Phoenix Analysis &
Design Technologies

WORKSHOPS FOR WEBINAR: INTRO TO WORKBENCH FRAMEWORK SCRIPTING

Created by: Eric Miller
PADT, Inc.
3/8/2012

More Information: Contact PADT at 480.813.4884 or support@padtinc.com

Contents

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/us/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Workshop 1: Record and Use a Python Macro for Changing Density

In this workshop you will do the most basic tasks involved in scripting the Workbench framework: you will record a simple journal file, review it, make a change, and then play it back. To keep things simple we will be modifying the density material property.

This is a good first example because it is a simple version of what we will often be doing: You want to automate some step in the program and instead of digging through the documentation to figure out how to do it; we do it interactively and record the commands that repeat the interactive actions. Now we just need to modify the parts we want to change or control. Much simpler.

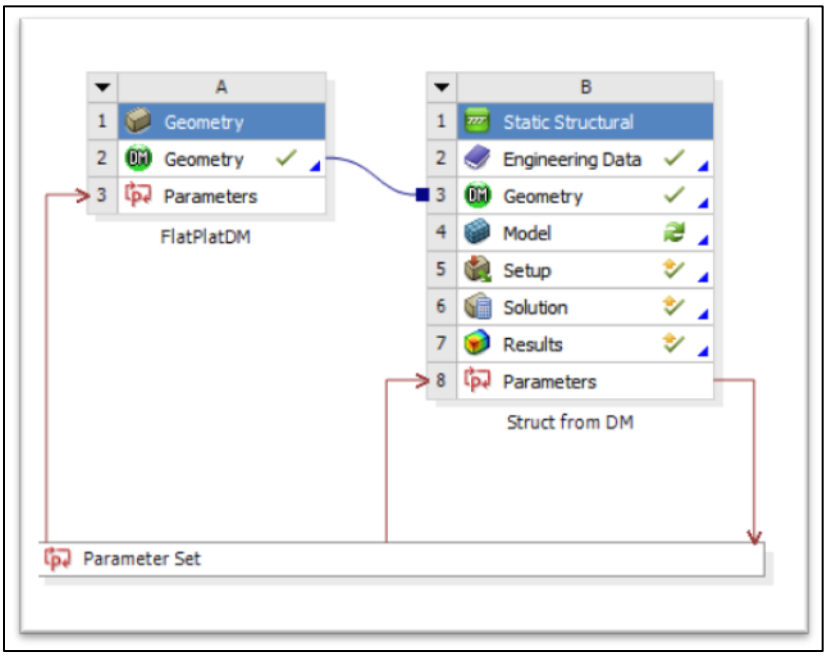
Files needed:

Project File:	`Flat-Plate-1.wbpj`
Resulting Script:	`ws1-matmod-mod1.wbjn`

Steps:

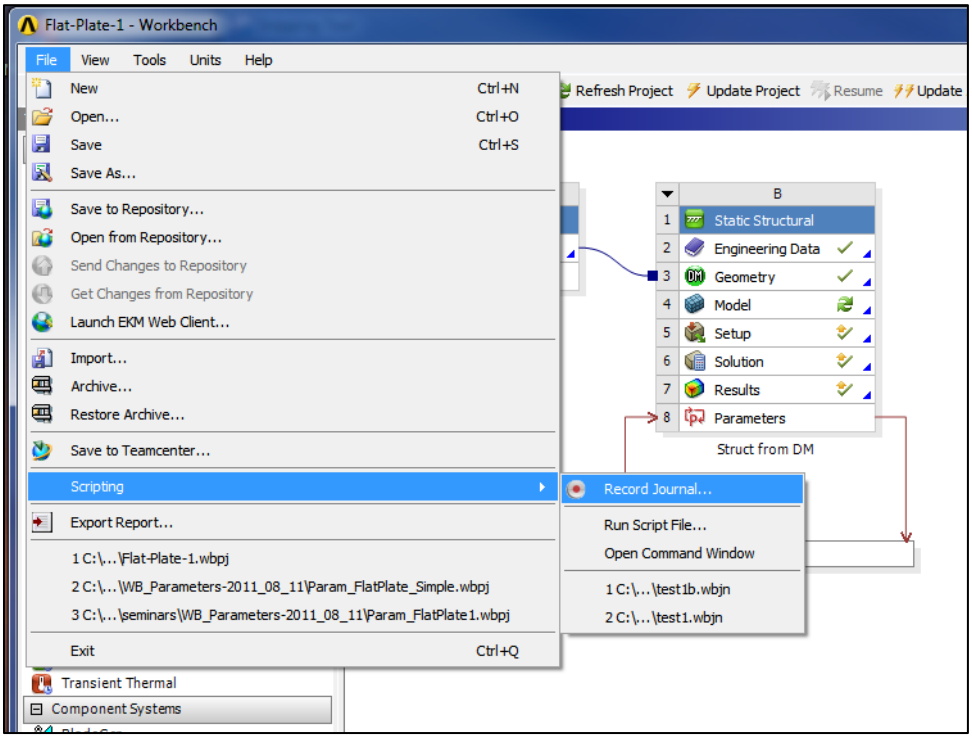
1. Start Workbench up
2. Go to File->Restore Archive
3. Choose Flat-Plate-1.wbpz
4. Save as `Flat-Plate-1.wbpj` in whatever directory you want to work in.

