

IM227166-L

iLogic from Zero to 60 in 90

Jon Balgley
Autodesk

IM227166-L

iLogic from Zero to 60 in 90

Jon Balgley
Autodesk

Learning Objectives

- Create an iLogic rule that validates the 'range' of the value of a parameter.
- Create an iLogic rule that updates an iProperty whenever the file is saved.
- Create an iLogic "form" that makes it easy to change certain Inventor parameter or iProperty values.
- Explain the difference between internal and external iLogic rules

Description

Have you heard great things about iLogic but never really learned about how or when to use it? This hands-on lab will cover key concepts and techniques, with minimal expectations of programming experience. You'll leave with an understanding how you can be more efficient with Inventor by using iLogic to automate routine tasks.

We will begin with the basics: what rules are and when do rules fire? Samples and exercises will illustrate the extents of what can you do with rules – in parts, assemblies and drawings. We'll work with iLogic "forms" which let you make a "user interface" without programming. We'll use rules in different circumstances/scenarios – for parameter validation, for updating iProperties on file open/save.

Finally, you'll get some exposure to more advanced capabilities like external rules and using the Inventor API.

Speaker Bio



Jon has worked at Autodesk since 2005, and has designed, developed and taught CAD automation technologies since the 1980's. Recent work includes Design Automation API for Inventor, Configurator 360, and iLogic.

jon.balgley@autodesk.com

[Inventor Customization Forum](#) -- your best bet for answers

Intro & Overview.

In this section, we'll create a simple iLogic rule. It will validate that the value of parameter is within a given range, and then take various actions if it is outside the range.

Let's start with some basics:

What is iLogic?

iLogic is a built-in capability of Inventor that allows you to easily run bits of logic (program code) at certain predefined & convenient times.

The “bits of logic” are called **rules**. You get to write the rules. Simple logic is easy to write. More complex logic is harder to write. You can do a lot with simple logic!

“Running” a rule is also called “triggering”. The most common way to run a rule is:

➔ Refer to a parameter in the logic in the rule

When you change the value of that parameter, iLogic automatically triggers (runs) your rule.

There are other ways to trigger/run rules, too.

What can you do with iLogic?

Just about anything you can do, “by hand”, with Inventor!

But in practice, most people focus on simple “routine” operations, such as:

- Checking for obvious errors (e.g., out-of-range parameters)
- Automatically setting iProperties based on various conditions
- Filling in title blocks, setting the view-scale, etc.

iLogic rules can be used in parts, assemblies and drawings.

iLogic rules can *analyze* the content of a document.

iLogic rules can *modify* the content of a document.

iLogic rules can *synthesize* new content for a document.

In the following exercises:

Yellow highlight means some action that you have to do.

Green highlight indicates something that is optional. OK to skip if you're short on time.

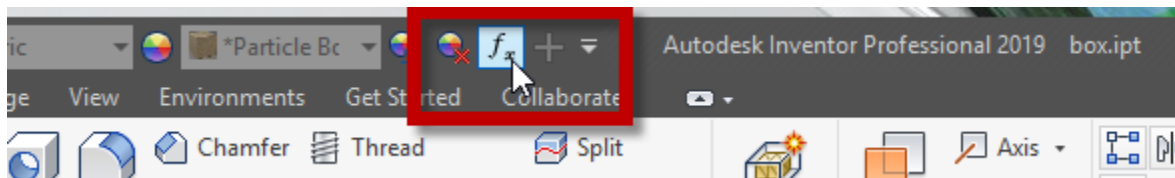
1. Exercise: Rule to enforce the “range” of parameters.

Step-by-step:

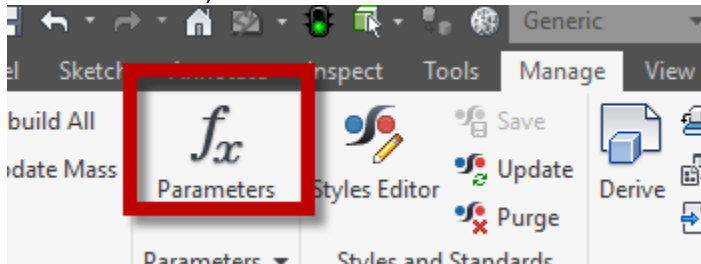
1. Start Inventor 2019.
2. Select project, ex1.1.clean
3. In ex1.1.clean, open box.ipt.

