Workbench and Excel, Part 2: Driving Workbench From Excel with Python

by **Eric Miller** July 13, 2011 1:42 pm **Leave a comment** **The Focus**

For a couple of releases now you have been able to use the very common scripting language **python** to script applications that are considered Workbench native: the project page, Engineering Data, DesignXplorer, and the Parameter Manager. This is pretty cool and something that we will cover in a future **The Focus** article. But one thing that most people don't know is that python also works well with Microsoft Excel. So if it works with Excel and Workbench, you should be able to use it to connect the two.

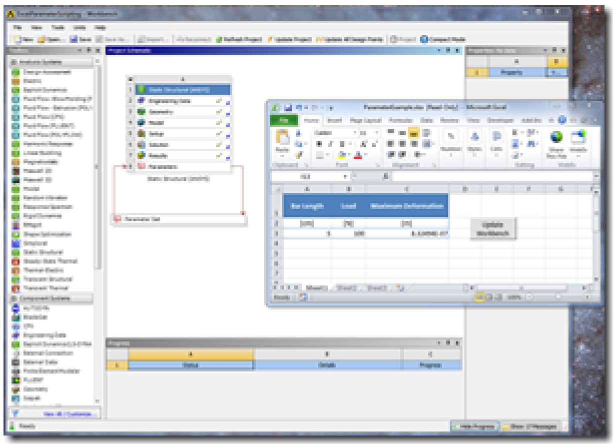
Now before you get too excited, realize that ANSYS Mechanical, CFX, FLUENT, CFX Post and most other solvers and post processors that launch from Workbench are not workbench native applications, so they don't speak python. So no, this is not a replacement for PADL or CCL. They are called "data integrated" applications in that they share data through the Workbench interface, but they have their own software structure that is unique, and their own scripting language.

So to take advantage of this python commonality you have to connect to one of the r
applications, and the parameter manager makes the most sense. And fortunately th

kind technical people at ANSYS, Inc. have provided a great example in the help system to do just that. And in doing so they have cut the length of this article in half.

ANSYS Documentation Example

You can find the example under: Workbench // Scripting Guide // Using Scripting in ANSYS Workbench // Usage Examples // Updating a Workbench Project and Parameters from Excel. It is a very simple example, a cantilever beam, but it shows the basics and is a great place to start.



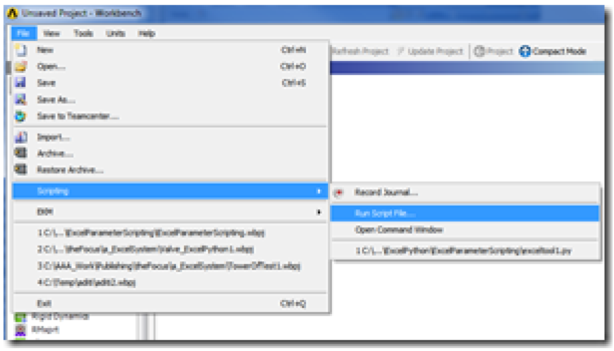
Read the manual page completely, especially the notes at the bottom. They point out the key things you need to know. The files you need are in the documentation, but you need to dig up the excel/workbench files and copy the python script and get rid of the line numbers. I've put them all together in a zip file to make things easier:

ExcelPythonDocExample.zip

There are a few things worth discussing on this example before we move on to something more complicated.

How to Run It

One thing the example leaves out is the details on how to run it. To start, you need to copy all the files to one directory: the python script, the workbench database, the workbench files, the excel file. Then (it does point this out) you need to set the working directory variable in the script to the directory with all the files. Now it is time to run. To do this open up Workbench with a new project (or File->New if you already have Workbench open). Now run the script with File->Scripting->Run Script File…



This will launch Excel and Workbench and add the command to execute the update script when you push the button. So, what next? Change the value for Bar Length or Load and press the button. This should cause Workbench to update and the Maximum Deformation value to change to Updating… and then the new value when the run is done. And that, in a nutshell, is the basics of making the two tools talk.

How it Works

The documentation does a pretty good job of explaining the details. The important take away is that you have to do a couple of key steps in any program that is going to do this sort of thing:

1. Tell python to load the Excel stuff
2. Set up the working directory, the routines all need absolute paths to work
3. Grab cell's from excel, grab params from Workbench, and set one to the other as needed
4. Do an update
5. Provide a mechanism to start the process (the button in this case)

If you have these steps, you can make Excel and ANSYS talk.

