Autodesk University 2012

Scripting Components for AutoCAD Plant 3D

David Wolfe – ECAD, Inc.

PD1746 In this class, we will introduce you to the programming language Python. You will learn how to set up a simple development environment and create a few basic scripts. Next, you will learn to install and test the scripts with AutoCAD Plant 3D software. Come and learn how to use scripting to create custom objects for use in AutoCAD Plant 3D.

Learning Objectives

At the end of this class, you will be able to:

- Explain what Python is and how you can use it
- Draw Components using Python Scripting
- Test and Use Python components in AutoCAD Plant 3D
- Describe steps for creating scripts

About the Speaker

David Wolfe has extensive experience customizing AutoCAD using Lisp, VBA, and .Net. He is a Process and Power Specialist with ECAD, Inc. and trains clients how to use and implement the AutoCAD Plant Design Suite. His experience helps him tailor AutoCAD installations to meet company standards and helps students get quickly up to speed using industry best practices.

What is Python?

Python is a functional programming language that is used to create Plant 3D components.
http://www.python.org/

Setting up a Development Environment

To develop scripts without impacting your installation of Plant 3D, I recommend setting up a virtual machine using virtual machine software like VirtualBox https://www.virtualbox.org/.

After setting up your development machine, or ignoring me and moving to the next step, you should select an IDE or integrated development environment. IDE's make developing easier by highlighting syntax, and providing other useful features. With the way Plant 3D is setup right now, debugging and other features can't be implemented, so the most you'll get is syntax highlighting and corrections. Right now, I use Komodo Edit https://www.activestate.com/komodo-edit because it's free.

When working with equipment scripts, we are going to be editing .peqx files. I recommend installing 7-Zip because it allows us to open the zip archive without extracting the contents – and it's free. http://7-zip.org/

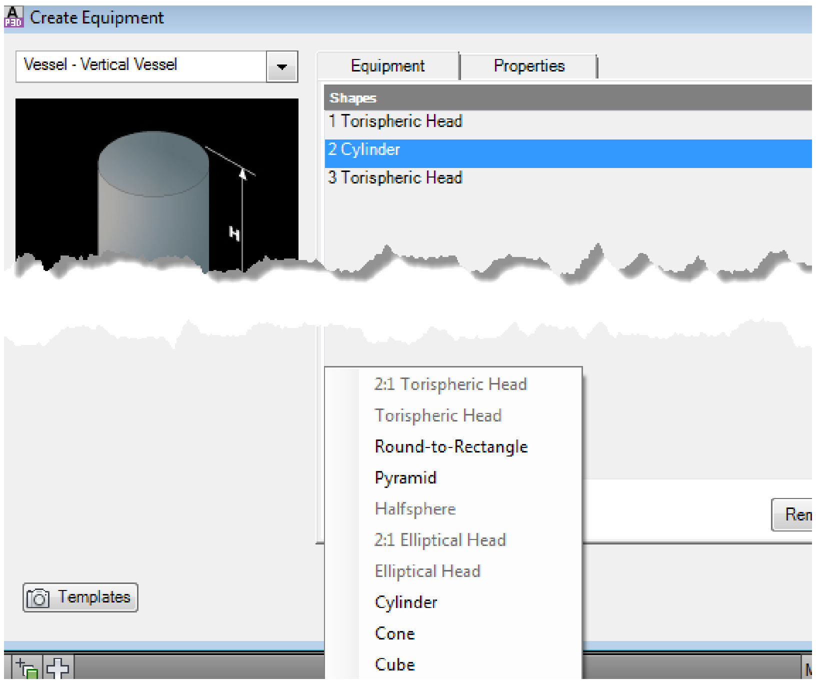
With xml files, I use Foxe - http://www.firstobject.com/dn_editor.htm which offers a customizable tree view of the xml beside the xml content with syntax highlighting.

Thanks to Carsten Beinecke (http://de.linkedin.com/pub/carsten-beinecke/27/964/8a7) and

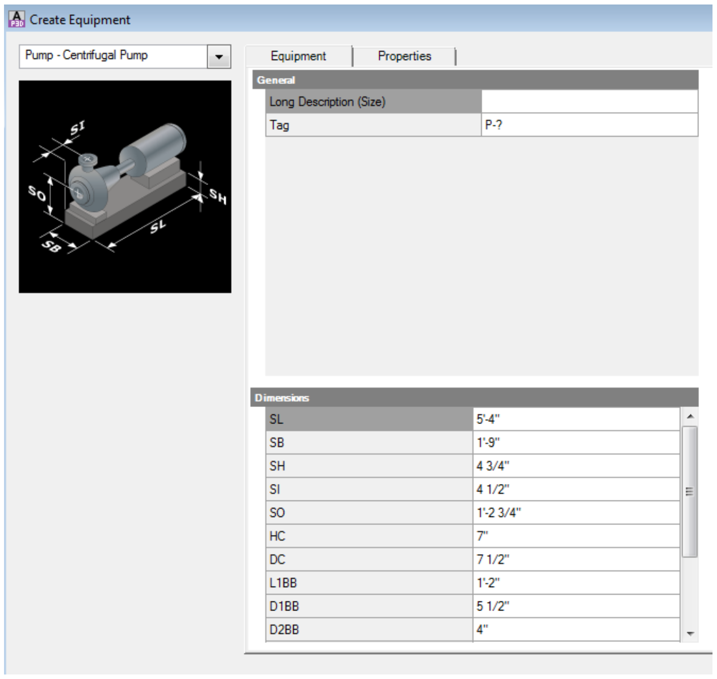
Felix Beer (http://at.linkedin.com/pub/felix-beer/11/308/581) for teaching me about scripts. I could not have gotten scripts running without their help.