4. Open Parameters dialog:

Either use the shortcut on the title bar...

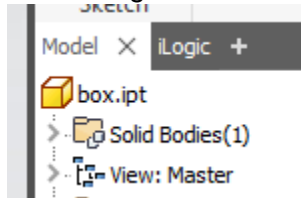


..Or the regular command on the ribbon... (“Manage” ribbon for IPTs ... “Assemble” ribbon for IAMs)

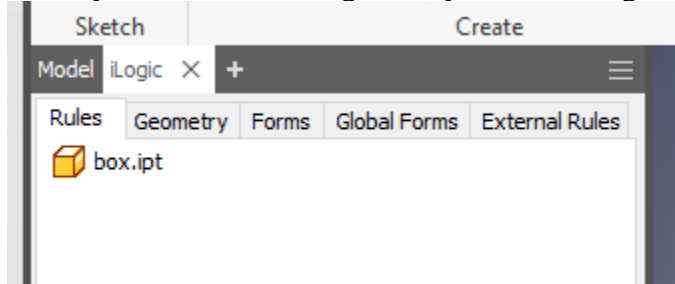


5. Observe that the box size is controlled by the four parameters. Change ‘width’ from its initial value to something really small, like “3 in”. While it “works”, it’s probably not acceptable to manufacture/sell. Now change it to something REALLY small, like 0.5 (less than the thickness). Observe that the model breaks. Set ‘width’ back into appropriate range (48). Close Parameters dialog.
6. Idea: Limit ‘width’ value to be in the range of 8” – 24”. If outside the range, clamp it to the min/max.

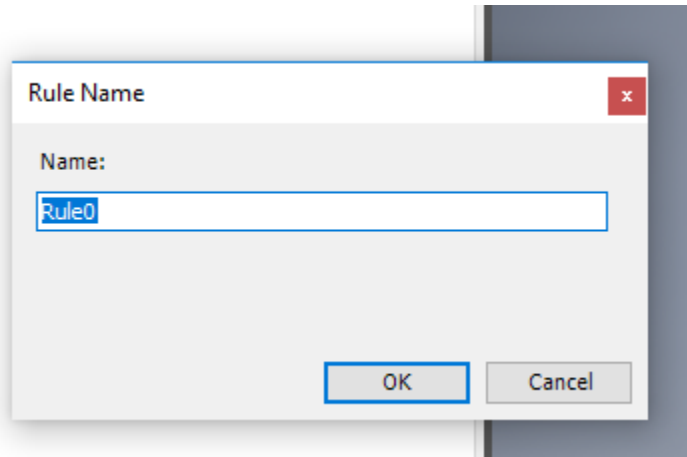
7. **Open iLogic browser** using the tab next to “Model”. If “iLogic” is not showing, use the “+” button to get it:



When you click on the iLogic tab, you’ll see a big blank panel like this:

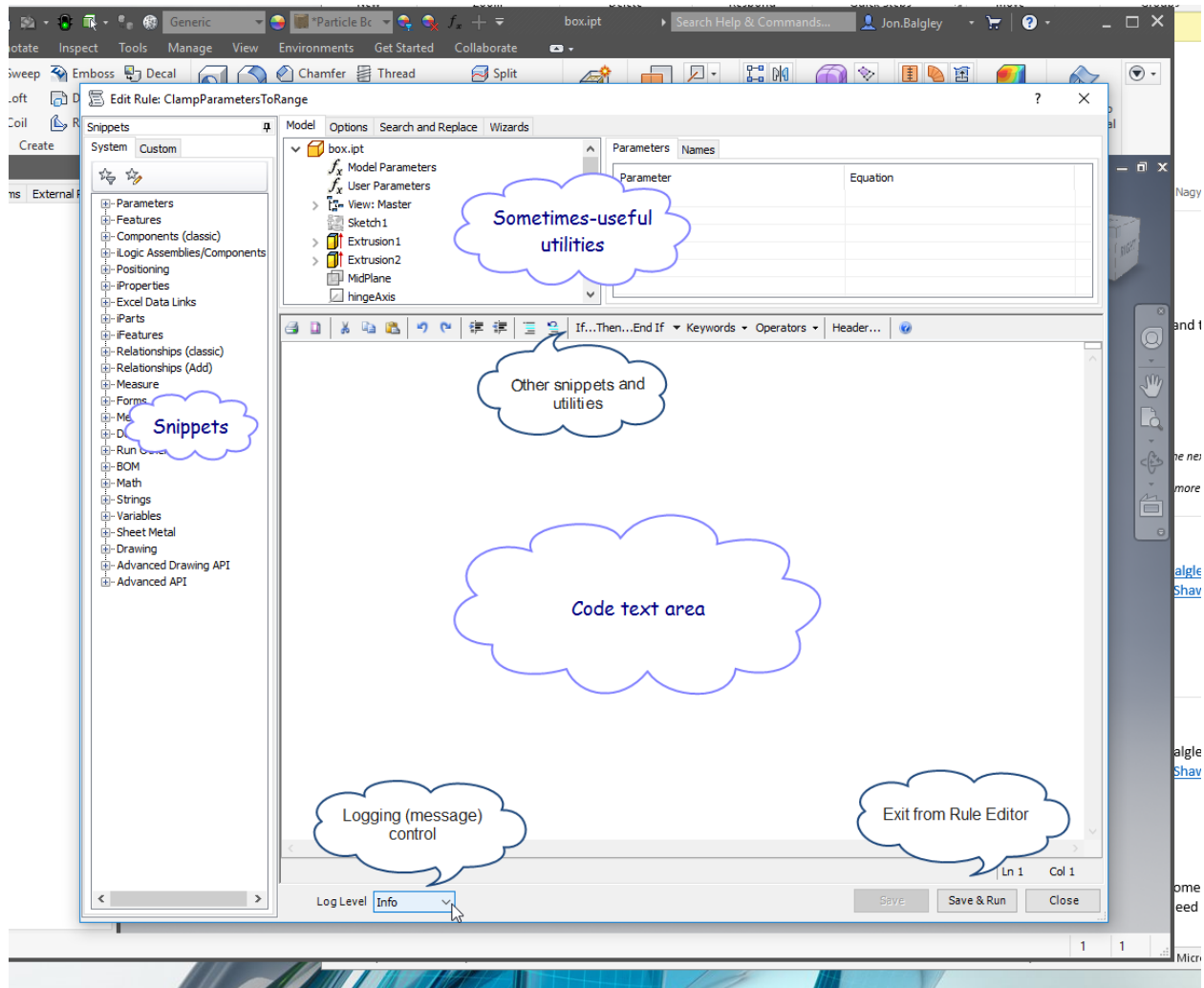


8. Click right on the blank area (with the “Rules” tab selected). Select “Add Rule”.

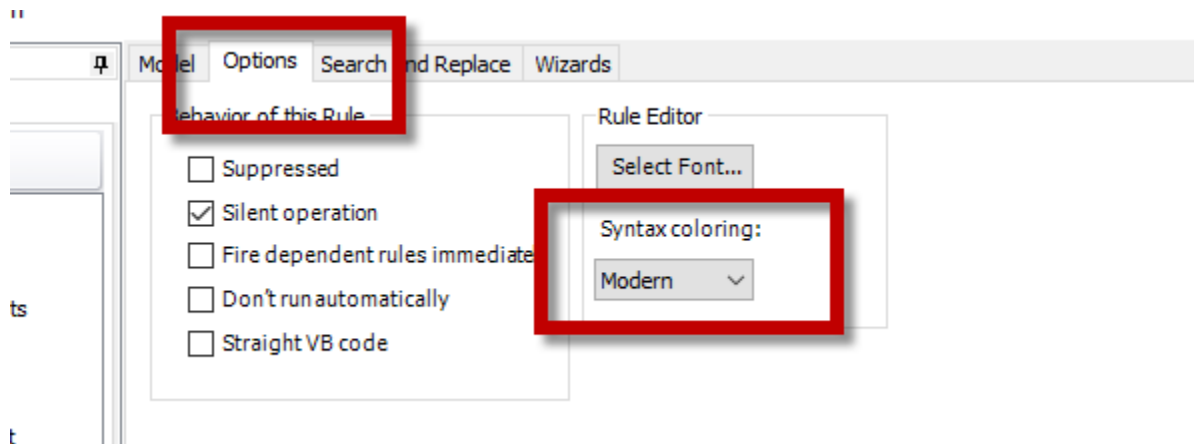


Change the rule name to “ClampParametersToRange” (You weren’t going to name your rules “Rule0”, “Rule1”, were you???)