Where Things Run

An important thing to get your head around is where the script is running, or more accurately, what process is reading and executing the script. The python interpreter is actually in the Workbench environment. When you run the python script you are not running it in Excel or in the operating system. It is running through a interpreter inside Workbench.

This is important because it controls what information is automatically available to the script as well as how you launch it. In this example, you launch from Workbench and it starts up Excel, does stuff in Excel, and then assigns a python routine to the button.
the button is clicked, you are still running a python script in Workbench – it is just ta
to Excel.

Limitations in the Example

There are two big limitations with this example when it comes to using it on a real problem. First, it is not general enough because you have to hard code in file names, directory locations, cell locations and strings. This is not a huge deal because if you are going to use an excel based simulation tool over and over, you only have to set it up once and go.

The other problem is that the action to update the spreadsheet and model is assigned as an OLEObject to the button. Unfortunately this is only active when the script is running. Once it is done the click action is removed, and the button becomes useless… allowing for only one use of the button. This is a significant issue.

A More General and Complex (Sort Of) Example

A common potential usage for this sort of tool would be to provide a user with an Excel based tool that they would use to design a part and see the results. For this example we are going to use a very complicated tool for designing towers with a rectangular cross section. The input variables are the length, width, and height of the tower as well as a pressure applied to one face. The output variables are the deflection from the pressure and the first 10 natural frequencies.

Of course a real example would have more complex input and output values, and the spreadsheet would probably do a lot more calculations, perhaps even optimization.

This example avoids the button problem by actually running Workbench from Excel by launching it from the command console and supplying the project file AND the python script as startup parameters to Workbench. Workbench loads the project, runs the script, then exits.

The Spreadsheet

The spreadsheet is called TowerTool1 and it is a macro (xlsm) spreadsheet. You can download it here:

TowerTool1.xlsm

The sheet is not that different from the ANSYS provided example. It has cells for input and output parameters. It also has a button to update the system. The first thing that is different is that it uses named ranges for the input and output cells. To see the nam
to Formulas->Defined Names->Name Manager. I used “Create from Selection” to automatically assign the text in Column B to the values in Column C an D for each ro

Also note the units. Because Workbench parameters (with the exception of Design Modeler parameters) are unit sensitive, you can specify the units in those columns and use them to do unit conversion as needed. In this example only the units on the Pressure are implemented to show how this is done.

There is also a little Drop Down Yes/No choice for the user to decide if they want to save the project file or not when they are done running.

The button is also different from the ANSYS example. Instead of being assigned an action by the python script, it executes a very simple Visual Basic routine:

```
Private Sub CommandButton1_Click()
retval = Shell("C:\Program Files\Ansys
Inc\V130\Framework\bin\win64\runwb2 \
-X -R updateWB2.py -F ExcelTower1.wbpj", vbNormalFocus)
End Sub
```

The line uses the Shell function to execute a system level command in Windows. The command it runs is the command line way to start Workbench.

The –X says run interactive and exit when any given command line options are completed.

The –R gives a python script to be run, and this is our python script that is presented in detail below.

The –F specifies that project file to be executed. As an aside, you could make this program more general by having a cell in the spreadsheet hold that file name and then substituting that value in the execution string.

The ANSYS Model

This extremely complex model can be found in the following archived project file:

ExcelTower1.wbpz

The model contains a DesignModeler solid, a static structural solution, and a modal analysis. The key aspect of the model are the parameters.

The outline for the parameter manager looks like this:

The various parameters were set up in each system in the normal way, by clicking on the parameter box next to them for each value you want to supply or return. The key thing to note here are the Names (P1-P15) for the parameters, because this is what our python script will use to change or retrieve values before and after the model update.

The Script

The actual script we are using is shown below. Comments have been added to describe important steps. You can download it, without the line numbers, here:

updateWB1.py

Please don’t laugh too hard. I ran out of time to convert the code into something more concise using do loops and variables. The whole thing could be made much more general if you know python better than I do.

```python
# IronPython imports to enable Excel interop,
import clr
clr.AddReference("Microsoft.Office.Interop.Excel")
import Microsoft.Office.Interop.Excel as Excel
from System.Runtime.InteropServices import Marshal

# import system things needed below
from System.IO import Directory, Path
from System import DateTime

# use the ANSYS function GetProjectDirectory to figure out what directory you are in
# and set that to the current directory
projDir = GetProjectDirectory()
Directory.SetCurrentDirectory(projDir)

# Open up a log file to put useful information in
logFile = open("TowerTool.log","w")

# Put a header in the log file
logFile.write("================================================\n")
logFile.write("Tower Tool Run Log File\n")
logFile.write("================================================\n")
logFile.write("Start time: " + DateTime.Now.ToString('yyyy-mm-dd hh:mm:ss') + "\n")
logFile.write("Proj Dir: %s\n\n" % projDir)