Types of Python Scripts

Plant 3D uses scripts in several ways. The first way is for Equipment. In the Equipment dialog, you can build equipment using basic shapes. These basic shapes (primitives) are hard coded into the program, and you cannot add your own routines to this list.



Another type of script is for generating a complete equipment item. For example, the Centrifugal Pump script generates the entire pump, and the user inputs values for the parameters.



General

Long Description (Size)	
Tag	P-?

Dimensions

SL	5'-4"
SB	1'-9"
SH	4 3/4"
SI	4 1/2"
SO	1'-2 3/4"
HC	7"
DC	7 1/2"
L1BB	1'-2"
D1BB	5 1/2"
D2BB	4"

We can create scripts like this type to generate entire equipment pieces.

Script Locations

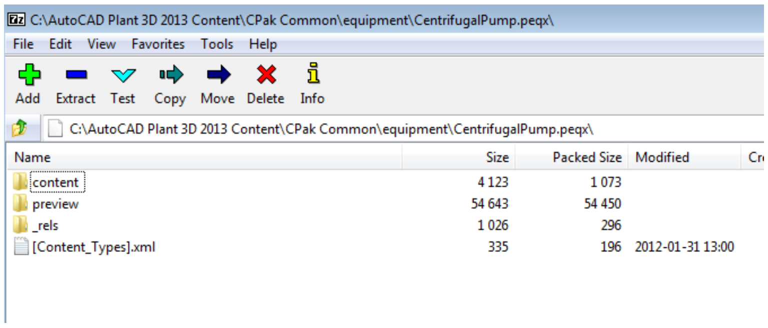
Plant 3D will load custom scripts from a specific location. By default that location is C:\AutoCAD Plant 3D 2013 Content\CPak Common\CustomScripts\. If you do not have this folder, you can create it in the CPak Common folder.

Equipment Script Format

For our equipment script we need to explore peqx files. First, a peqx is the template format for equipment. For our files that use scripts to build the equipment, it will contain dimension screenshots and an xml file that links the dimensions to the script parameters. The compiled scripts themselves will reside in the script location.

Equipment templates are installed under C:\AutoCAD Plant 3D 2013 Contnet\CPak Common\equipment.

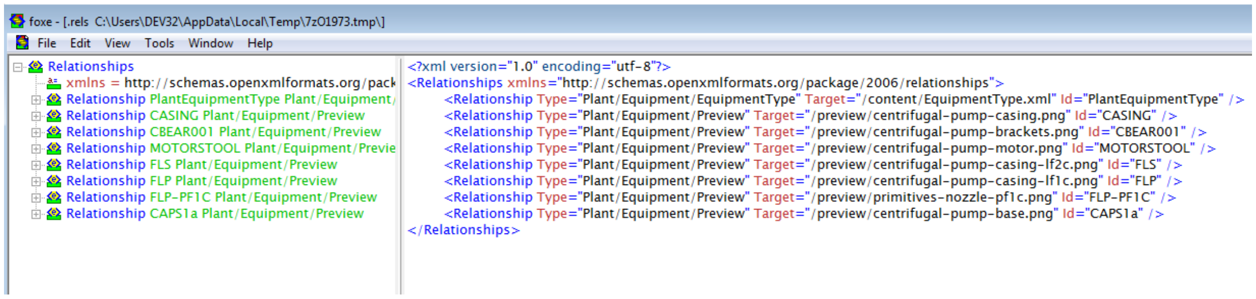
Use 7-Zip to explore the CentrifugalPump.peqx.


C:\AutoCAD Plant 3D 2013 Content\CPak Common\equipment\CentrifugalPump.peqx\
File Edit View Favorites Tools Help
Add Extract Test Copy Move Delete Info
C:\AutoCAD Plant 3D 2013 Content\CPak Common\equipment\CentrifugalPump.peqx\

Name	Size	Packed Size	Modified	Cr
content	4 123	1 073		
preview	54 643	54 450		
_rels	1 026	296		
[Content_Types].xml	335	196	2012-01-31 13:00	

The peqx files use a format introduced by Microsoft to structure zip files. The [Content_Types].xml can be read to let programs know what file types are used in the zip. The _rels folder contains a .rels file that describes relationships between files.

Preview images with dimensions are linked in the .rels. Preview images are 200 x 200.