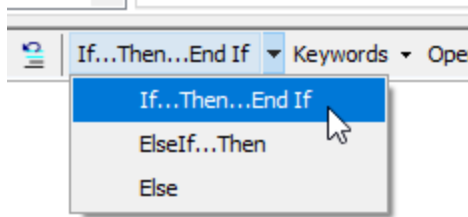
9. The Rule Editor pops up. Take some time to become familiar with this window.



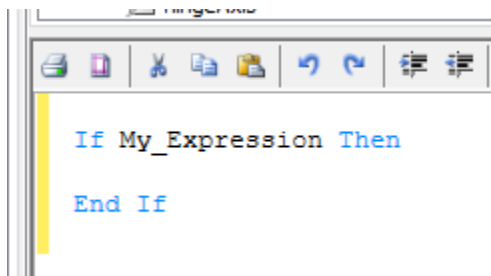
10. In the Options tab, change the Syntax coloring from “Classic” to “Modern” (it’s OK to keep “Classic” colors, but colors won’t match with this document).



11. Click on “If...Then...End If”, in the middle:



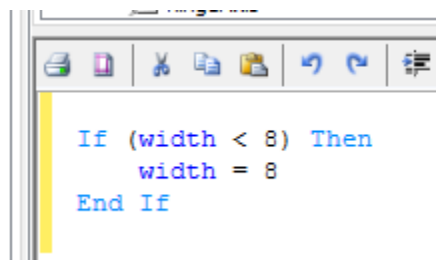
iLogic inserts code like the following:



12. Modify the template code (by typing on the keyboard).

You may notice the iLogic tries to help you by giving you a menu of possible completions. Feel free to use that.

- a. Change “My_Expression” to “(width < 8)”
- b. Insert “width = 8” on the blank line after the “Then”

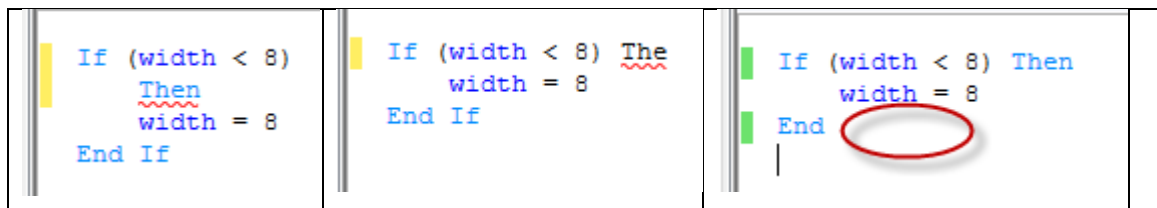


Notice that “width” appears in a darker blue. This means it is correctly referring to a parameter. If it’s not dark blue like that, you’ve probably made a spelling error. Make sure you are using the correct “case” (Inventor is case-sensitive ... this parameter name has no capitals).

13. Click on “Save & Run”. Nothing happens ... apparently. Did it run? Probably yes; but we can’t tell. (Note that “Save” here means that the text of the rule is now included in the IPT *in memory only*. You still have to save the file to make it permanent.)

14. Open Parameters dialog again, and change width to 3 ... only goes to 8, not 3. Did it run? Yes. What made it run? Triggered by change to 'width' parameter. Normally iLogic rules are triggered (executed) if any referenced (i.e., in dark blue) parameter is changed. Close the Parameters dialog.
15. Now is a good time to save the file. The rule is an integral part of the IPT file, just like sketches, features, workfeatures, etc.
16. Now let's see what happens if you make certain mistakes. You can open the rule for editing by clicking on it in the iLogic panel. You can right-click and select "Edit Rule", or double-click on it.

