BES219848

Revit Programming for beginners: How to easily access the power of the Revit API using free easy to use tools.

Dan Mapes
ME Engineers

Learning Objectives

- Use free tools to access the Revit API.
- Collect and filter elements with Revit Python Shell.
- Perform a transaction on elements with Revit Python Shell.
- Save and use Python tools.

Description

Ever want to learn to use the Revit API? Programming in Revit seems like a scary thing, but it doesn't have to be. I will present on how to get started using the Revit API using free tools. This class will guide attendees who use Revit but never have tried to access the API to be able to accomplish simple tasks. Using element filters, Transactions, and User Interface will be the focus. This will give each attendee a basic understanding and roadmap to creating their own scripts for Revit. I will use the following software: Revit, Revit Python Shell, Revit Lookup and pyRevit. Attendees will need basic understanding of Revit and entry level understanding of Python.

Speaker

Daniel Mapes is a BIM Coordinator/Developer for ME Engineers a multi discipline engineering firm based in Denver Colorado. ME Engineers specializes in large professional and collegiate stadium projects. Daniel Mapes has been Lead BIM coordinator on high profile projects such as T-Mobile Arena in Las Vegas, NV, Mississippi State Baseball stadium, Twins stadium renovations in Minnesota and many more. He provides BIM implementation and training for the firm's production software including Revit, Navisworks, and ACAD for the Kansas City office and provides support for other branch offices around the country. He has 14 years in the building engineering field using Revit for 9 years. Dan currently develops mechanical families for the firm's standard library and python-based tools used by multiple offices.

Why Python

Python is a general-purpose open source programming language. It is used for a variety of purposes including machine learning, mathematics, data science and web site development. Since python is an interpreted language it is not required that it be compiled. Instead an interpreter is needed to execute the code. The Iron Python interpreter is written in C# and allows python to run on windows systems.

Of all the programming languages Python is popular because it is easy to learn. The syntax used makes it easier to read than most other languages. Python also comes standard with a comprehensive library of modules that provide a lot of additional functionality. This makes Python very accessible to those who are not trained in programming but still want to harness the power of the Revit API to create time saving tools.

The advantage if being an interpreted language instead of a compiled language is ease of implementing updates to tools without needing to reinstall.

Getting Started

There are a few thigs that need to be installed to be able to create and update your own tools with pyRevit.

1. Install Python version 2.7. PyRevit runs on this version of Python. There are later versions of Python available so be sure to choose version 2.7 prior to installing.

2. Install a text editor. I prefer to use Visual Studio Code but there are many options to choose from.

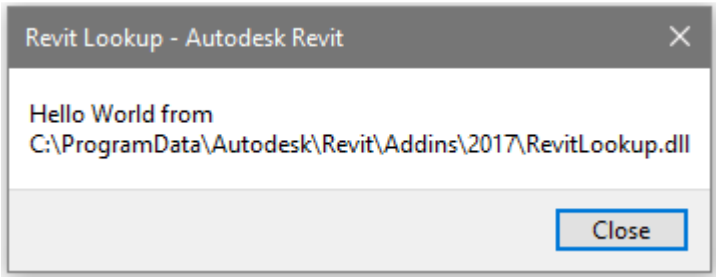
3. Install the free add-ins listed below and configure as described.