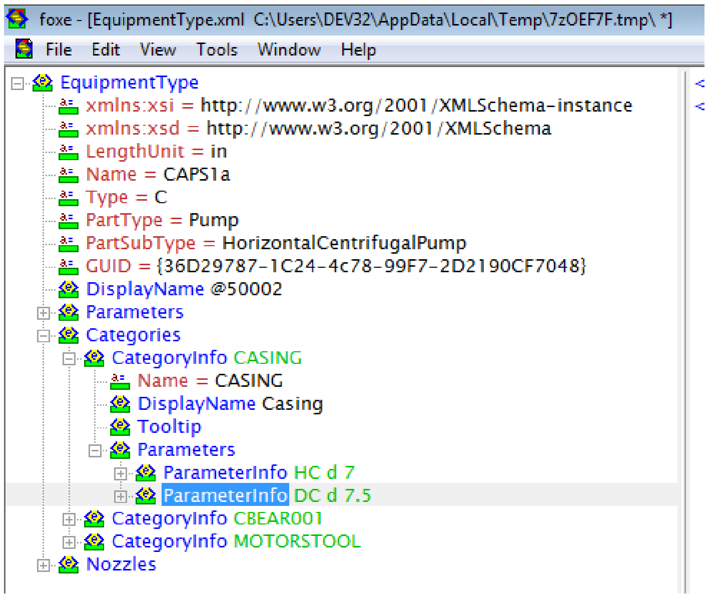

foxe - [.rels C:\Users\DEV32\AppData\Local\Temp\7zO1973.tmp\]
File Edit View Tools Window Help

```
Relationships
   xmlns = http://schemas.openxmlformats.org/pack
   Relationship PlantEquipmentType Plant/Equipment/
   Relationship CASING Plant/Equipment/Preview
   Relationship CBEAR001 Plant/Equipment/Preview
   Relationship MOTORSTOOL Plant/Equipment/Previe
   Relationship FLS Plant/Equipment/Preview
   Relationship FLP Plant/Equipment/Preview
   Relationship FLP-PF1C Plant/Equipment/Preview
   Relationship CAPS1a Plant/Equipment/Preview
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Type="Plant/Equipment/EquipmentType" Target="/content/EquipmentType.xml" Id="PlantEquipmentType" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-casing.png" Id="CASING" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-brackets.png" Id="CBEAR001" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-motor.png" Id="MOTORSTOOL" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-casing-lf2c.png" Id="FLS" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-casing-lf1c.png" Id="FLP" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/primitives-nozzle-pf1c.png" Id="FLP-PF1C" />
    <Relationship Type="Plant/Equipment/Preview" Target="/preview/centrifugal-pump-base.png" Id="CAPS1a" />
</Relationships>
```

Notice the relationship type are all the same. The Target refers to the location within the peqx and the Id refers to an element within the EquipmentType.xml file.

The EquipmentType.xml is located in the content folder.



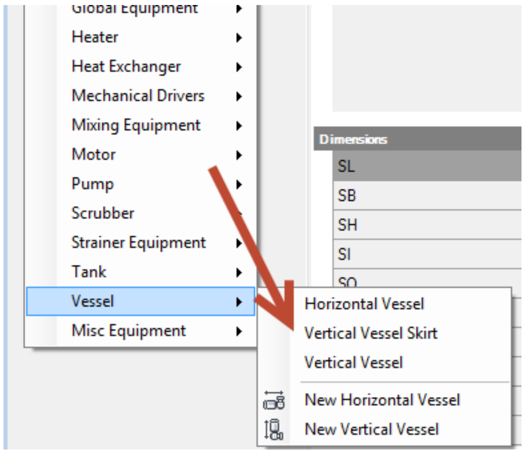
```
foxe - [EquipmentType.xml  C:\Users\DEV32\AppData\Local\Temp\7zOEF7F.tmp\ *]
File  Edit  View  Tools  Window  Help
EquipmentType
    xmlns:xsi = http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd = http://www.w3.org/2001/XMLSchema
    LengthUnit = in
    Name = CAPS1a
    Type = C
    PartType = Pump
    PartSubType = HorizontalCentrifugalPump
    GUID = {36D29787-1C24-4c78-99F7-2D2190CF7048}
    DisplayName @50002
    Parameters
    Categories
        CategoryInfo CASING
            Name = CASING
            DisplayName Casing
            Tooltip
            Parameters
                ParameterInfo HC d 7
                ParameterInfo DC d 7.5
        CategoryInfo CBEAR001
        CategoryInfo MOTORSTOOL
    Nozzles
```

The Name property under EquipmentType should match the name of the script we create. Type can have one of three possible values: C, F, I. Type F is for fabricated equipment like tanks that uses primitives only. Type I is for converted models that use block content. Type C is for custom equipment that is completely defined by a script.

The PartType and PartSubType define where the equipment fits within the menu and the class structure. I've not seen any documentation on where PartTypes are defined, so you'll have to examine existing peqx files and determine what matches your needs.

The Guid is a unique identifier for your script. You need to generate a new guid for each new type of equipment you create. You can generate guids using this site: http://www.guidgenerator.com/

The DisplayName element is the value that appears in the Equipment template menu:



For our parameters, we need to be aware of the available types:

d	Length or distance, cannot be less than or equal to 0.
d0	Length or distance, may be 0
d-	A linear offset, maybe less than, equal to, or greater than 0.
a	An angle – typically degrees (vs radians)
r	Dimensionless numbers – e.g. number of segments
b	Boolean – true or false
Button	Button with text, in addition to the type, specify a Data property with the button text. No idea how to hook into it.
List, Combo	Multi-value selection options with values separated with a bar "|" like Date="item1|item2|item3"

These basic elements are what we need to start our own piece of equipment.

Drawing Components with Python

Having setup our development environment, we should make a quick script. The simplest script to create is for Equipment. Create a new file called SimpleVesselSkirt.py in the CustomScripts folder.

To start we are going to define the basic script and then complete part of the drawing routine.

```
from varmain.primitiv import *
from varmain.custom import *
from math import *

@activate(Group="Vessel", TooltipShort="Skirt", TooltipLong="A skirt with a base", LengthUnit="in")
@group("MainDimensions")
@param(D=LENGTH, TooltipShort="Skirt OD")
@param(L=LENGTH, TooltipLong="Length of the Skirt")
@param(D1=LENGTH, TooltipShort="Base OD")
@param(L1=LENGTH, TooltipShort="Base Thickness")
@param(OF=LENGTH, TooltipShort="Skirt Thickness")

def SIMPLEVESSELSKIRT(s, D=48, L=48,D1=50,L1=.25, OF=0,**kw ):
    #create the base shape of the skirt
   thck = OF
    if thck <= 0:
        thck = .05*D
    s = CYLINDER(s, R=D/2,H=L,O=(D-thck)/2)
```

The section at the top includes declarations. These are references to other libraries that we call in our file. For example, the varmain references are for Plant 3D scripts and do the heavy work of drawing shapes in AutoCAD.