Make each of the indicated errors below, and click "Save & Run" to see what happens.



After you've tried these, put rule back to its working version.

17. Well, silently clamping a parameter to value that's different from what the user requested can be inappropriate. The user might not notice that you've clamped it. How can we make the "clamping" more obvious? Here are three options:

When clamping is required, use one of the following statements: (free-form exercise, no specific steps, you have to)

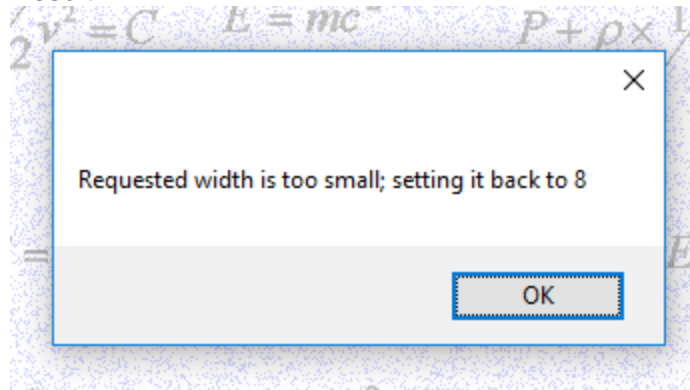
- a. Copy-and-paste:

```
MessageBox.Show("Requested width is too small; setting it to 8")
```

Image of desired rule:

```
If (width < 8) Then
    MessageBox.Show("Requested width is too small; setting it back to 8")
    width = 8
End if
```

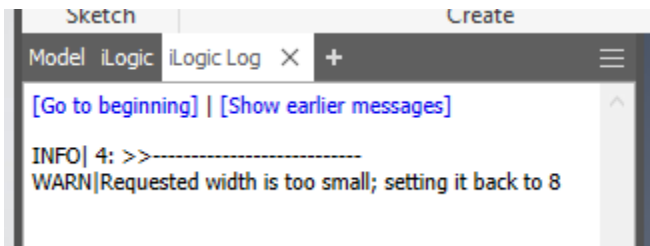
Result:



Using MessageBox.Show is a good idea when you want to be intrusive.

- b. `Logger.Warn`("Requested width is too small; setting it to 8") `Skip this unless you're interested.`

```
If (width < 8) Then
    Logger.Warn("Requested width is too small; setting it back to 8")
    width = 8
End if
```



The message appears in the “iLogic Log” tab. It doesn’t show automatically; you have to add it with the “+”.

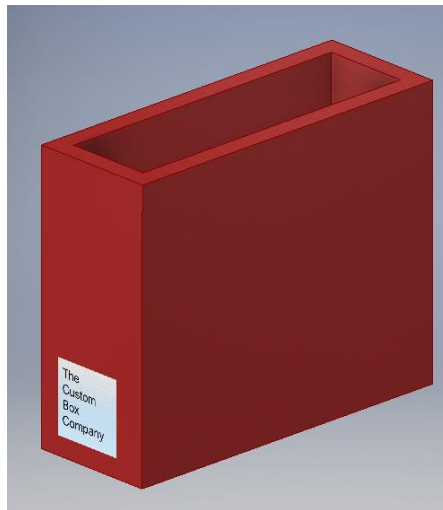
Use `Logger.Warn` when you want to be subtle. User might not be looking for these warnings. More about “Logger” later...

Also note: “Logger” is a new capability in Inventor 2019.1. Your experienced colleagues may not be aware of it.

- a. Change color or other aspect of model. Skip this unless you're interested.

```
If (width < 8) Then
    iProperties.PartColor = "Red"
    width = 8
Else
    iProperties.PartColor = "Particle Board"
End If
```

(You might notice that iLogic helps you when you press the “.” after “iProperties”.)



Use this when you want to be intrusive, but you don't want to block processing until the user responds. Probably some additional notification to the user may be necessary. Consider a special “warning decal”.

18. **Optional – it's tricky.** The previous errors were fairly obvious. Let's examine a more subtle one.