5. You should have a project that looks like this:



If you want, explore the model and get a feel for it. Very simple so we can focus on scripting and not the model.

6. Now it is time to record our journal file. Remember that every step you take in the GUI that impacts your project gets recorded in a journal file, if you tell Workbench to record.



We do that by going to File->Scripting->Record Journal

7. You will prompted for a file name and where to put the file. Make sure you are in your working directory and always pick a descriptive name.

   For this workshop, save it as: `ws1-matmod-record.wbjn`

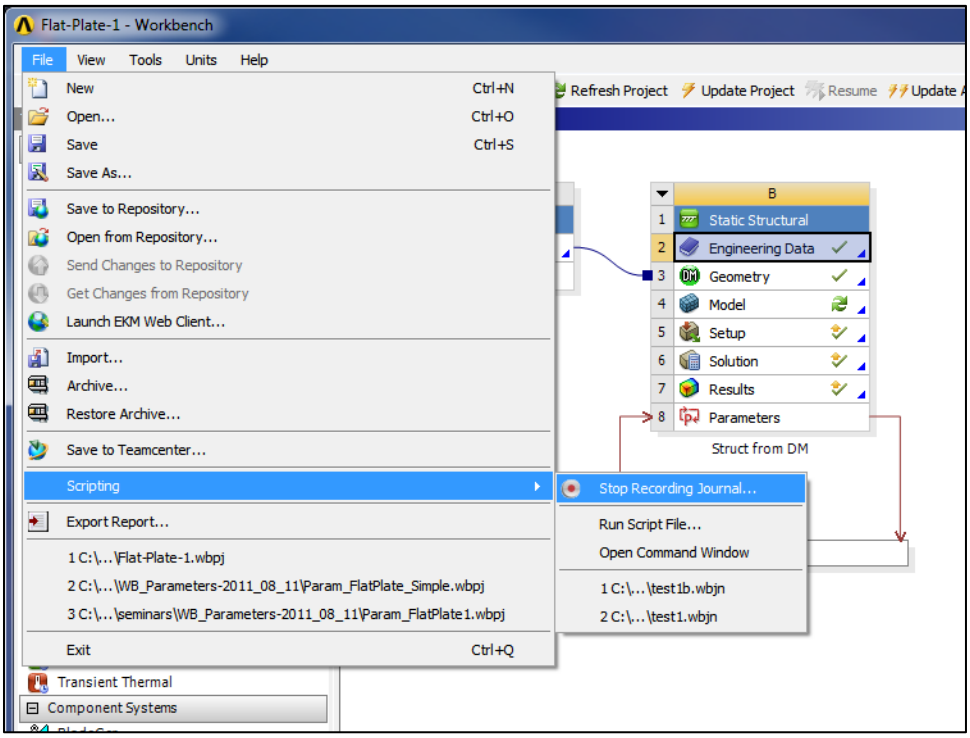
8. And now we come to the reason why we are here. We want to change the density of our material so we get a recording on how to do that.
   Right Mouse Button (RMB) on B2- Engineering Data and choose Edit…

9. This will bring up the Engineering data outline and properties. As you can see the only material in the model is the default Structural Steel. Looking down at properties you should see that Density is the first entry. Change the value from 7850 kg m^-3 to 7000 kg m^-3