Free tools to access the Revit API

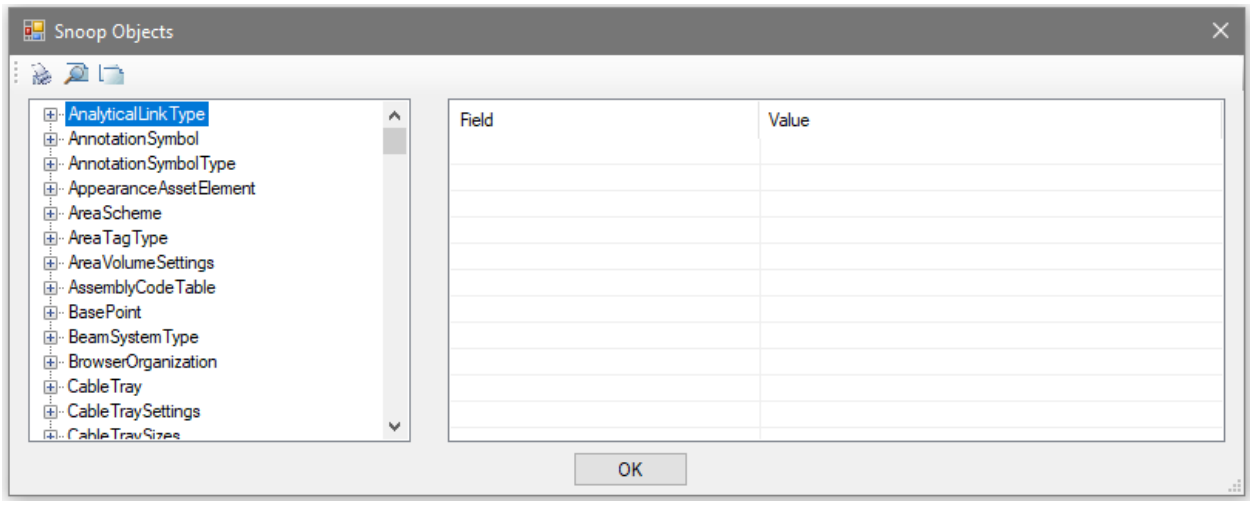
Revit Lookup

Interactive Revit BIM database exploration tool to view and navigate element properties and relationships. Revit Lookup is an add-in that lets the user explore the Revit API on an element or project basis.

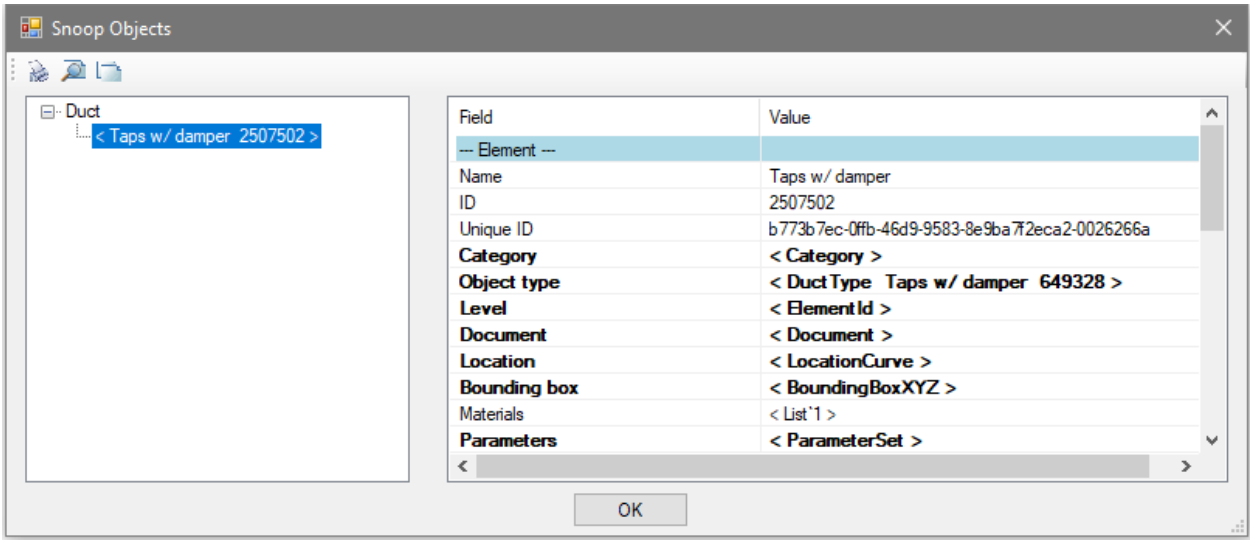
Hello World – will list the location of the add-in .dll file.