- a. **Open the Parameters dialog** and **set the width to be 3**. Your rule fires and clamps the width to be 8.
- b. **Edit the rule** to have the following mistake:

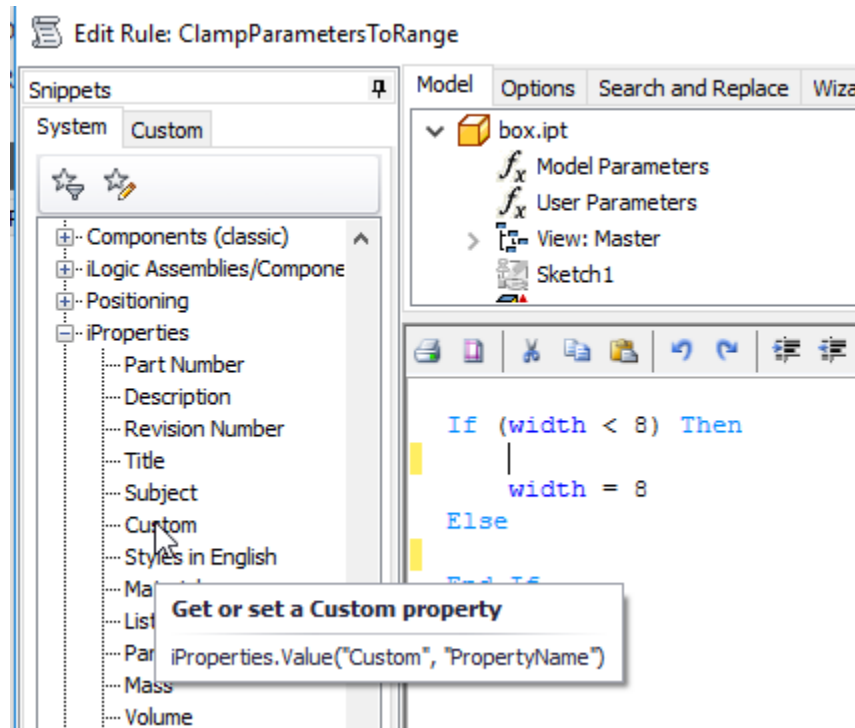
```
If (widthxxxxx < 8) Then  
    width = 8  
End if
```

- c. **Click "Save & Run"**.
- d. Nothing obvious happens to the model.
- e. **Change the width to be 3**. Your rule fires, and the width is clamped to 8. So it still works right?
- f. **Now change the width to be 36**. Your rule fires, and ... the width is clamped to 8!!! What happened? Well, iLogic is happy to allow you to have program variables that are not pre-declared, such as "widthxxxxx". And if you don't set their value, the value is automatically set to zero. So zero is ALWAYS less than 8. So it always clamps width to 8. Ugh! Tricky! This is probably one of the worst kind of subtle errors, and now you know about it.

If necessary, put the rule back to the working version, and re-save the file.

19. **Optional** Use some basic iLogic snippets. For example, to add or modify a custom iProperty:

- Put the cursor where you want the snippet text to be inserted.
- Double-click on the snippet (iProperties → Custom)



c. The snippet text is inserted:

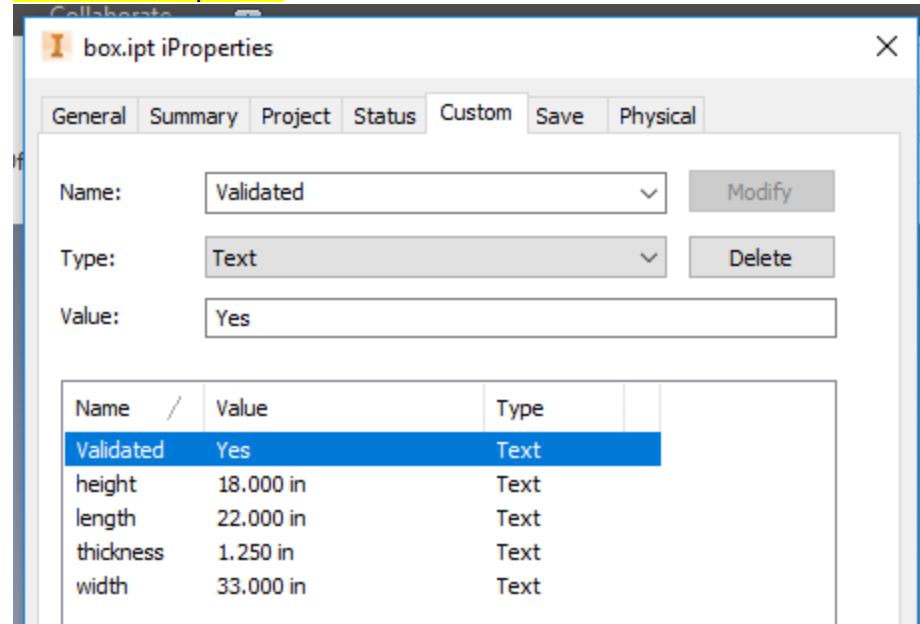
```
If (width < 8) Then
    iProperties.Value("Custom", "PropertyName")
    width = 8
Else
End If
```