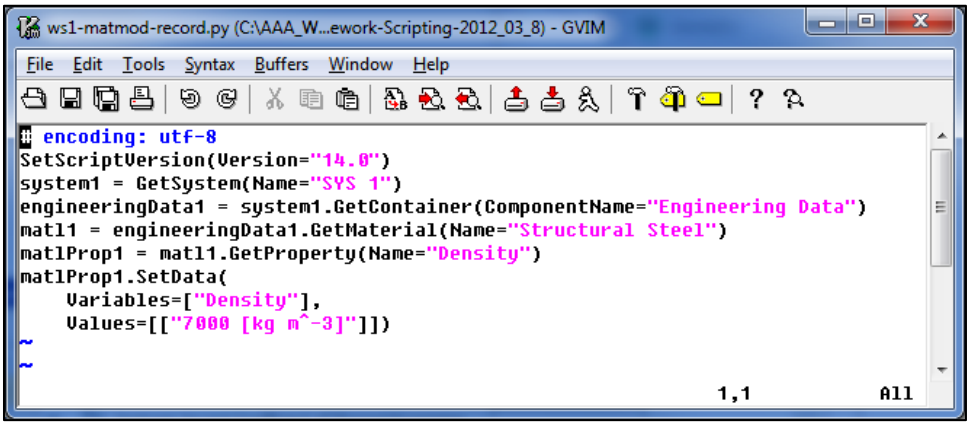
Properties of Outline Row 3: Structural Steel

	A	B	C	D	E
1	Property	Value	Unit		
2	Density	7000	kg m^-3		
3	Isotropic Secant Coefficient of Thermal Expansion				
6	Isotropic Elasticity				
12	Alternating Stress Mean Stress	Tabular			
16	Strain-Life Parameters				
24	Tensile Yield Strength	2.5E+08	Pa		
25	Compressive Yield Strength	2.5E+08	Pa		
26	Tensile Ultimate Strength	4.6E+08	Pa		
27	Compressive Ultimate Strength	0	Pa		

10. At the top of the page, click on “Return to Project” to go back
11. Turn off the journal recording by going to File-> Scripting->Stop Recording Journal…



12. Open up the journal file in your favorite text editor and take a look:



This is as simple as it gets and shows a lot of what you need to know to script in the Workbench Framework.

a. The first two lines are header info that tells the program version information.

b. The next line grabs the system you are working with. If you go back to your project page in click on the Static Structural system and look at properties, you will note that the name is “Sys 1.” That is the internal name and how you refer to everything in that

system. On this line we are grabbing the object to the container and storing it in system1

c. Now we use that system1 object and getting an object for the engineering data stored in the system: engineeringData1. Note that there is no engineering data container. Instead Workbench stores all the containers together and you use the name, Engineering Data in this case, to get to it.

d. This object is then used to find the object in engineeringData1 that contains a material with a name of "Structural Steel" and we store that in mat1

e. As you would expect, the next thing to do is use mat1 to get at the density. The GetProperty() method is used with name="Density." That is given a name of matlProp1. We have now achieved what we wanted to do. We now have an object with a name we know that contains the properties we want to use. Note how we used GetContainer(), GetMaterial(), and GetProperty() methods to find our way down to what we wanted.

f. With that object, matlProp1 we can now make changes using the SetData() method.

   i. SetData() takes two arguments: the variable you are setting and the values. Both can be an array, to allow for tabular data. By saying "Density" and not "Temperature","Density" you are telling workbench this is not a temperature dependent property table… we will use that next.