The section with @activate defines our properties to Plant 3D. The param section tells Plant 3D what kind of parameters to expect/accept.

The def keyword is the start of our routine. Python has a couple peculiarities. First is that the lines after the def keyword have to be tabbed in. The script above draws a vertical cylinder with a hole through it.

Test and Use scripts with Plant 3D

After getting our initial script going, we need to test it. In the absence of a debugger or compiler feedback, testing the script in Plant 3D is mandatory.

Plant 3D uses a special arx file to test scripts (PnP3DACPAdapter.arx). This adapter must be loaded in order to test custom script routines. In addition, once a script is loaded into memory, Plant 3D won't release that script or refresh it. Therefore, we have to restart Plant 3D to check whether the changes we make to the script break anything. Also, to build the scripts, we have to run a special command in plant that build the scripts. After running that command, we need to restart Plant so the existing scripts can be released from memory.

My personal favorite way to manage the development environment for scripts is to build a tool palette.

Setting up a Testing Environment

The following three tools can be placed on a tool palette for easy access and use.

Use this command macro to load the adapter we need: ^C^C(arxload "PnP3dACPAdapter.arx");

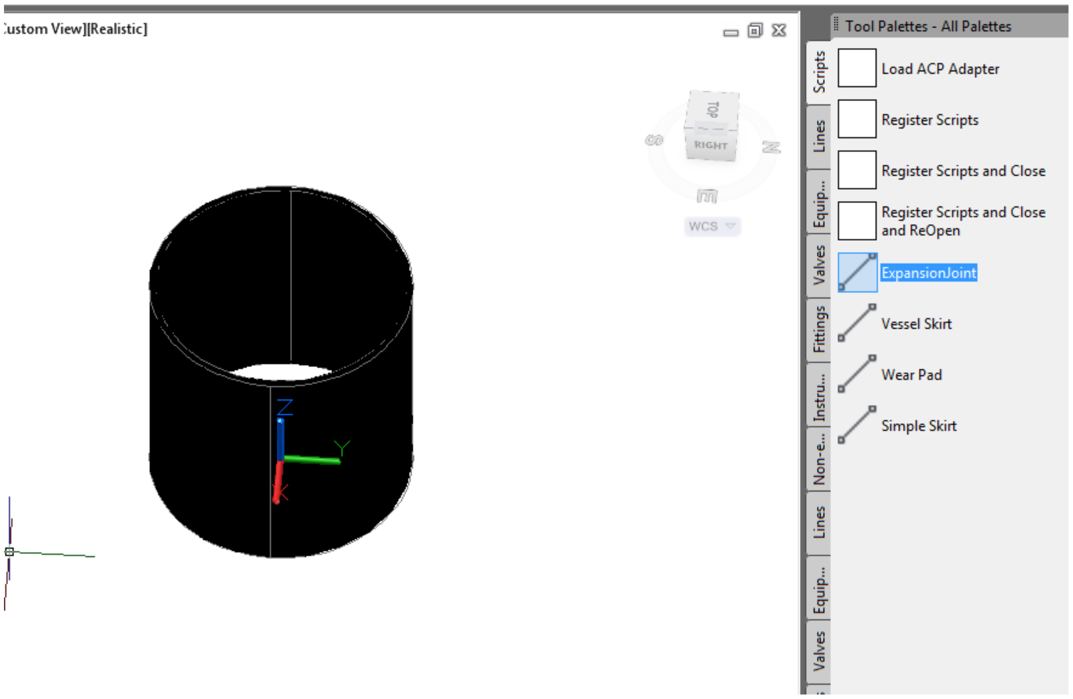
Use this command macro to register the scripts we wrote with Plant: ^C^C(arxload "PnP3DAcpadapter.arx");PLANTREGISTERCUSTOMSCRIPTS;

Use this command macro to register scripts and re-start Plant: ^C^CPLANTREGISTERCUSTOMSCRIPTS;(startapp "C:/Program Files/Autodesk/AutoCAD Plant 3D 2013 - English/acad.exe");QUIT

For each script you need to test, create another tool palette button and use this macro: ^C^C(TESTACPSCRIPT "EXPJOINT1_F_F")

The part in parentheses "EXPJOINT1_F_F" should be replaced with your script name.

With our development environment setup, you should be able to register the scripts and test our SimpleVesslSkirt.py.