d. Change the snippet text as needed:

```
If (width < 8) Then
    iProperties.Value("Custom", "Validated") = "No"
    width = 8
Else
    iProperties.Value("Custom", "Validated") = "Yes"
End If
```

e. Click "Save & Run" to test.

- f. Show the iProperties and look at the Custom tab:



20. Additional exercises:

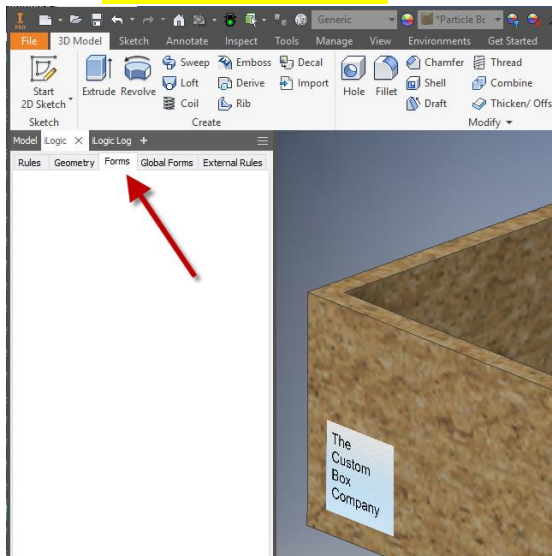
- Make this work for maximum values too.
- Make this work for length and height too.
- See how much you can avoid duplication in your code as you do that.
- Add additional features (decal or emboss or extra “body”) that shows “warning” text in the bad state.

2. iLogic Forms

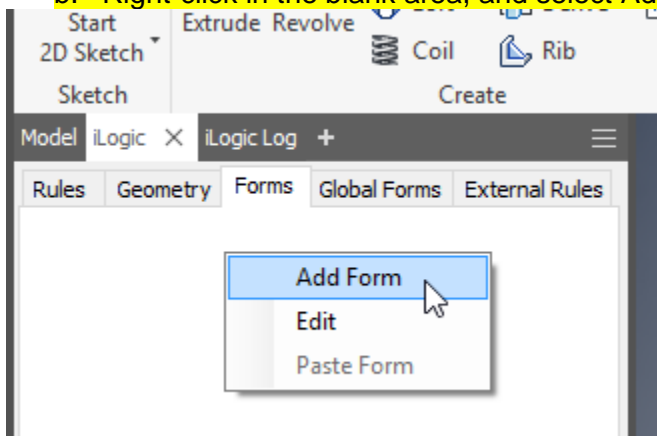
- Nice way to interact with (customize, configure) a model
- The “Forms” tab
- Parameters, properties, rules.
- Plus pictures, for show!
- Multi-value parameters
- Also stored in file, usually; implies file-copying