13. Save your file with a new name: ws1-matmod-mod1.wbjn

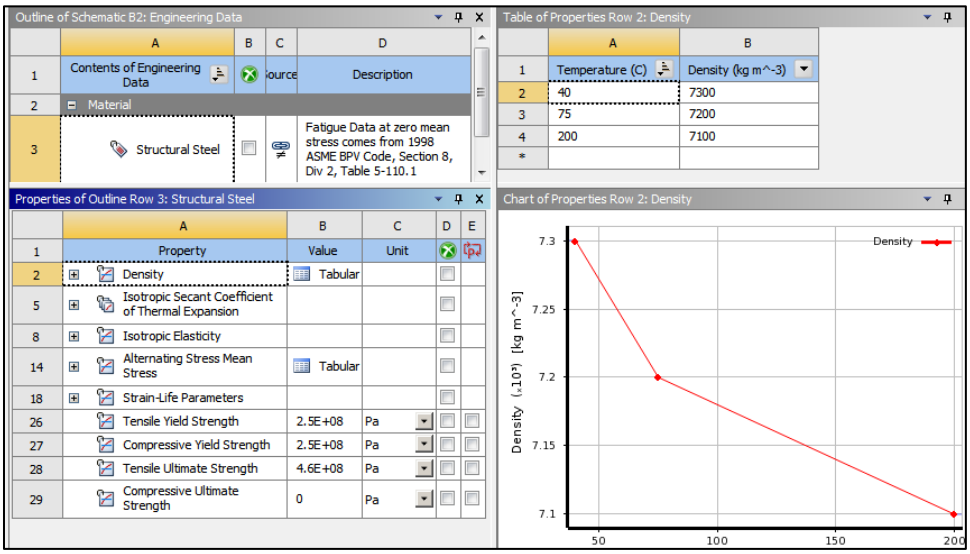
14. To change things lets modify the variables and values to be a temperature dependent table. Replace the last three lines of your file with these lines:

```
tmps = ["40 [C]","75 [C]", "200 [C]"]
dens  =  ["7300 [kg m^-3]", "7200 [kg m^-3]", "7100 [kg m^-3]"]
matlProp1.SetData(
    Variables=["Temperature","Density"],
    Values=[tmps, dens])
```

15. Save your file then use File->Scripting->Run Script File… and choose the file you just saved.

16. If that generated an error, check for the line number in the error message and look at your script. You probably forgot a bracket, parenthesis, comma, or a double quote. Common problem with Python arrays.

17. If it worked, your density should now be a table and look like this:



18. Save your project.

19. Extra Credit: Add Bilinear Kinematic Hardening to your Model

    a. Use the record feature to add it using the GUI

    b. Add the results to the bottom of your existing script, skipping the parts that get the engineering data, and material objects.

    c. Note that it uses the CreateProperty() method to add Bilinear Kinematic Hardening to the material.

    d. Here is what the new lines should look like:

```
matlProp1 = matl1.CreateProperty(
    Name="Kinematic Hardening",
    Definition="Bilinear")
matlProp1.SetData(
    Variables=["Yield Strength"],
    Values=[["250000000 [Pa]"]])
matlProp1.SetData(
    Variables=["Tangent Modulus"],
    Values=[["20000000 [Pa]"]])
```

Workshop 2: Writing a Helper Functions for Setting Material Properties

Mainly because we find it annoying that it takes at least eight lines of code to change a density, one of the first things we did was write a helper function to set it in one line. You could even create a whole library that set all of the standard material properties, which will be the thing we do second in this workshop. Although simple, these will serve as a good example for making more complex, and more useful helper functions.

Files needed:

Project File:	`Flat-Plate-1.wbpj`
Resulting Script:	`ws2-matmod-func1.py`
	`mat-helpers.py`

Steps:

1. Start Workbench up
2. Go to File->Restore Archive
3. Choose Flat-Plate-1.wbpz
4. Save as `Flat-Plate-2.wbpj` in whatever directory you want to work in.
5. Open up your text editor with a blank file. Remember that tabs count in python, so be careful to make sure things are indented correctly.
6. Our first line is a defines our function. Start your script with this:
   ```
   def setDens(sysName,matName,densval):
   ```
   We have called the function SetDens() and it takes three arguments: The name of our system, the name of the material we want to change, and the density value we want to use.
   If this line doesn't make sense to you, go back to your python training material and study how functions work.
7. Save it as `ws1-matmod-func1.py`
8. Now let's use the arguments to work our way down to our material: GetSystem(), GetContainer(), GetMaterial(). Add these lines:
   ```
   sys = GetSystem(Name=sysName)
   ed  = sys.GetContainer(ComponentName="Engineering Data")
   mat = ed.GetMaterial(Name=matName)
   ```
   Note how we use the sysName and matName arguments and just use the generic "Engineering Data" for the container. This is because it will always be "Engineering Data"
9. Using the container for the material we want to change, we simply need to get the density property and change it:
   ```
   densProp.SetData(
   ```

```
Variables=["Density"],
Values=[densval])
```