```

```
# Use the Excel GetActiveObject funtion to get the object for the excel session
ex = Marshal.GetActiveObject("Excel.Application")
# Make Excel visible
ex.Visible = True
# Define the active workbook and worksheet
wb = ex.ActiveWorkbook
ws = wb.ActiveSheet

# In Excel: Grab values for the cells that we want data from (input cells)
length = ws.Range["Length"](1,1).Value2
width = ws.Range["Width"](1,1).Value2
height = ws.Range["Height"](1,1).Value2
press = ws.Range["Pressure"](1,1).Value2
upress =  ws.Range["Pressure"](1,2).Value2

# In Excel: See if the user wants to save the project after the update
saveit = ws.Range["Save_Project"](1,1).Value2

# In Workbench: Grab the parameter objects for the input values
lenParam = Parameters.GetParameter(Name="P1")
widParam = Parameters.GetParameter(Name="P2")
hgtParam = Parameters.GetParameter(Name="P3")
prsParam = Parameters.GetParameter(Name="P5")

# In Workbench: Get the object for the deflection parameter vlue
defParam = Parameters.GetParameter(Name="P4")

#In Workbench: Set the value of the input parameters in Workbench using the values
#   we got from Excel
lenParam.Expression = length.ToString()
widParam.Expression = width.ToString()
hgtParam.Expression = height.ToString()
prsParam.Expression = press.ToString() + " [" + upress + "]"
```

```

# Set the output values to "Calculating..." since they no longer match the input values
ws.Range["Max_Bending_Distance"](1,1).Value2 = "Calculating..."
ws.Range["Mode_1"](1,1).Value2 = "Calculating..."
ws.Range["Mode_2"](1,1).Value2 = "Calculating..."
ws.Range["Mode_3"](1,1).Value2 = "Calculating..."
ws.Range["Mode_4"](1,1).Value2 = "Calculating..."
ws.Range["Mode_5"](1,1).Value2 = "Calculating..."
ws.Range["Mode_6"](1,1).Value2 = "Calculating..."
ws.Range["Mode_7"](1,1).Value2 = "Calculating..."
ws.Range["Mode_8"](1,1).Value2 = "Calculating..."
ws.Range["Mode_9"](1,1).Value2 = "Calculating..."
ws.Range["Mode_10"](1,1).Value2 = "Calculating..."

# Now let Workbench go to town and update the systems using the new parameter values
logFile.write("Updating Project\n")
Update()

# If asked for, save the project

if saveit == "Yes":
    logFile.write("Saving Project\n")
    Save()

# Assign the value of the Excel deflection cell output deflection from Workbench
ws.Range["Max_Bending_Distance"](1,1).Value2 = defParam.Value.Value

# Now go through the value of each natural frequency in Workbench and
#   set the corresponding cell in Excel
#    This could be made more general or at least more concise by using a do loop
#    Also note that instead of getting the objects, then the values the two steps are
#    combined for these values
ws.Range["Mode_1"](1,1).Value2 = Parameters.GetParameter(Name="P6").Value.Value
ws.Range["Mode_2"](1,1).Value2 = Parameters.GetParameter(Name="P7").Value.Value
ws.Range["Mode_3"](1,1).Value2 = Parameters.GetParameter(Name="P8").Value.Value
ws.Range["Mode_4"](1,1).Value2 = Parameters.GetParameter(Name="P9").Value.Value
ws.Range["Mode_5"](1,1).Value2 = Parameters.GetParameter(Name="P10").Value.Value
ws.Range["Mode_6"](1,1).Value2 = Parameters.GetParameter(Name="P11").Value.Value
ws.Range["Mode_7"](1,1).Value2 = Parameters.GetParameter(Name="P12").Value.Value
ws.Range["Mode_8"](1,1).Value2 = Parameters.GetParameter(Name="P13").Value.Value
ws.Range["Mode_9"](1,1).Value2 = Parameters.GetParameter(Name="P14").Value.Value
ws.Range["Mode_10"](1,1).Value2 = Parameters.GetParameter(Name="P15").Value.Value

# Done!  Close the log file and move on
logFile.write("End time: " + DateTime.Now.ToString('yyyy-mm-dd hh:mm:ss') + "\n")
logFile.close()
```

Execution

Execution is pretty simple. When the user pushes the Update system button the program goes off and launches Workbench with the script as input. The script changes the parameters, updates the model, saves if asked for, updates the values in the spreadsheet, then Workbench exits.