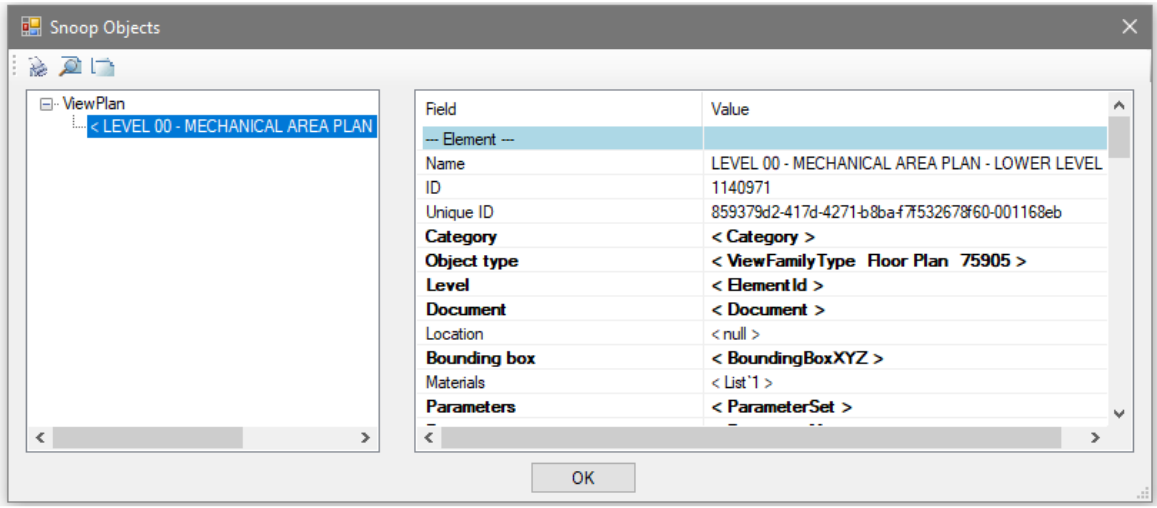
Snoop DB – will show the entire database for the current model.



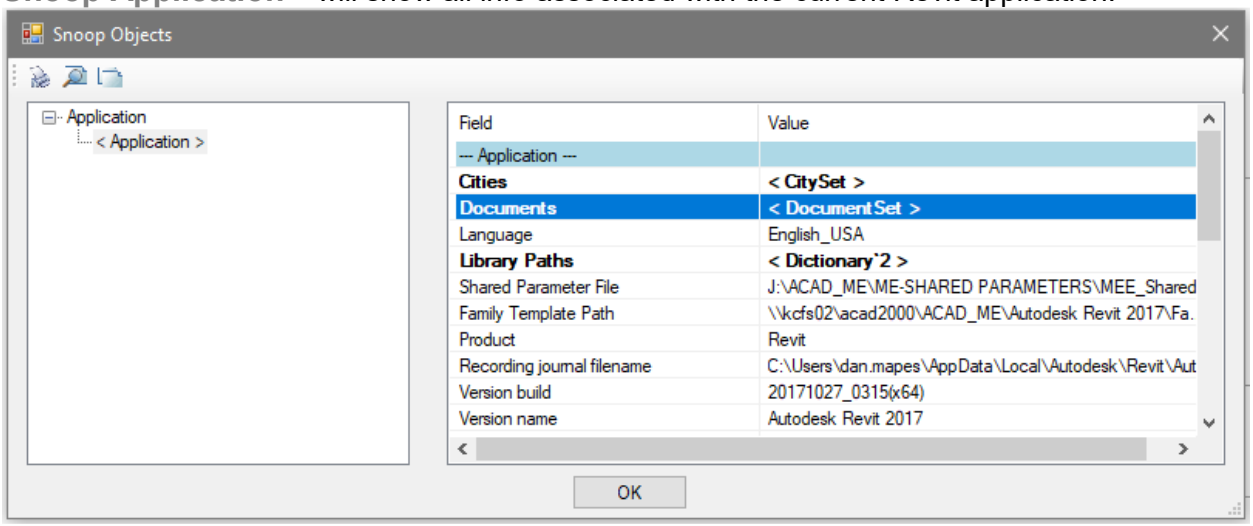
Snoop Current Selection – will show all info associated with the element that is currently selected.