10. We can read this file in to workbench and then use it all we want after that. But you probably have some typos (if you type like I do) and we need to practice using the console, so we will do that. Go to File->Scripting->Open Command Window…
11. Copy and paste the setDens() helper function into the command window. If there were no errors and you just get a prompt back, then you are pretty good. If not, read the error messages and correct your script until it works without an error.
12. Now type in: `setDens("SYS 1", "Structural Steel", "1234.34 [kg m^-3]")` This should reset your density for you. One line. Simple.
13. One of the nice things about python (or bad, depending on how tyrannical you are about your object oriented programming) is that parameters are global (there is one namespace). For the second part of this workshop we will do a more generic set of commands for setting any property, and we will take advantage of this fact to make it simpler.
    To start, save your `ws1-matmod-func1.py as mat-helpers.py`
14. To keep things simple we will establish two "global variables" curSysName and curMatName. That way we don't have to supply it every time we call our function.
15. Change that first line to a new function called modifyMats() and change the arguments so that all it expects is the property we want to change (propa and propb) and the value we want to set it to (val):
    `def modifyMats(propa,propb,val):`
    We need two prop's because many of the material properties are grouped under a "Property" and each one is called a "Variable". As an example, to set the Young's Modulus we need to change the Property of "Elasticity" and the Variable of "Young's Modulus".
16. So try and modify the macro to do what we want without looking at the next page. Remember to use propa for the Property and propb for the variables. When you think you have it paste it in and give a try with these commands:

```
curSysName = "SYS 1"
curMatName="Structural Steel"
modifyMats("Density","Density","7123")
modifyMats("Elasticity","Poisson's Ratio",".23")
modifyMats("Elasticity","Young's Modulus","20e6")
```

17. Here is the function you should end up with:

```
def modifyMats(propa,propb,val):
      sys = GetSystem(Name=curSysName)
      ed  = sys.GetContainer(ComponentName="Engineering Data")
      mat = ed.GetMaterial(Name=curMatName)
      propc = mat.GetProperty(Name=propa)
      propc.SetData(
            Variables=[propb],
            Values=[val])
```

18. Extra Credit:

    Use the record function to understand more complex material properties like the coefficient of thermal expansion. Then modify the modifyMats() routine to handle those.

Workshop 3: Writing Out Parameters to a File

The easiest and best way to interact with Workbench programmatically is to read and write parameters. So our next step is going to be writing a very simple script that writes all of the currently active parameters to a text file. It also introduces how you use python modules besides the workbench ones.

Files needed:

Project File: `Flat-Plate-1.wbpj`
Resulting Script: `ws3-writeParams.py`

Steps:

19. Start Workbench up
20. Go to File->Restore Archive
21. Choose Flat-Plate-1.wbpz
22. Save as `Flat-Plate-3.wbpj` in whatever directory you want to work in.
23. Take a look at the parameter manager. You should have these values:

Outline of All Parameters

	A	B	C	D
1	ID	Parameter Name	Value	Unit
2	Input Parameters			
3	FlatPlatDM (A1)			
4	P29	XYPlane.D5	0.25	
5	P2	DM_Length	2	
6	P8	DM_Thick	1	
7	P4	DM_Width	3	
8	Struct from DM (B1)			
9	P24	Mesh Element Size	0.2	in
10	P28	Pressure Magnitude	100	psi
11	P38	Solid	0	
*	New input parameter	New name	New expression	
13	Output Parameters			
14	Struct from DM (B1)			
15	P22	Equivalent Stress Maximum	1856.4	psi
16	P23	Total Deformation Maximum	0.00010608	in
17	P25	Mesh Elements	1750	
18	P26	Mesh Average	0.815	
19	P31	EPELY Maximum	1.7326E-05	in in^-1
20	P32	Stress Probe Equivalent (von-Mises)	743.02	psi
*	New output parameter		New expression	
22	Charts			

24. Open up a new blank text file and start by telling python to use the operating system module:

```
import os
```

25. Now we want to find out how to get to the list of all the parameters. There is no easy way to do this with a record, so we have to use the help. We want a parameter that lists all the parameters, so we look in the Namespaced Commands section of the manual. You will notice that each application that supports scripting has a unique set of commands that refer only to the data that is unique to each of them. We want the Parameters namespace.
Scroll down till you find something that looks good. In this case it is pretty obvious we want GetAllParameters(). We know that this is in the parameter namespace so to get a list of all the parameters we do: `myParams = Parameters.GetAllParameters()`. We will use a for-in loop to step through all the parameters.

