Follow @jeremytammik
About – Topics – Index – Source

Search | Submit

Subscribe to this blog's feed

July 27, 2023

Export, gbXML and Python Tips

Looking at several useful discussions on Python, handling DLLs, and various aspects of exporting to gbXML, FBX and MongoDB today:

- DLL paradise in Python
- Multiple gbXML export
- GbXML energy settings
- Automate FBX export with `SendKeys`
- RFA export to MongoDB

DLL Paradise in Python

Jake of Ripcord Engineering shared several useful Revit API discussion forum solutions recently. Many thanks to Jake for his support!

One is a possible approach to handle DLL Hell using the Python `subprocess` module for disentanglement without need for any IPC, e.g., for CPython and pyRevit:

Question: I need to use CPython via pyRevit to have access to libraries such as `numpy` and `pandas`. At the same time, I want to take advantage of pyRevit's capabilities such as forms etc. As far as I understood, I can't have both of these in a single script file. If I got this correctly, is there any way to do this?

The numpy part is quite decoupled since it is meant to help me with the data exchange process from other data sources; after that point, everything would be focused on Revit APIs.

Answer: I dealt with the same challenge a little while back. Please look at the pyRevit issue 1731 on Dynamo incompatibility: two versions of Same DLL for a short discussion on using Python `subprocess` module for subprocess management in the Revit/pyRevit context.

While I am not a Revit API / Python / pyRevit expert I can report that `subprocess` worked well enough. Learning `subprocess` should be a productive use of time assuming the underlying characteristics are a good match for your application.

Response: Thanks, Jake. I tried the same approach, and it also worked perfectly for my case. Appreciate it.

Answer: Thanks for giving it a go. And thanks for the feedback.

Multiple GbXML Export

Jake also helped answer the question on export of multiple gbXML models:

Question: For my university thesis work I have to create a lot of different GBXML models (around 18000). No way I can do that without code. This is what I came up with (I attached only a part of it; FloorR, WallsR, RoofR are lists to set R value of corresponding elements):

```
### Setting Energy Analysis parameters ###

opt=Analysis.EnergyAnalysisDetailModelOptions()
opt.EnergyModelType=Analysis.EnergyModelType.BuildingElement
opt.ExportMullions=False
opt.IncludeShadingSurfaces=False
opt.SimplifyCurtainSystems=True
opt.Tier=Analysis.EnergyAnalysisDetailModelTier.SecondLevelBoundaries

### loop over all R-value combinations and create models ###

t=Transaction(doc,"R change")
c=Transaction(doc,"model creation")

for i in range(len(FloorR)):
  for j in range(len(WallsR)):
    for k in range(len(RoofR)):
    t.Start()
    Floor.Set(FloorR[i]/0.3048)  #R-value change for floor
    Wall.Set(WallsR[j]/0.3048)#R-value change for Walls
    Roof.Set(RoofR[k]/0.3048)#R-value change for roof
    t.Commit()
    t.Dispose()

    c.Start()
    model=Analysis.EnergyAnalysisDetailModel.Create(doc, opt)
    model.TransformModel()
    GBopt=GBXMLExportOptions()
    GBopt.ExportEnergyModelType=ExportEnergyModelType.BuildingElement
    doc.Export("C:\Users\Миша\Desktop\ASD","0"+","+str(0.2/FloorR[i])+","+str(0.3/WallsR[j])+","+
str(0.3/RoofR[k]), GBopt)
    c.Commit()
```

This creates models, but I ran into a problem I don't fully understand: as the process continues, it slows down and stops at about 170-175 created models. Apparently, something is taking up the memory. I tried calling `doc.Delete(model)` at the end of each `for` loop, but that didn't help either.

What could be a solution?

Answer: The behaviour you describe is completely expected and as designed.