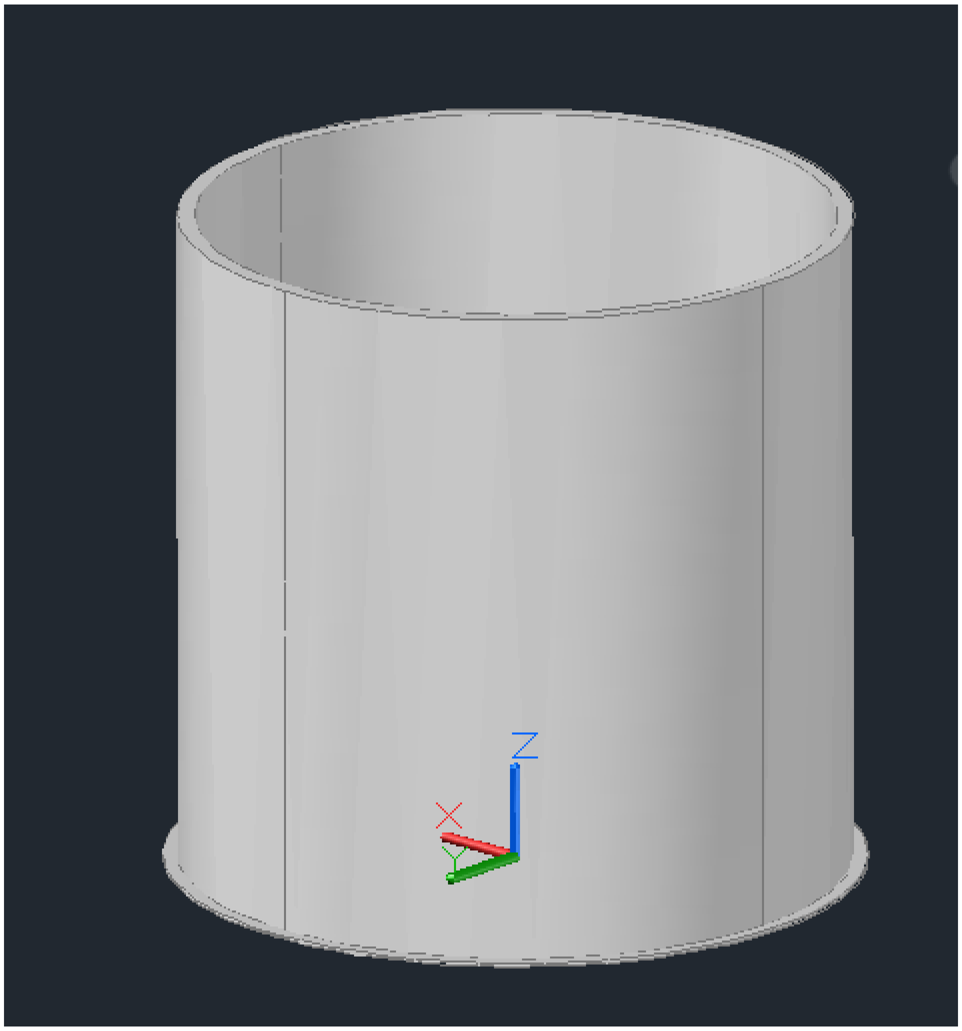


Modify the Script

After verifying that our script is formatted well initially and we get a successful shape, we need to finish out the skirt.

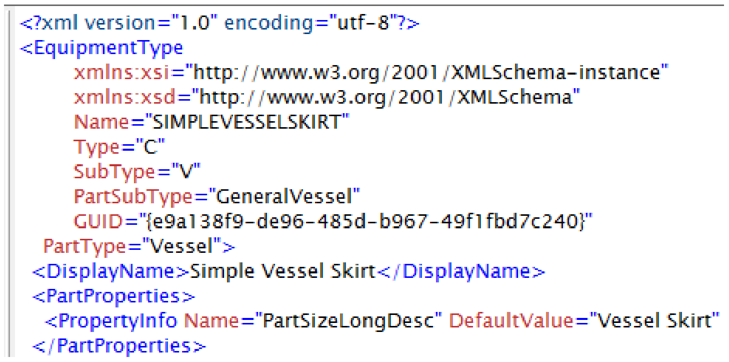
Adding these lines finishes out our simple script:

```
#create the base of the skirt
  b = CYLINDER(s, R=D1/2,H=L1)
   #union the two pieces together
  skirt.uniteWith(b)
   #release the secondary object
  b.erase()
```



Create the Equipment Package

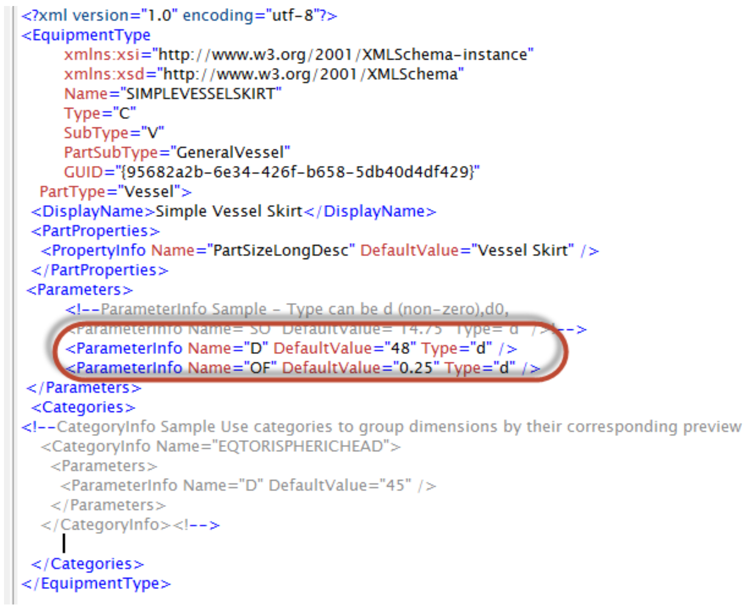
With our script complete, we need to create the package. We will copy and modify the equipment template zip. Change the EquipmentType.xml so it has a guid, display name, and the correct script name. Make sure you are not in the peqx file while Plant 3D is trying to load, it will not be able to read the peqx.



```
<?xml version="1.0" encoding="utf-8"?>
<EquipmentType
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       Name="SIMPLEVESSELSKIRT"
       Type="C"
       SubType="V"
       PartSubType="GeneralVessel"
       GUID="{e9a138f9-de96-485d-b967-49f1fbd7c240}"
   PartType="Vessel">
  <DisplayName>Simple Vessel Skirt</DisplayName>
  <PartProperties>
    <PropertyInfo Name="PartSizeLongDesc" DefaultValue="Vessel Skirt"
  </PartProperties>
```