Snoop Current View – will show all info associated with the view that is current .



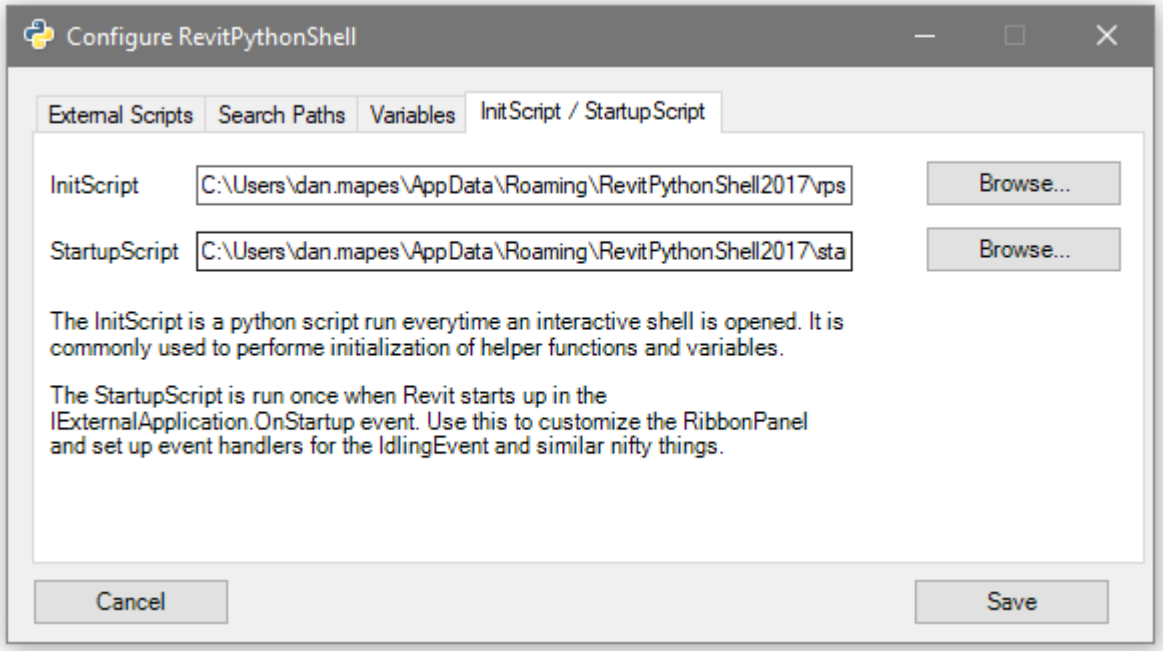
Snoop Application – will show all info associated with the current Revit application.



Revit Python Shell

Revit Python Shell is an Iron Python console that runs inside of Revit. It is used to write and run scripts directly in the Revit environment. Since it is interactive it allows the user a means to explore the Revit API as you are writing.

Configure – will allow the user to customize Revit Python Shell.



As you can see this has many options. We will just focus on the InitScript. This script will be executed each time you run Revit Python Shell. It is useful for including typical imports and basic functions that you will not want to have to type each time.

Ehsan Irannejad the creator of pyRevit https://github.com/eirannejad has provided an example of a customized script for this purpose. Simply path your InitScript location to this file and customize as you would like. (full script is on the next page)

1. The first 6 lines import all available classes from the Revit API
2. Then clr is loaded this is the Common Runtime language
3. Then the Revit Db is loaded as DB for ease in writing later.
4. Doc, uidoc and selection are called.
5. Alert message Quit are defined.
6. Information about the current selection is accessed using ”el”.