26. But what is in myParams now. It is a list of parameters but how do we know what each parameter in the list holds? For that we need to the Data Containers documentation in the manual. There is one for every little nook and cranny of the workbench framework. We want Parameters. And if we scroll down, we will find Parameter. Under its list of Properties are all the things you might want to list out. Name, DisplayText, and Value are the most important.

Properties

Description

A human-readable description of the parameter.

Type string
Read Only No

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all data entities but is used only in those entities that present a label in the user interface.

Type string
Read Only No

ErrorMessage

An error message as a result of a validation or expression evaluation error.

Type string
Read Only Yes

Expression

The parameter definition as an expression. This property may be a constant expression, or it may depend on the values of other parameters.

Type string
Read Only No

ExpressionType

Specifies the type of paramter expression.

Type ExpressionType
Read Only Yes

Usage

Specifies how the paramter is used in the data model.

Type ParameterUsage
Read Only Yes

Value

The current value of the parameter. For derived parameters, this value is the evaluated expression result.

Type Object
Read Only Yes

ValueQuantityName

Gets the value quantity name if ValueSpec is a QuantitySpec. Null otherwise.

Type string
Read Only No

27. Just like in any programming language, we need to first open a file for writing, then write to it, and then close it. To do this in python we use the open() method then the file methods. If you are not familiar with this, take a look at the python documentation on docs.python.org for the open function (http://docs.python.org/library/functions.html#open). Go ahead and give this a try. Open a file for writing and store its container as outFile.

28. Now use the `for param in Parameters.GetaLlParameters():` to step through the parameters. Use an outFile.write() to write to the file. Then a outFlie.flush().

29. When you think you have it copy and paste into the command window to debug. Our version is shown here:

```
import os

def writeParams():
      outFile = open("c:/temp/foo.out","w")
      for param in Parameters.GetAllParameters():
            outFile.write(param.Name + ", " +
param.DisplayText + ", " + param.Value.ToString() + "\n")
      outFile.flush()
      outFile.close()

# End of writeParams()
writeParams()
```

Workshop 4: Putting a GUI on your Project

This is a workshop version of a posting on the Focus (www.padtinc.com/thefocus) presented here in more of a step by step form.

Files needed:

Project File: `Tower-of-Test-1.wbpj`
Resulting Script: `towerTool.py`

Steps:

1. Start Workbench up
2. Go to File->Restore Archive
3. Choose `Tower-of-Test-1.wbpz`
4. Save it as `Tower-of-Test-1.wbpj`
5. Any time you are doing a GUI based program, you should always start with "Hello World!" We will do that here as well to explore how to add the GUI tools to your script and how to use them in a basic way.

   For our "Hello World!" we will open up a window and display some text. That is it.

   Open up a blank file in your text editor to get started and call it ws4-helloworld.py.
6. We need to tell python that we want to use the .NET GUI tools (clr) and that we want to to access the forms part of those tools. As you can imagine, the clr module is huge so you don't just turn everything on. It uses the concept of an assembly to store subsets of the module. To start we want the System.Windows.Forms assembly. So start your script with:

```
import clr
clr.AddReference('System.Windows.Forms')
```

7. We are not done. Now we must tell python that we not only want to play with forms, but that we need to import an Application, Form, and Label module:

```
from System.Windows.Forms import Application, Form, Label
```

8. Now we can finally get started. Frist we want to make a form and a label:

```
form = Form(Text="Hello World Form")
label = Label(Text="Hello World!")
```

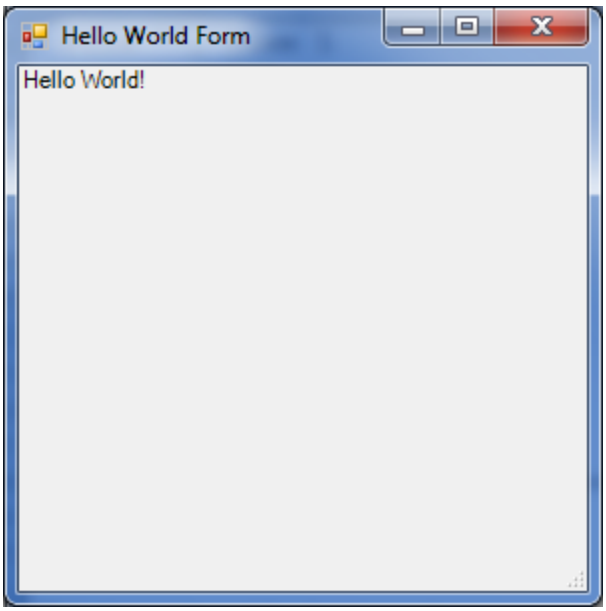
9. We now have a text label and a place to put it. But, they don't know about each other, so we need to tell the form to display the label:

```
form.Controls.Add(label)
```

