)
```

Ressources:

- ..

0.14 Exporting each configuration as DXF (Not working yet)

[130]:
```
swDoc = sw.ActiveDoc

dimensions = range(1,11)
tolerances = [0.05, 0.1]
```

[136]:
```
part_name = swDoc.GetPathName
root, name = os.path.split(part_name)

Alignment = [0.0, 0.0, 0.0, # New origin
             1.0, 0.0, 0.0, # New x vector
             0.0, 0.0, -1.0, # New y vector
             0.0, -1.0, 0.0] # New normal vector

Views = ['*Dessus']

VR_Alignment = win32.VARIANT(pythoncom.VT_VARIANT, Alignment)
VR_Views = win32.VARIANT(pythoncom.VT_VARIANT, Views)

for dim in dimensions[0:1]:
    for tol in tolerances:
        name = f'peigne_ep{dim}_tol{str(tol).replace(".", "-")}.DXF'
        dxf_name = os.path.join(root, name)
        swDoc.ShowConfiguration2(f"ep-{dim}_tol-{tol}")
```

```
        status = swDoc.ExportToDWG2(dxf_name, part_name, 3, True, VR_Alignment,␣
 ↪False, False, 0, VR_Views)
        print(name + " .. " + ('Success' if status else 'Failure'))

print('Finished!')
```

```
peigne_ep1_tol0-05.DXF .. Failure
peigne_ep1_tol0-1.DXF .. Failure
Finished!
```

[112]:
```
status = swDoc.ExportToDWG2(dxf_name, part_name, 3, True, VR_Alignment, False,␣
 ↪False, 0, VR_Views)

print(status)
```

```
False
```

Ressources: * Solidworks Forum * ExportToDWG2 * StackOverFlow1

0.15 Exporting to STEP and PDF

[7]:
```
schema = r'^WOM_04[4-5]_.*\.SLD(DRW|PRT)$'
root = r'S:\320-Technologies_Microsystèmes\320.4-Packaging\320.4.
 ↪18-CarnotFlex\2- Realisation du projet\DSYS\WP4\Banc3D\01-Mecanique\Dossier␣
 ↪final'
```

[25]:
```
l = [x for x in os.listdir(root) if re.fullmatch(schema, x, re.IGNORECASE)]
print(l)
```

```
['WOM_044_A_adaptateur-capteur.SLDPRT', 'WOM_045_A_adaptateur-touche-
capteur.SLDDRW', 'WOM_044_A_adaptateur-capteur.SLDDRW', 'WOM_045_A_adaptateur-
touche-capteur.SLDPRT']
```

[26]:
```
if True: #if input('Voulez-vous traiter cette liste ?') == 'y':
    for file in l:
        filename, file_extension = os.path.splitext(file)
        if file_extension == '.SLDDRW':
            open_spec = sw.GetOpenDocSpec(os.path.join(root, file))
            swDoc = sw.OpenDoc7(open_spec)
            if not swDoc == None:
                swModelDocExt = swDoc.Extension
                swExportPDFData = sw.GetExportFileData(1)
                ERR_VT_BYREF = win32.VARIANT(pythoncom.VT_BYREF | pythoncom.
 ↪VT_I4, 0)
                WAR_VT_BYREF = win32.VARIANT(pythoncom.VT_BYREF | pythoncom.
 ↪VT_I4, 0)
```

```
                if not swModelDocExt.SaveAs2(os.path.join(root, f'{filename}.
↪PDF'), 0, 0, swExportPDFData, '', False, ERR_VT_BYREF, WAR_VT_BYREF):
                    print('Failure at saving')
                sw.QuitDoc(swDoc.GetTitle)
                swDoc = None
            else:
                print('Failure at opening')
        if file_extension == '.SLDPRT':
            open_spec = sw.GetOpenDocSpec(os.path.join(root, file))
            swDoc = sw.OpenDoc7(open_spec)
            if not swDoc == None:
                swModelDocExt = swDoc.Extension
                swExportSTEPData = sw.GetExportFileData(1)
                ERR_VT_BYREF = win32.VARIANT(pythoncom.VT_BYREF | pythoncom.
↪VT_I4, 0)
                WAR_VT_BYREF = win32.VARIANT(pythoncom.VT_BYREF | pythoncom.
↪VT_I4, 0)
                if not swModelDocExt.SaveAs2(os.path.join(root, f'{filename}.
↪STEP'), 0, 0, swExportSTEPData, '', False, ERR_VT_BYREF, WAR_VT_BYREF):
                    print('Failure at saving')
                sw.QuitDoc(swDoc.GetTitle)
                swDoc = None
            else:
                print('Failure at opening')
print('Finished!')
```

```
Finished!
```

Ressources:

- ...

0.16 Exiting application

[1]:
```
sw = None
```

[ ]:
```

```