Dynamo: Parts custom nodes

Packages

Julien_BENOIT 2014-01-22 15:53:14 UTC #1

Hi guys,

I think that I will have to dive into some code sooner or later. As a start, I try to implement some python script to deal with Parts in Dynamo.

I have opened many custom nodes with Python inside, but it seems that the Parts "thing" is a bit tricky.

Does any python guru out there would be kind enough to show me a starting point? I get the Parts

Util method from API search, but translating this to python....can't do.

Thanks, Julien

Julien_BENOIT 2014-01-24 02:23:40 UTC #2

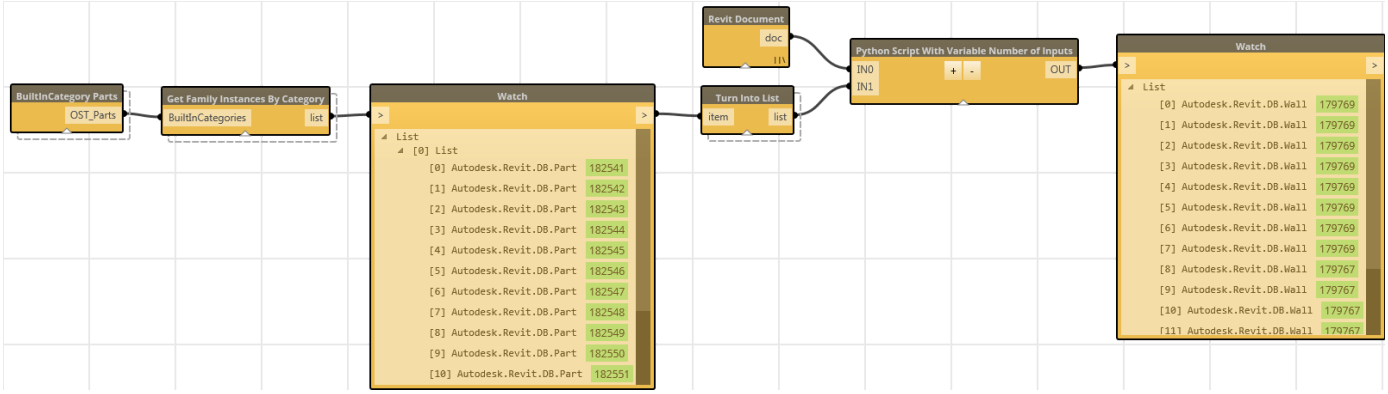
Andreas, I owe you a serious one! many thanks for the code AND the tutorial. A milestone in my "IwannaBeACoder" career! let me know if ever you plan a travel here, we could do something....;-)

I had a quick look at Nathan's notebook, awesome content for sure.

thanks again for your time on this one.

Andreas_Dieckmann 2014-01-24 01:11:44 UTC #3

Okay, here's how you would do it:




```
import clr
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *
import Autodesk

doc = IN0
items = IN1
elementlist = list()
for item in items:
sourcelist = list()
for source in item.GetSourceElementIds():
sourcelist.append(doc.GetElement(source.HostElementId))
if len(sourcelist) < 2:
elementlist.append(sourcelist[0])
else:
elementlist.append(sourcelist)
OUT = elementlist
```

I am by no means a skilled coder, but I'll make this into a tutorial as best as I can. So let's walk through this step by step.

```
import clr
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *
import Autodesk
```

Here we import all the references necessary to interact with the Revit API.

```
doc = IN0
items = IN1
elementlist = list()
```

Next, we need to declare some variables. We could just keep using the automatically declared variables *IN0* and *IN1* from our input ports, but I prefer to keep my code readable, so I usually choose a variable name that describes what my input is exactly. (You could actually declare *doc* directly in Python without using a node as an input, but there was a time when this didn't work, so I got used to using the *Revit Document* node instead...) The *elementlist* is an empty list that will later be filled with stuff.

```
for item in items:
```

Iterate through every item in the input list. Notice how I'm using a custom node called *Turn Into List* between the list of parts and the Python node. I can't know whether the user wants to put in a list or just a single object. Also, my code is usually not fit to go through a list of lists. *Turn Into List* will convert a single value into a list of 1 item or a list of lists into a flat list.

```
sourcelist = list()
```

Declare another empty list. In theory, parts can have more than one parent object, so the *GetSourceElementIds* method will always return an *ICollection* (basically a glorified list). The sourcelist is reset to an empty list in every iteration (i.e. when looking at the next part), so it can work as a sublist of our elementlist.

```
for source in item.GetSourceElementIds():
```

Iterate through the list of elements that the *GetSorceElementIds* method will return for a given part.