10. And last but not least, we need to tell python that we want to show the form because it only exists as a description now. So add the following to the bottom of your script:

```
Form.ShowDialog(form)
```

11. To try this out, simply open up the command window in Workbench and paste it in. You could also read in the file but I like pasting because I get better feedback on my mistakes. This is what you should see:



Very exciting!

12. As you can see, we are missing some basic things that all user interfaces have: Input areas and control buttons. In order to add those you should first save your script as `towerTool.py`

13. Because we want to control where things are, we need to add the System.Driving assembly to our script. We then need to get the objects we need:
Button and Text Box from the System.Windows.Forms assembly
Point, and Content Alignment from the System.Drawing assembly
The first couple of lines of your script should look like this now:

```
import clr
clr.AddReference('System.Windows.Forms')
clr.AddReference('System.Drawing')
from System.Windows.Forms import Form, Label, Button,
TextBox
from System.Drawing import Point, ContentAlignment
```

14. The next part was the hardest for me to understand. The way Windows handles events (mouse clicks, key board clicks, etc…) you have to make a class out of your dialog box. This gives Windows a container to deal with when you interact with it. So we need to make a TowerForm class.

At this point it is hard to do a "workshop." You can either try and figure out what is needed on your own, or copy what we have below. Either way, try and think about what it all means as you put it in. Note the inclusion of more assemblies into your script, including fonts and units.

```
# --  Bring in the .NET stuff, and reference Forms and drawing
import clr
clr.AddReference('System.Windows.Forms')
clr.AddReference('System.Drawing')

#   get the objects you need from forms and drawing
from System.Windows.Forms import Form, Label, Button, TextBox
from System.Drawing import Point, ContentAlignment, Font, FontStyle, GraphicsUnit
#-- The way python works with events, you have to make a class out of your form.
#   Then the objects
#    you add are available in event functions.
class TowerForm(Form):
   def __init__(self):
#-- First, make the form (self)
          self.text="Tower Simulation Tool"
          self.Height=500
          self.Width=250

#-- Create a Simple Text Label describing the tool
#   it is not self.label because we are not going to access it in event functions
          lbl1 = Label()
          lbl1.Text = "This is the Tower of Test Tool. \n\
                Please Enter values and click OK to run the \
                model and get new results\n"
          lbl1.Height = 50
          lbl1.Width = 200

#-- Create the Text Labels and text boxes to get Dimensions and Pressure
#   We need to get to the textboxes so we will make them part of the class
#
#   some constants to make the positioning easier
          x1 = 10
          y1 = 80
          w1 = 70
          w2 = 80
          x2 = x1 + w1 + 10
#   Make the labels for the text boxes.  This is done in the Labe() call with
#   Name = Value arguments
#   instead of object.name = value to save space.  All in one line instead of
#   four lines.
#     The alighment is done so that the labels are all right justified to line
#     up with the boxes. Set ContentAlignment.MiddleRight to mr to save space
          mr = ContentAlignment.MiddleRight
          lb_Length=Label(Text="Length", Width=w1,TextAlign=mr)
          lb_Width  = Label(Text="Width",   Width = w1, TextAlign=mr)
          lb_Height = Label(Text="Height",  Width = w1, TextAlign=mr)
          lb_Press  = Label(Text="Pressure",Width = w1, TextAlign=mr)
```

```
#   Make a label to give status of the update in:
          self.updlbl = Label(Text=" ",Width=200, Height=100,\
                       TextAlign=ContentAlignment.BottomCenter)
          self.updlbl.Location = Point(0,200)
          self.updlbl.Font = \
                Font('Verdana',10, FontStyle.Bold, GraphicsUnit.Point,0)

#   Make the text boxes. Note that these are put in the self. class.
#   We do this so that when the OK
#   button is pushed, we have access to the text boxes and therefore the values
#   typed within

          self.tb_Length = TextBox(Width = w2)
          self.tb_Width  = TextBox(Width = w2)