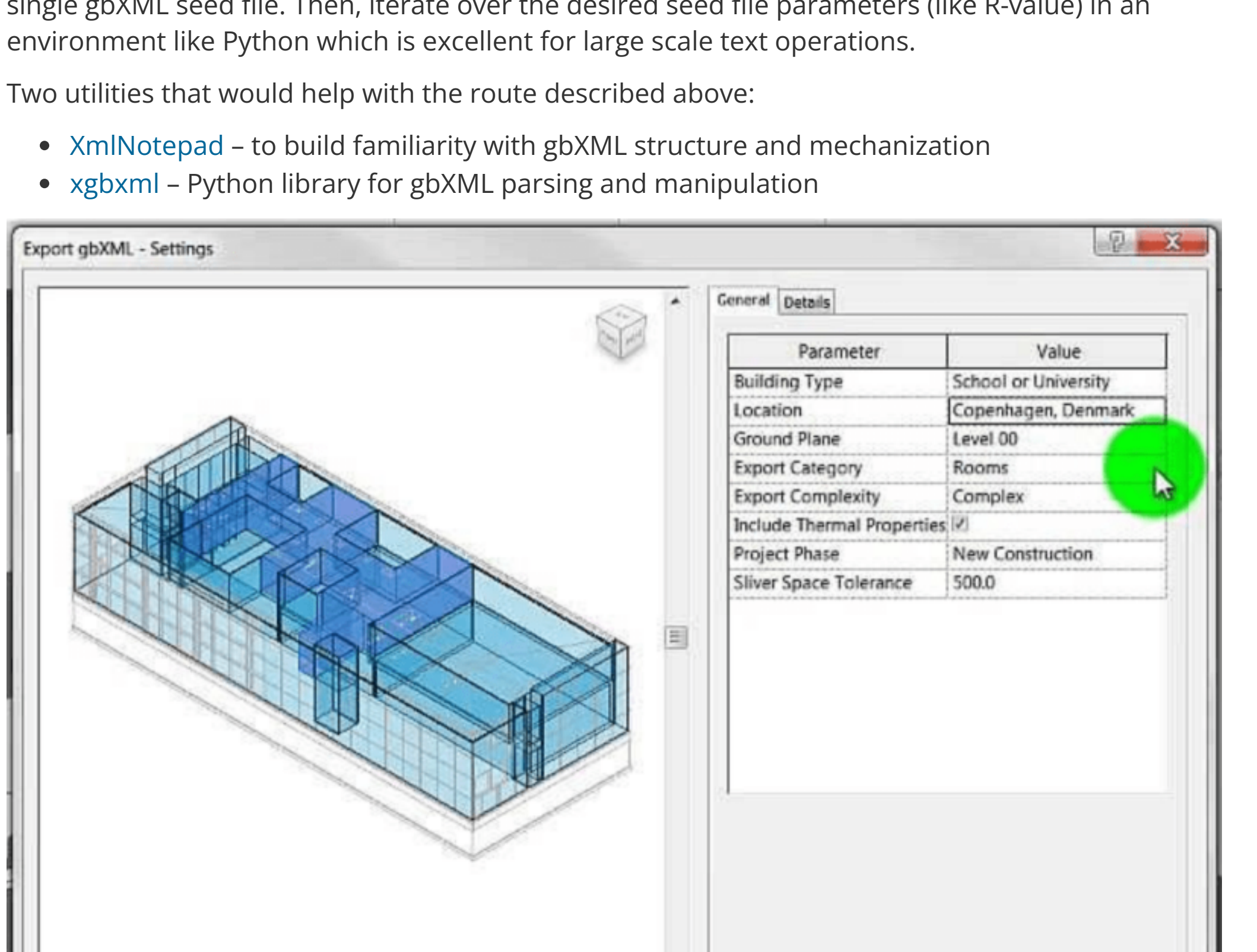
Revit is an end user product designed to be driven by a human being. Human beings are not expected to sit down and create 18000 models in one sitting. I suggest you implement an external executable that drives Revit using the code you shared above and monitors progress as you export results from the models you create.

Whenever Revit starts slowing down, take note of how far you got in processing, kill the process, restart Revit and continue from where you left off. This is a common approach to programmatically drive processes in batch mode that were not designed for it. You can also search The Building Coder for further hints on batch processing Revit documents. Alternatively, you could generate your 18000 models online using APS and DA4R.

Furthermore, based on the code snippet provided, it appears that only R-values are manipulated and not the underlying model geometry. If that's the case, it might be best to use Revit to export a single gbXML seed file. Then, iterate over the desired seed file parameters (like R-value) in an environment like Python which is excellent for large scale text operations.

Two utilities that would help with the route described above:

- XmlNotepad – to build familiarity with gbXML structure and mechanization
- xgbxml – Python library for gbXML parsing and manipulation



GbXML Energy Settings

Jake points to the same solution to answer another question as well, on gbXml export using energy settings:

Question: Sorry to revive a thread which has been solved more than one year ago, however the solution provided is not working for me as I am programming in Python.