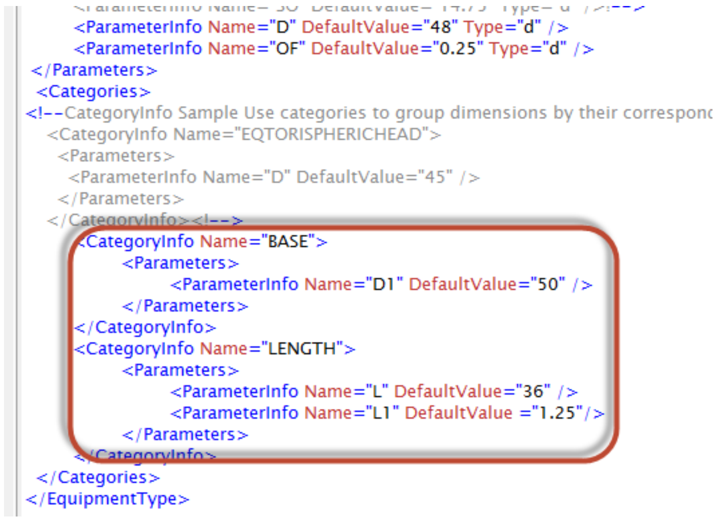
Add Parameter Information

In order for Plant to know what parameters the script needs, we define them in the EquipmentType.xml. To the equipment type xml, add the following parameter information for parameters, D, L, D1, L1, and OF.



```xml
<?xml version="1.0" encoding="utf-8"?>
<EquipmentType
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        Name="SIMPLEVESSELSKIRT"
        Type="C"
        SubType="V"
        PartSubType="GeneralVessel"
        GUID="{95682a2b-6e34-426f-b658-5db40d4df429}"
   PartType="Vessel">
  <DisplayName>Simple Vessel Skirt</DisplayName>
  <PartProperties>
    <PropertyInfo Name="PartSizeLongDesc" DefaultValue="Vessel Skirt" />
  </PartProperties>
 <Parameters>
        <!--ParameterInfo Sample - Type can be d (non-zero),d0,
        <ParameterInfo Name="SO" DefaultValue="14.75" Type="d" />-->
        <ParameterInfo Name="D" DefaultValue="48" Type="d" />
        <ParameterInfo Name="OF" DefaultValue="0.25" Type="d" />
 </Parameters>
  <Categories>
<!--CategoryInfo Sample Use categories to group dimensions by their corresponding preview
    <CategoryInfo Name="EQTORISPHERICHEAD">
      <Parameters>
        <ParameterInfo Name="D" DefaultValue="45" />
      </Parameters>
    </CategoryInfo><!-->
        |
  </Categories>
</EquipmentType>
```

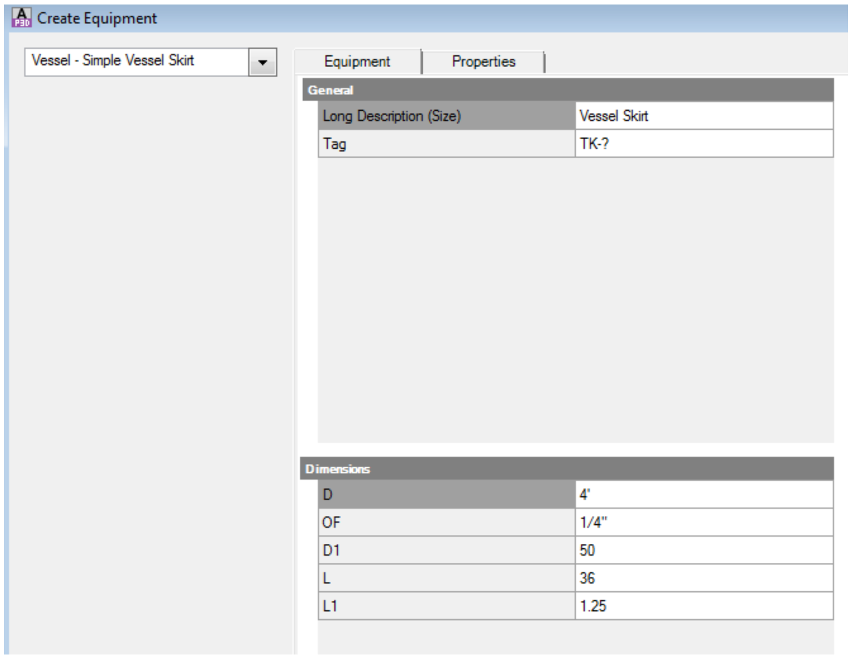
The parameter names match the parameters being used in our script. We can divide our parameters into categories according to how we will show their dimensioned preview. For example, the main image will be the dimensions for the OD of the skirt and the thickness of the shell. We will create a new category info group to give dimensions for the height of the skirt, thickness of the base and od of the base.



```xml
        <ParameterInfo Name="D" DefaultValue="48" Type="d" />
        <ParameterInfo Name="OF" DefaultValue="0.25" Type="d" />
 </Parameters>
  <Categories>
<!--CategoryInfo Sample Use categories to group dimensions by their correspond
    <CategoryInfo Name="EQTORISPHERICHEAD">
      <Parameters>
        <ParameterInfo Name="D" DefaultValue="45" />
      </Parameters>
    </CategoryInfo><!-->
        <CategoryInfo Name="BASE">
                <Parameters>
                        <ParameterInfo Name="D1" DefaultValue="50" />
                </Parameters>
        </CategoryInfo>
        <CategoryInfo Name="LENGTH">
                <Parameters>
                        <ParameterInfo Name="L" DefaultValue="36" />
                        <ParameterInfo Name="L1" DefaultValue ="1.25"/>
                </Parameters>
        </CategoryInfo>
  </Categories>
</EquipmentType>
```

We end up needing to create three images for the general vessel image which will have the D and OF parameters dimension, one for the BASE with D1 dimension, and one for LENGTH with L and L1 dimensioned.