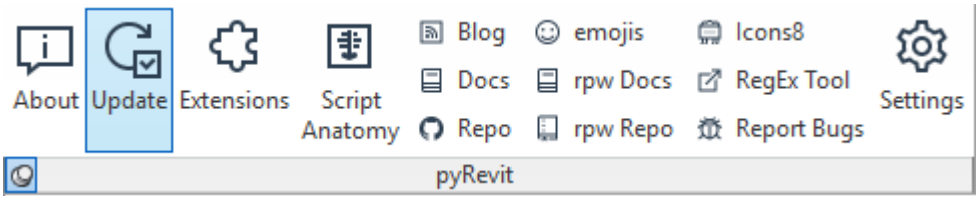
Here is the entire code for your use. Save as rps_init.py and use with your Revit Python Shell.

```
# these commands get executed in the current scope
# of each new shell (but not for canned commands)
from Autodesk.Revit.DB import *
from Autodesk.Revit.DB.Architecture import *
from Autodesk.Revit.DB.Analysis import *
from Autodesk.Revit.UI import *
import clr

import Autodesk.Revit.DB as DB

clr.AddReferenceByPartialName('PresentationCore')
clr.AddReferenceByPartialName("PresentationFramework")
clr.AddReferenceByPartialName('System.Windows.Forms')
import System.Windows

uidoc = __revit__.ActiveUIDocument
doc = __revit__.ActiveUIDocument.Document
selection = [doc.GetElement(elId) for elId in
__revit__.ActiveUIDocument.Selection.GetElementIds()]


def alert(msg):
    TaskDialog.Show('pyRevit', msg)

def quit():
    __window__.Close()

# element
if len(selection) > 0:
    el = selection[0]
```

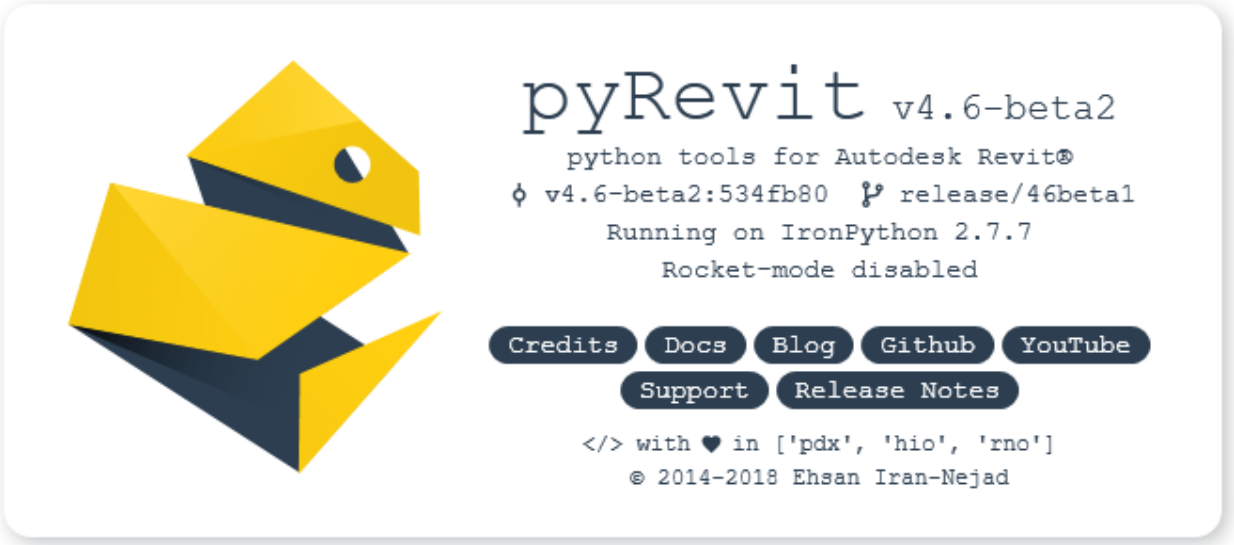
pyRevit

pyRevit is an Iron python script library and prototyping environment for Revit. The pyRevit add-in allows the user to create python scripts and add them directly to a Ribbon in Revit. pyRevit will run each script file when the button is activated. When changes are made to script tools it will run the updated version without reinstalling or reloading.