I am using pyRevit to program functions in Python. When I use the code presented above, I get the following error:

```
  EnergyAnalysisDetailModelOptions.ExportMullions = False

  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  AttributeError: static property 'ExportMullions' of 'EnergyAnalysisDetailModelOptions' can only
be assigned to through a type, not an instance
```

If I understand this correctly, I am having a problem due to the type of variable in my code. However, Python does not allow the declaration of variables. How can I make the statement to become a type and not an instance?

Any idea how I can get past this issue without moving on to another language?

Answer: A nice example of Python `EnergyAnalysisDetailModelOptions` administration is discussed in the export of multiple gbXML models.

The relevant code snippet is this:

```
### Setting Energy Analysis parameters ###

opt=Analysis.EnergyAnalysisDetailModelOptions()
opt.EnergyModelType=Analysis.EnergyModelType.BuildingElement
opt.ExportMullions=False
opt.IncludeShadingSurfaces=False
opt.SimplifyCurtainSystems=True
opt.Tier=Analysis.EnergyAnalysisDetailModelTier.SecondLevelBoundaries
```

Automate FBX Export with SendKeys

We already shared a C# solution to handle a Revit dialogue using `Idling`, `DialogBoxShowing` and `SendKeys` to implement the TwinMotion dynamic link export FBX automatically.

Now Onur Er cleaned it up further in his updated answer:

Question: I want to export FBX using TwinMotion Dynamic Link. I would like to export FBX files from many Revit files. How I can use `PostCommand` and then handle the Windows forms on the export panel?

Answer: Thank you for sharing your solution. It saved me unbelievable amount of time, maybe days or weeks. Thank you VERY VERY MUCH!!! I cleaned the code and made it more readable in case someone needs it. My own Revit plugin calls this Twinmotion macro automatically after Revit starts up like this:

```
using System.Threading.Tasks;
using Autodesk.Revit.UI;
using System.Windows.Forms;
using Autodesk.Revit.UI.Events;

namespace YourNamespaceHere
{
  public class Class2 : IExternalApplication
  {
    UIControlledApplication UIControlledApplication;

    public Result OnStartup(UIControlledApplication Application)
    {
      UIControlledApplication = Application;
      UIControlledApplication.Idling += Application_Idling;

      return Result.Succeeded;
    }

    public Result OnShutdown(UIControlledApplication Application) => Result.Succeeded;

    void Application_Idling(object Sender, IdlingEventArgs E)
    {
      UIControlledApplication.Idling -= Application_Idling;

      var UIApplication = (UIApplication)Sender;

      MyMacro(UIApplication);

      //TaskDialog.Show("Application_Idling", Sender.GetType().FullName);
    }

    void OnDialogBoxShowing(object Sender, DialogBoxShowingEventArgs Args) => ((TaskDialogShowing
EventArgs)Args).OverrideResult((int)TaskDialogResult.Ok);

    static async void RunCommands(UIApplication UIapp, RevitCommandId Id_Addin)
    {
      UIapp.PostCommand(Id_Addin);
      await Task.Delay(400);
      SendKeys.Send("{ENTER}");
      await Task.Delay(400);
      SendKeys.Send("{ENTER}");
      await Task.Delay(400);
      SendKeys.Send("{ENTER}");
      await Task.Delay(400);
      SendKeys.Send("{ESCAPE}");
      await Task.Delay(400);
      SendKeys.Send("{ESCAPE}");
    }

    void MyMacro(UIApplication UIapp)
    {
      try
      {
        var Name = "CustomCtrl_%CustomCtrl_%Twinmotion 2020%Twinmotion Direct Link%ExportButton";
        var Id_Addin = RevitCommandId.LookupCommandId(Name);

        if (Id_Addin != null)
        {
          UIapp.DialogBoxShowing += OnDialogBoxShowing;

          RunCommands(UIapp, Id_Addin);
        }
      }
      catch
      {
        TaskDialog.Show("Test", "error");
      }
      finally
      {
        UIapp.DialogBoxShowing -= OnDialogBoxShowing;
      }
    }
  }
}
```

Thank you, Onur Er!