          self.tb_Height = TextBox(Width = w2)
          self.tb_Press  = TextBox(Width = w2)

#   Specify the location for the label and the text boxes.  Move down by
#   30 after each line
          lb_Length.Location = Point(x1,y1)
          self.tb_Length.Location = Point(x2,y1)
          y1 = y1 + 30
          lb_Width.Location = Point(x1,y1)
          self.tb_Width.Location = Point(x2,y1)
          y1 = y1 + 30
          lb_Height.Location = Point(x1,y1)
          self.tb_Height.Location = Point(x2,y1)
          y1 = y1 + 30
          lb_Press.Location = Point(x1,y1)
          self.tb_Press.Location = Point(x2,y1)

          self.tb_Length.Text = Parameters.GetParameter(Name="P1").Expression
          self.tb_Width.Text = Parameters.GetParameter(Name="P2").Expression
          self.tb_Height.Text = Parameters.GetParameter(Name="P3").Expression
          self.tb_Press.Text = Parameters.GetParameter(Name="P5").Expression

#   Make an OK and a Cancel Button
          okBut = Button(Text="OK", Width=50)
          okBut.Location = Point(30,430)
#       This is where you specify the funtion to be called when the button is
clicked
#       note that you call the function self.functionname.
          okBut.Click += self.okButPressed

          cancelBut = Button(Text="Cancel", Width=50)
          cancelBut.Location = Point(110,430)
          cancelBut.Click += self.cancelButPressed

#   Put everything on the form (some sort of list and loop would make this easier)
          self.Controls.Add(lbl1)
          self.Controls.Add(lb_Length)
          self.Controls.Add(self.tb_Length)
```

```
            self.Controls.Add(lb_Width)
            self.Controls.Add(self.tb_Width)
            self.Controls.Add(lb_Height)
            self.Controls.Add(self.tb_Height)
            self.Controls.Add(lb_Press)
            self.Controls.Add(self.tb_Press)
            self.Controls.Add(self.updlbl)
            self.Controls.Add(okBut)
            self.Controls.Add(cancelBut)

#-- Now define the button event functions as part of the class
#
# Cancel simply closes the form down.  Nothing fancy
    def cancelButPressed(self, sender, args ):
            print 'Closing the Window... Bye...\n'
            self.Close()

# OK prints the values of the text boxes to the console
#   This will be replaced with calls to workbench for the next step
    def okButPressed(self, sender, args ):
            print 'OK Pressed, Closing the Window... Bye...\n'
            self.Close()

#---------End of class definition
```

15. Give this code a try. It is not hooked up to anything but make sure it works before you move on.
16. Now we want to connect this dialog box with Workbench. Specifically, we want to play with parameters. For this we will use Parameters.GetParameter() to get the container for each parameter then we will use the Expression method for each one to set its value where needed, and the Value method to get the values of the output param we want (deflection).
    To do this we will change the okPutPressed() function. Also, we need to do an Update() on our workbench systems to make sure the new parameters get used, then grab that deflection parameter.

```
def okButPressed(self, sender, args ):
# In Workbench: Get objects for all the parameters you need access to
#   Remember, it goes by parameter name, not the name you give them.
            lenParam = Parameters.GetParameter(Name="P1")
            widParam = Parameters.GetParameter(Name="P2")
            hgtParam = Parameters.GetParameter(Name="P3")
            prsParam = Parameters.GetParameter(Name="P5")
# don't forget the resulting deflection output parameter:
            defParam = Parameters.GetParameter(Name="P4")

# Now set the workbench values to the values from the dialog
            lenParam.Expression = self.tb_Length.Text
            widParam.Expression = self.tb_Width.Text
            hgtParam.Expression = self.tb_Height.Text
            prsParam.Expression = self.tb_Press.Text
```

```
            self.updlbl.Text = "Updating"
# New Parameters are in, update model with those
            Update()
# Get the new deflection, post update
            dflValue = defParam.Value.Value
# Set the label in the dialog to show the deflection value
            self.updlbl.Text = "Deflection is %s in\n" %dflValue
```

17. And that is it. The basics of this script can be modified to do so much more.