Core:



About:

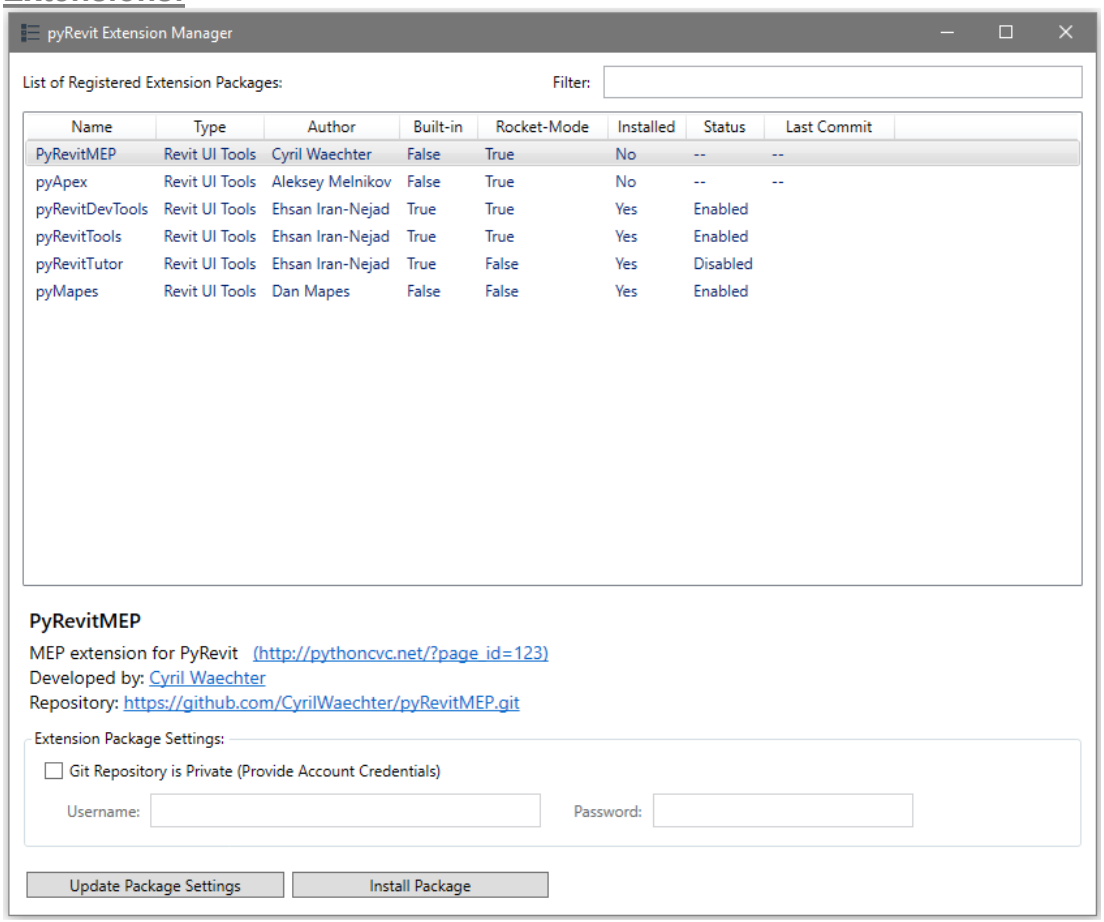


Update:

Runs a script that updates the installed version of pyRevit by looking at the GIT repository and downloading changes. This button will be changing in newer versions as the deployment method will be changing to a CLI (Command Line Interface) based update process for more control and better consistency.

See https://github.com/eirannejad/pyRevitLabs/blob/master/README_CLI.md for more information.