One key thing to know is the issue with working directories. The Excel file needs to be in the same directory as the script and the project. Sometimes Excel changes the working directory (say you opened a different spreadsheet). If you get an error on execution that Workbench can't find the project file, you just need to go back to Excel and do a Save As in of your file on top of itself. That fixes the problem. More advanced users can use VBA to control directories.

Resources

The first resource for doing this type of work is the ANSYS Workbench->Scripting Guide in the online help.

ANSYS uses a flavor of python called IronPython. They have a great web site at:

www.ironpython.info

Specific info on the connection to Excel can be found at:
http://www.ironpython.info/index.php/Interacting_with_Excel

Thoughts and Suggestions

Both examples presented are pretty simple, but they have the core of what you need to drive Workbench from Excel. It really is not that hard to do and can provide a nice interface for users who don't know Workbench.

Some areas for improvement and suggestions for a real application are:

- Use jScript in Workbench to add a button that runs a script that attaches to an open Excel file and update based on the current values in that Excel file. So instead of running Workbench in batch from Excel, pressing the button would update everything. That seems to be a more effective way to do it. (maybe a quick article in the future if Matt help me figure out how to add a button quickly…)
- Do more with VBA or Python in the Excel file to make you Excel based tool more sophisticated.
- There is no error checking in this example… very bad. For a real world tool a lot more code should be added to check things and to capture errors from various functions.
- As one can imagine, this can be extended to other programs. A good example would be an optimization tool or another "calculating" tool like MathCAD or Matlab.

Related

Workbench Scripting: Using Forms in your Script
September 7, 2011
In "The Focus"

Starting ANSYS Products From the Command Line
February 29, 2012
In "The Focus"

Workbench and Excel, Part 1: Using the Excel Component System
June 9, 2011
In "The Focus"

‹ Dust In the Wind: Stress and Strain Results in Workbench Mechanical vs. Mechanical APDL

CAE Market Fully Analyzed… Is that Cannibalism? ›

Contact PADT
1-800-293-PADT
info@padtinc.com

SEARCH

Search

LINKS

PADT Website
Contact PADT
PADTMarket.com
Manage PADT Subscriptions

SUBSCRIBE TO OUR BLOG

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 1,804 other subscribers

Email Address

Subscribe

UPCOMMING EVENTS

01/19/2022 - 01/21/2022
Arizona Photonics Days

04/04/2022 - 04/07/2022
37th Space Symposium
Arizona Space Industry Booth

04/12/2022 - 04/14/2022
D&M West | MD&M West

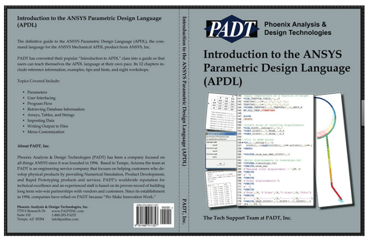
3D PRINTING GLOSSARY

3DP GLOSSARY

3D PRINTING GLOSSARY

A RESOURCE FOR ADDITIVE MANUFACTURING

INTRODUCTION TO THE ANSYS PARAMETRIC DESIGN LANGUAGE


PADT Phoenix Analysis & Design Technologies
Introduction to the ANSYS Parametric Design Language (APDL)
The Tech Support Team at PADT, Inc.

Learn APDL with this workshop based book written by PADT's Technical Support Team. The new and improved Second Edition contains additional chapters on APDL Math and APDL in ANSYS Mechanical.

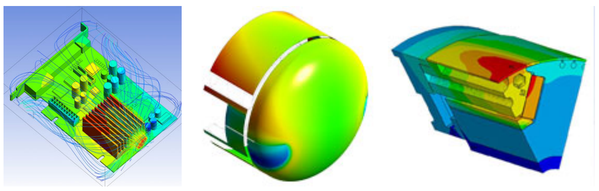
Now available on Kindle as well as paperback.

More…

SIMULATION SERVICES

PADT's simulation engineers are true experts in virtual prototyping. Trust the people you come to for ANSYS expertise to handle your simulation outsourcing needs.

Structural, Thermal, Fluid, Electromagnetic, and Systems.



LOOKING FOR ANSYS TRAINING?

Let the people who write "The Focus" train you and your team. Check out our **schedule** or **contact us** to set up a class at your place or something custom.