If the xml is formatted correctly, and we included our parameters, we should be able to use the Create equipment dialog:

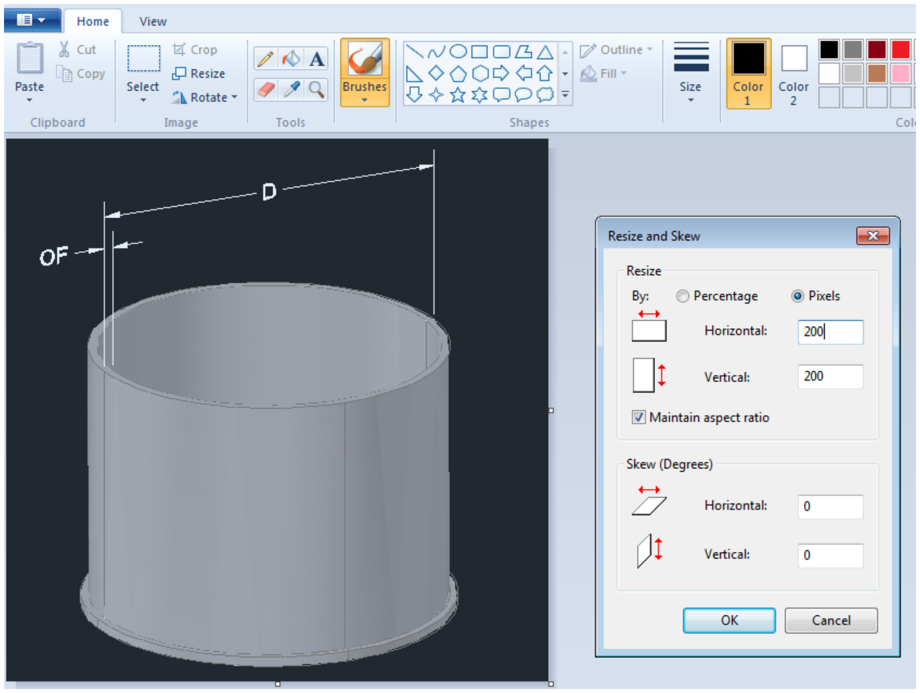


General	
Long Description (Size)	Vessel Skirt
Tag	TK-?

Dimensions	
D	4'
OF	1/4"
D1	50
L	36
L1	1.25

Create Help Images

Now we can set up our first screenshot. I use Jing (http://jingproject.com) to take screenshots, and we can resize them down using Paint.



Resize and Skew

Resize

By: Percentage [✓ Pixels]

Horizontal: 200

Vertical: 200

[✓] Maintain aspect ratio

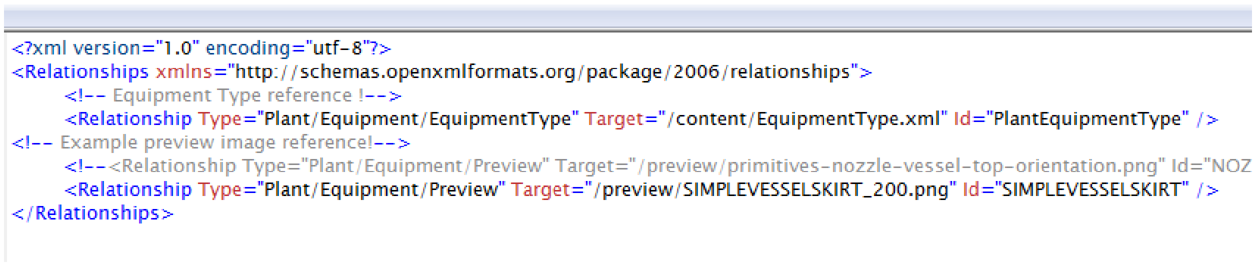
Skew (Degrees)

Horizontal: 0

Vertical: 0

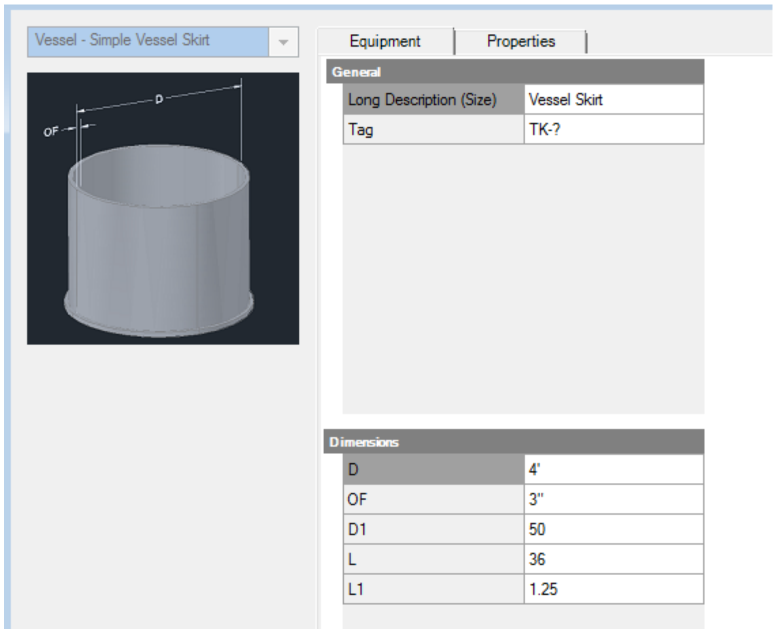
OK Cancel

After adding that to our peqx, we need to add a reference to it in our .rels. Because we don't have the dimensions this image references in a category, we simply use the script name as the id.



```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
        <!-- Equipment Type reference !-->
        <Relationship Type="Plant/Equipment/EquipmentType" Target="/content/EquipmentType.xml" Id="PlantEquipmentType" />
<!-- Example preview image reference!-->
        <!--<Relationship Type="Plant/Equipment/Preview" Target="/preview/primitives-nozzle-vessel-top-orientation.png" Id="NOZ
        <Relationship Type="Plant/Equipment/Preview" Target="/preview/SIMPLEVESSELSKIRT_200.png" Id="SIMPLEVESSELSKIRT" />
</Relationships>
```