BTW: Your code returned a list of items called *IronPython.Runtime.Types.BuiltInFunction*. That's because you called a method like *GetSourceElementIds* without adding brackets behind it. The brackets usually contain arguments for the method, but in the case of *GetSourceElementIds* no arguments are needed. If you look at Revit's API documentation, the members section is always divided into at least two subsections: methods and properties. This is how you would call them in python:

- object.method(args)
- object.property

```
sourcelist.append(doc.GetElement(source.HostElementId))
```

This next line looks a bit more complicated because there are several expressions nested into one another. First, *source.HostElementId* will get us the ID of the source object of the given part. Next, *doc.GetElement()* will get us a Revit Object for a given ID (the one we just found through *source.HostElementId*). Lastly, we can use *sourcelist.append()* to add that element to our list of source objects for the given part. If I weren't so lazy I could also write this in several lines to make it more legible:

```
elementID = source.HostElementId
```

```
element = doc.GetElement(elementID)

sourcelist.append(element)
```

And the last part:

```
if len(sourcelist) < 2:
elementlist.append(sourcelist[0])
else:
elementlist.append(sourcelist)
```

In most cases, each part will only have a single source element, so what I'm doing here is not returning a sublist of source elements if it isn't necessary (probably 98% of all cases...). So basically, *sourcelist[0]* gets me the first item of *sourcelist*.

I hope this helps a bit. Usually, for a node like this, I would try to make it safer. If you feed elements that aren't parts into this node it will throw an exception, because the *GetSourceElementIds* method is specifically designed for parts. If you look at some of my custom nodes you will find a *try/except* structure that can keep the node from turning red.

(EDIT: Just realized that the try/except bit is in your code already...) :-)

I would recommend you have a look at Nathan Miller's Revit API Notebook - that is what got me started coding Python for the Revit API (before Dynamo I used the RevitPythonShell). Also, to understand the basics of Python, I usually recommend **this site** as it covers most of the basics quite concisely.

Lastly, professional coders might cringe at some of what I've written here, but so far it's worked for me... 😉

Julien_BENOIT 2014-01-23 17:20:09 UTC #4

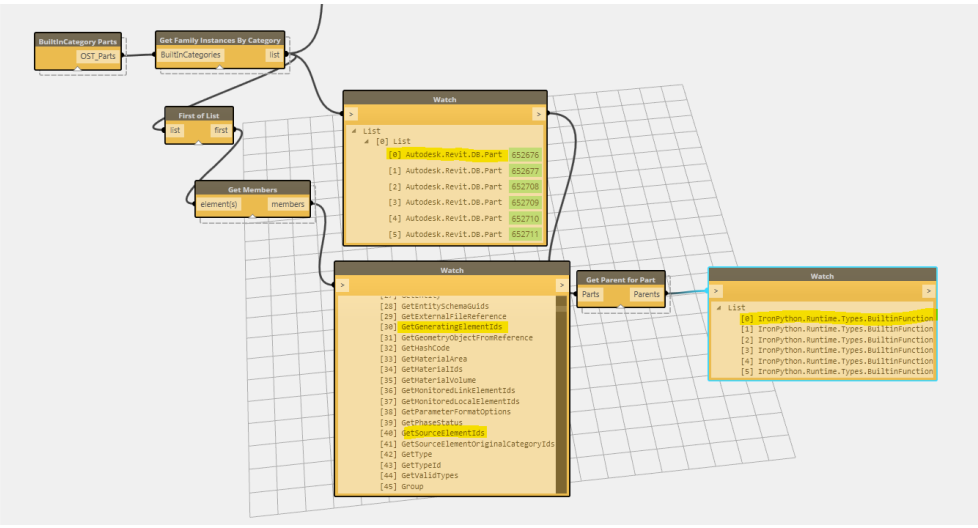
Thanks for the help Lev, answer is surely hidden somewhere :). will keep that for later.

first try is like this (coming from Andreas's Get Host "template"):

```
import clr
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *
import Autodesk

faminsts = IN
elementlist = list()
unmatched = list()
for item in faminsts:
try:
elementlist.append(item.GetGeneratingElementIds)
except:
elementlist.append('')
OUT = elementlist
```

I tried also with GetSourceElementIds, it returns this



humble I have to be.....

Lev_Lipkin 2014-01-23 14:47:34 UTC #5

I wonder if this link: **http://thebuildingcoder.typepad.com/blog/parts/** could be helpful.

Andreas_Dieckmann 2014-01-23 12:45:18 UTC #6

Not sure if that's possible.

You should be able to get the Revit element(s) a part was made from. Parts have a method called GetSourceElementIds. That should get you a collection of IDs(or probably just one ID per part). An ID can be converted back into an element (see my package Element From & To IDs).

Weird that autocomplete doesn't work on your machine.

Julien_BENOIT 2014-01-23 10:25:16 UTC #7

I think I have to learn a bit more. I started from your node Get host, my primary goal is to get the Parts that comes from an element, as a wall is kind of host for the parts.Parts are hosted by the parent.

About the autocomplete in the python node, nothing happens here: I may have to install something like the SDK.

thanks for asking!!

Andreas_Dieckmann 2014-01-23 04:43:54 UTC #8

What is it you want to do exactly?

PartUtils should be fairly accessible: If you type "PartUtils." into the Python Script node, autocomplete should kick in.

Powered by **Discourse**, best viewed with JavaScript enabled