PODCAST: ALL THINGS ANSYS

All Things ANSYS Podcast

Presented by PADT

ANSYS ACADEMIC PROGRAM

Ansys ACADEMIC PROGRAM

Learn about software for students and research

RSS SUBSCRIBE:

RECENT POSTS:

- Surprise – 2021 Turned out to Be a Lot Like 2020
- All Things Ansys 102: Electronics Reliability Updates in Ansys 2021 R2
- Simulating an Electro-Permanent Magnet (EPM) using Ansys Maxwell
- Ansys Pro – Premium – Enterprise Electronics Licensing Adjustments
- All Things Ansys 101: Additive & Structural Optimization Updates in Ansys 2021 R2

CATEGORIES

- Additive Manufacturing
- Ansys
- ANSYS Discovery
- ANSYS Energy Innovation Campaign
- ANSYS R 18
- Carbon
- Education
- Events
- Flownex
- Fun
- Getting To Know PADT
- News

Nimbix

PADT Medical

PADT Startup Spotlight

Podcast

Product Development

Publications

Startups

Stratasys

Stratasys Marketing

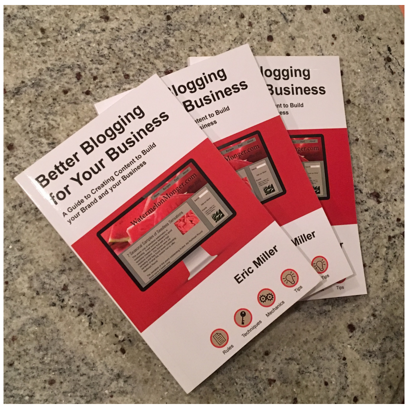
The Focus

Uncategorized

Webinar

Contact PADT
1-800-293-PADT
info@padtinc.com

THINKING ABOUT A BLOG FOR YOUR COMPANY?


Better Blogging for Your Business
A Guide to Creating Content to Build your Brand and your Business
Eric Miller

Where do you start? How do you keep it going? Where do I get ideas for posts? Should I use humor?

These questions and many others are answered in the book that was inspired by the success of PADT's blog:

Privacy - Terms

Better Blogging for your Business

Available now as a Kindle book or softcover on Amazon.

PADT EMAIL

Manage how PADT Emails You

Want to receive Emails from PADT? Please select any of the basic topics below to tell us what you are interested in. We promise to keep it simple sharing news, opportunities, and useful information. You can come back and change your settings whenever you need to.

* **Email**

State/Province

Company

* **Email Lists**

- ☐ PADT Additive & Advanced Manufacturing Email List
- ☐ PADT General Information Email List
- ☐ PADT Product Development and Testing Email List
- ☐ PADT Simulation Email List

Sign Up!

Contact PADT
1-800-293-PADT
info@padtinc.com

SEARCH

Search

LINKS

PADT Website
Contact PADT
PADTMarket.com
Manage PADT Subscriptions

ANSYS STARTUP PROGRAM

LOOKING FOR ANSYS TRAINING?

Let the people who write "The Focus" train you and your team. Check out our **schedule** or **contact us** to set up a class at your place or something custom.

TRYING TO FIND A COMPUTER BUILT FOR SIMULATION?

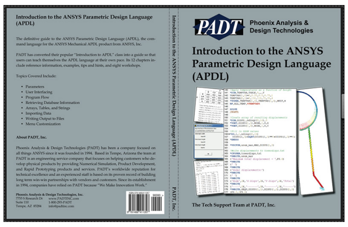
Workstations, Servers, and Clusters designed by

Privacy - Terms

PADT specifically for simulation users.

Learn More

INTRODUCTION TO THE ANSYS PARAMETRIC DESIGN LANGUAGE


PADT Phoenix Analysis & Design Technologies

Introduction to the ANSYS Parametric Design Language (APDL)

The Tech Support Team at PADT, Inc.

Learn APDL with this workshop based book written by PADT's Technical Support Team. The new and improved Second Edition contains additional chapters on APDL Math and APDL in ANSYS Mechanical.

Now available on Kindle as well as paperback

More...

RECENT POSTS

Surprise – 2021 Turned out to Be a Lot Like 2020

All Things Ansys 102: Electronics Reliability Updates in Ansys 2021 R2

Simulating an Electro-Permanent Magnet (EPM) using Ansys Maxwell

Ansys Pro – Premium – Enterprise Electronics Licensing Adjustments

All Things Ansys 101: Additive & Structural Optimization Updates in Ansys 2021 R2

CATEGORIES

Additive Manufacturing

Ansys

Connect with PADT