Assuming our xml is formatted, and we are not in the peqx, opening plant and modifying our current skirt or placing a new one should bring us to this:



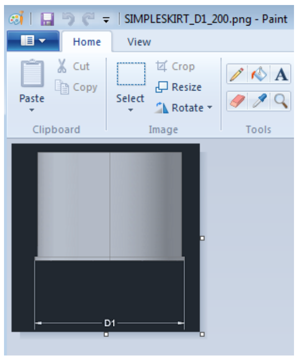
Vessel - Simple Vessel Skirt

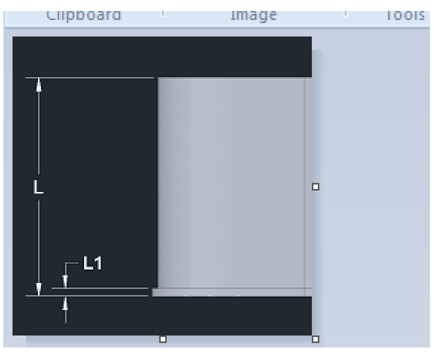
General	
Long Description (Size)	Vessel Skirt
Tag	TK-?

Dimensions	
D	4'
OF	3"
D1	50
L	36
L1	1.25

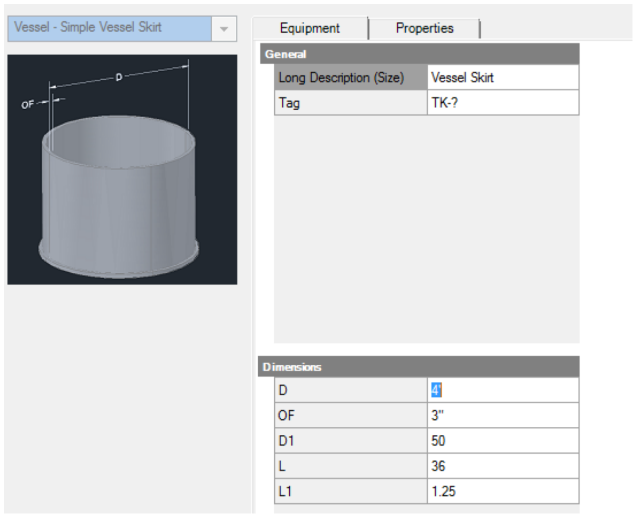
We can create the other images the same way, and relate them to their category ids.

We need an image for the BASE (D1).





After completing our .rels entries and packaging our peqx, we should get images for all of our dimensions.

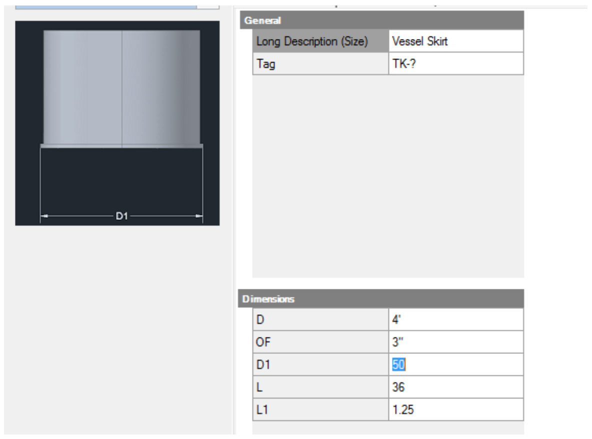


General

Long Description (Size)	Vessel Skirt
Tag	TK-?

Dimensions

D	4'
OF	3"
D1	50
L	36
L1	1.25

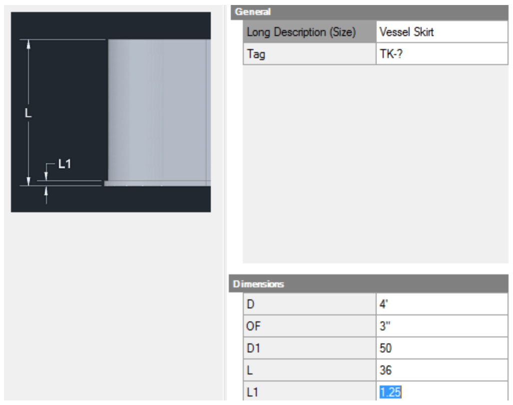


General

Long Description (Size)	Vessel Skirt
Tag	TK-?

Dimensions

D	4'
OF	3"
D1	50
L	36
L1	1.25



General

Long Description (Size)	Vessel Skirt
Tag	TK-?

Dimensions

D	4'
OF	3"
D1	50
L	36
L1	1.25

Deploying Content

The final step is to deploy our equipment. Depending on your scenario, you may be able to deploy the SIMPLEVESSELSKIRT.pyc, SIMPLEVESSELSKIRT.xml, variants.map, variants.xml, and ScriptGroup.xml if you know your users are not using other custom scripts. Otherwise, you’ll need to deploy your .py file and register the script. The equipment peqx we created can go in the CPak Common folder.