Extensions:

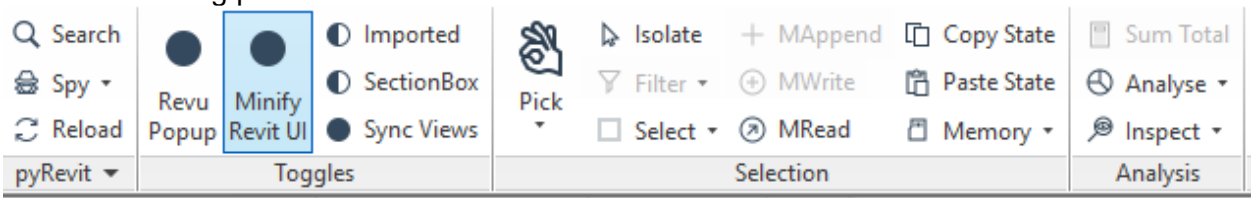


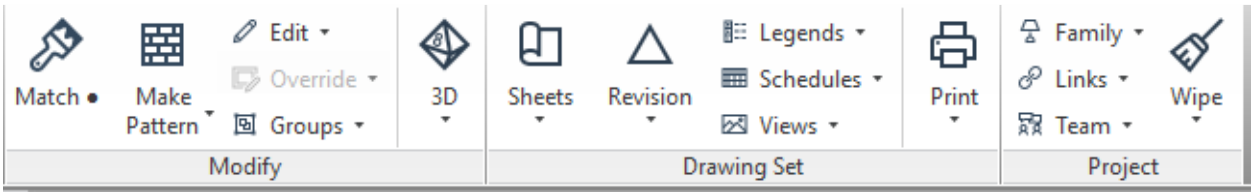
Settings:

Interface to update the settings for your configuration of pyRevit. More on this later when we add our own extensions.

Tools:

pyRevit's tools include a wide arrangement of great tools that showcase what is possible with this amazing platform.





Creating an Element filter using Python Shell

ElementFilter Class
A base class for a type of filter that accepts or rejects elements based upon criteria.

1. Create an instance of the filtered element collector by calling “DB.FilteredElementCollector” and giving it a name of “mech_equip”.

2. Choose the method of the filtered element collector to use. There are many to choose from but in this example “OfCategory” is used. See the API documentation for the other methods available.

3. In step 3 we add another method “WhereElementIsNotElementType” to further narrow the collector by not including elements that are an element type.

4. Finally, we get the collected elements in the form of element Id’s by calling the method “ToElementIds”.

```
mech_equip = DB.FilteredElementCollector(doc) \
          .OfCategory(DB.BuiltInCategory.OST_MechanicalEquipment) \
          .WhereElementIsNotElementType() \
          .ToElementIds()
```

Doing a simple Transaction using Python Shell

Transaction Class

Transactions are context-like objects that guard any changes made to a Revit model.

1. Create an instance of Transaction by calling “Transaction” and giving it a name of “t”.

2. Next call “t” and use the Start method to begin the transaction. No changes will be made to the database unless they are within a transaction.

3. Step 3 performs an action on some elements. In this case we are looking for a parameter “Mech Type Mark”.

4. Finally, we must finish the transaction by using the Commit method. This will let Revit know that we have finished the transaction and any changes will be committed to the database. If there are any incomplete parameters or if the methods within the transaction were done incorrectly Revit will roll back the changes and give an error.