RFA Export to MongoDB

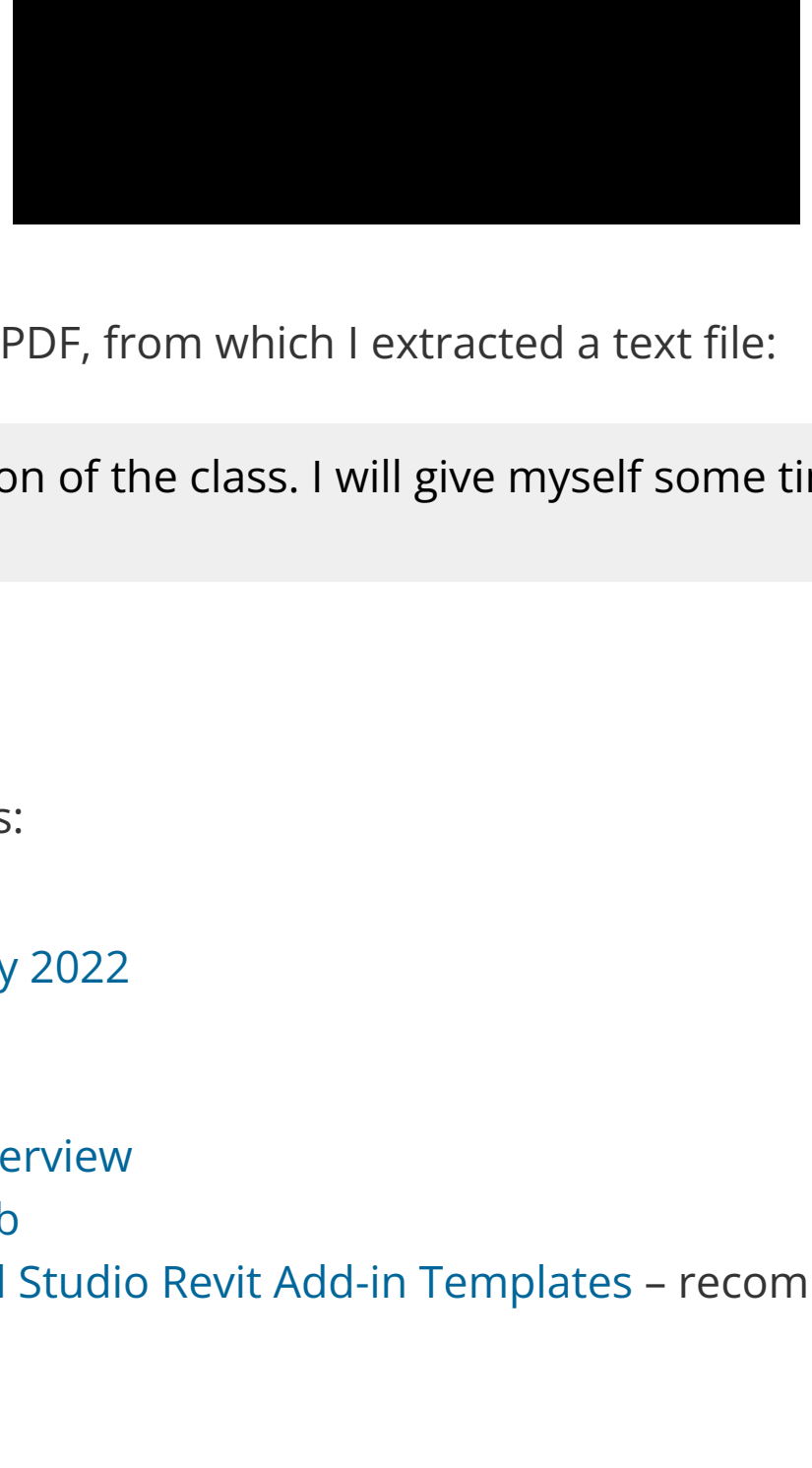
To wrap up, Eduardo Lalo Ibarra of Mexico City shares one of his favourite classes built with #VSC and #MongoDB to facilitate the export of data from Revit families:

The class implementation is encoded in the attached image files on LinkedIn:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace exportcloudMDB
{
    /// <summary>
    /// Door class with get/set autoproperties
    /// </summary>
    public class Door
    {
        public string _id { get; set; }
        public string FamilyType { get; set; }
        public string Mark { get; set; }
        public string DoorFinish { get; set; }

    }
}
```



Eduardo also provides it as a PDF, from which I extracted a text file:

> I share the construction of the class. I will give myself some time to share the whole process.

- mongodb_export.pdf
- mongodb_export.txt

Here is his useful list of assets:

- MongoDB Documents
- Visual Studio Community 2022
- Revit API docs
- Revit SDK
- My First Revit Plug-in Overview
- Create account in GitHub
- The Builder Coder Visual Studio Revit Add-in Templates – recommendation: clone the repository

Many thanks, Eduardo!

Posted on July 27, 2023 at 11:00 in Analysis, Export, FBX, gbXML, MongoDB, Python, Win32 | Permalink
Tags: akn_include, Jeremy Tammik, Revit API

Categories

Recent Posts

Archives