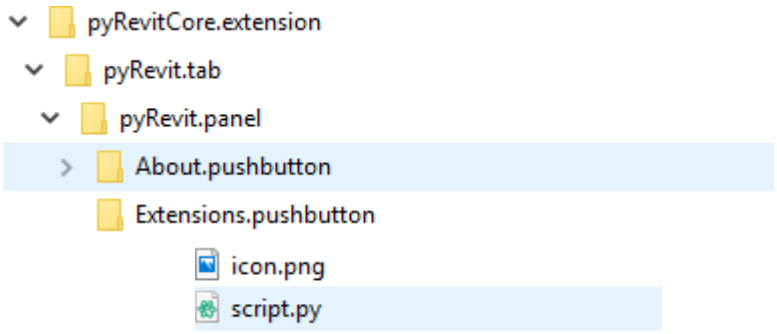
```
#transaction to update parameter
t = Transaction(doc, "Update Mech Type Mark")
t.Start()

for eq in mech_equip:
    eq_element = doc.GetElement(eq)
    mark_param = eq_element.LookupParameter('Mech Type Mark')
    mark = eq_element.LookupParameter('Mark').AsString()
    family_symbol_name = eq_element.Symbol.LookupParameter('Type
Name').AsString()
    family_ids = eq_element.Symbol.Family.GetFamilySymbolIds()
    for id in family_ids:
        fam_sym = doc.GetElement(id)
        fam_name = fam_sym.LookupParameter('Family Name').AsString()
        type_mark = fam_sym.LookupParameter('Type Mark').AsString()
        type_symbol_name = fam_sym.LookupParameter('Type
Name').AsString()
        ext_mark = '{0}'.format(type_mark)
        if family_symbol_name == type_symbol_name:
            mark_param.Set(ext_mark)

t.Commit()
```

Getting scripts implemented with pyRevit

pyRevit loads each tab, Panel, and pushbutton into Revit. Then pyRevit will run each script file when the button is clicked. If script files are updated pyRevit will run the updated version the next time the button is clicked

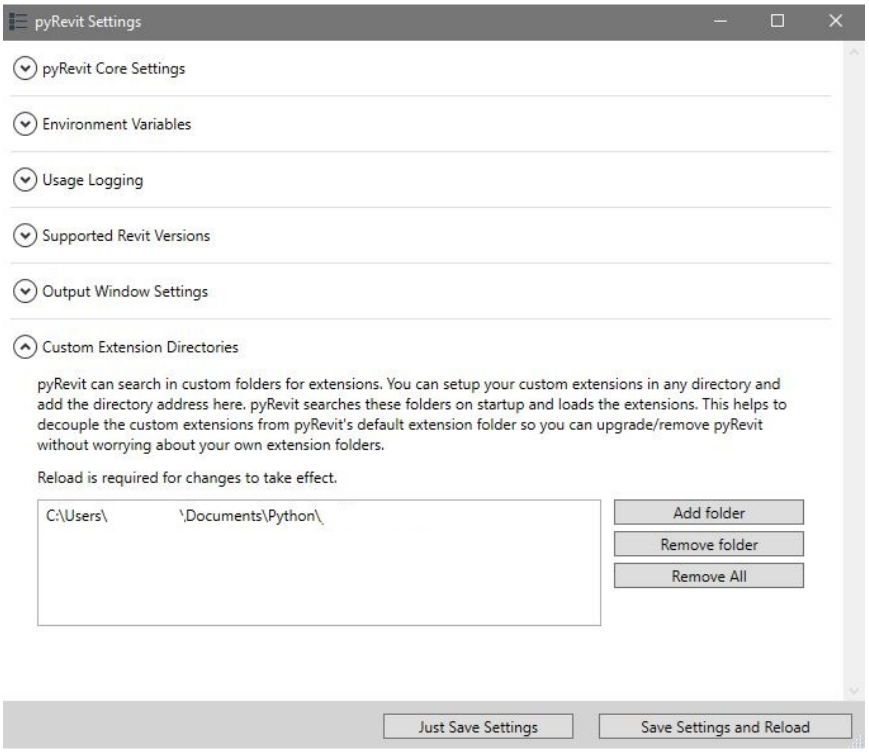
Step 1 save your script.

Save you script as script.py in a folder using the folder structure shown.



Step 2 Add path to pyRevit.

Add your custom path to the pyRevit application using the Settings tool.



Resources:

Revit Lookup:
https://github.com/jeremytammik/RevitLookup
https://boostyourbim.wordpress.com/2016/05/09/revit-lookup-2017-installer/

Revit Python Shell:
https://github.com/architecture-building-systems/revitpythonshell

pyRevit:
http://eirannejad.github.io/pyRevit/whatspyRevit/

Visual Studio Code text editor:
https://code.visualstudio.com/

API Documentation:
https://apidocs.co/