Exercise 2.1: Make a simple form

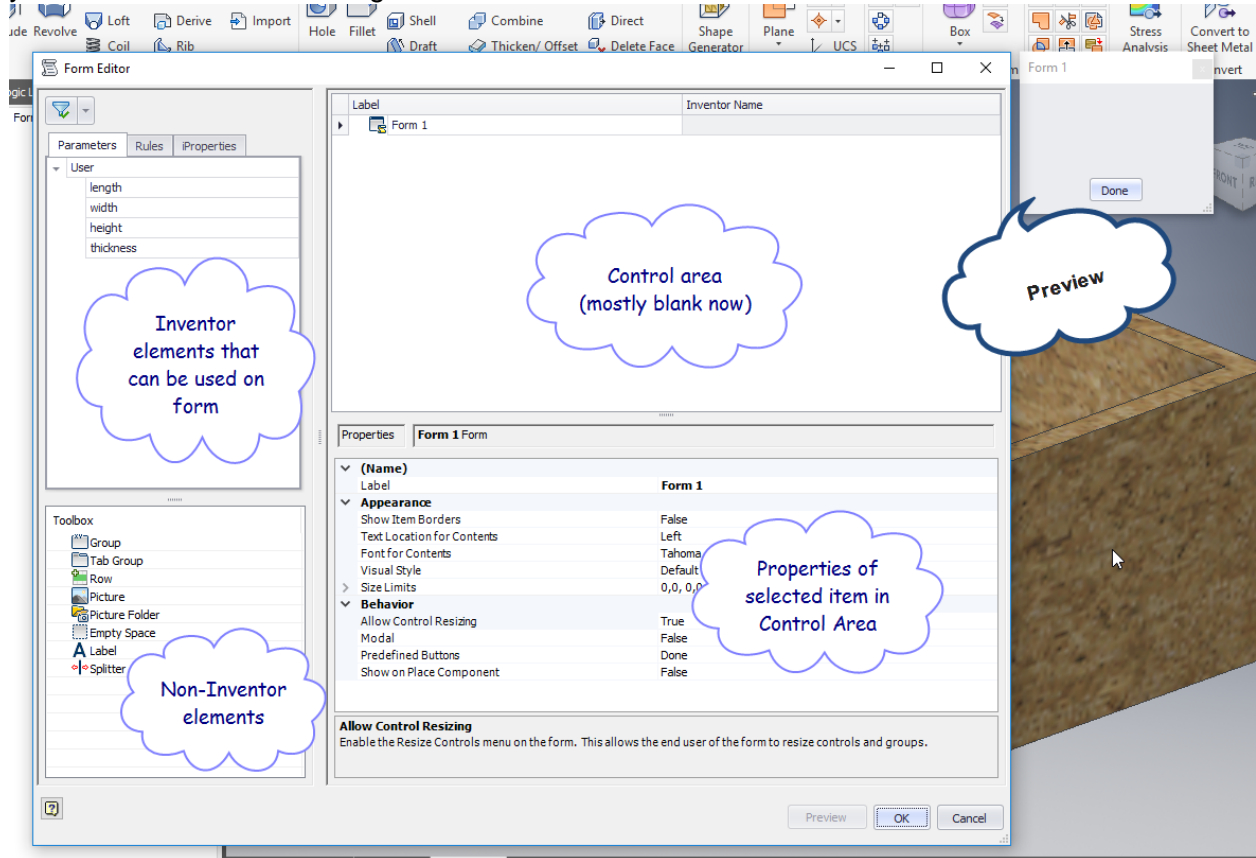
- Parameters
 - Multi-value parameter
 - True/False parameter
1. Restart Inventor, select project ex2.1.clean, and open the box.ipt from that folder.
 2. On iLogic browser panel,
 - a. select the “Forms” tab:



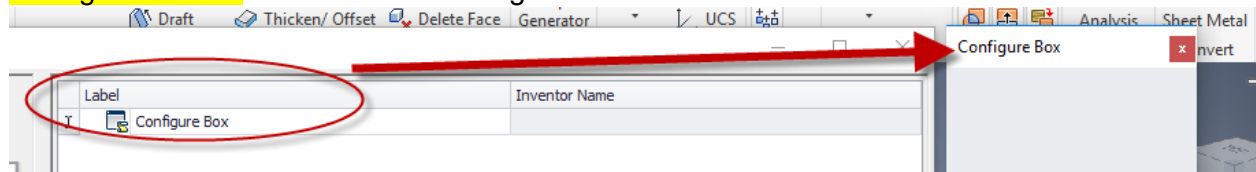
- b. Right-click in the blank area, and select Add Form:



- The Form Editor pops up, for the new form. Take a few minutes seconds to familiarize yourself with the different regions:

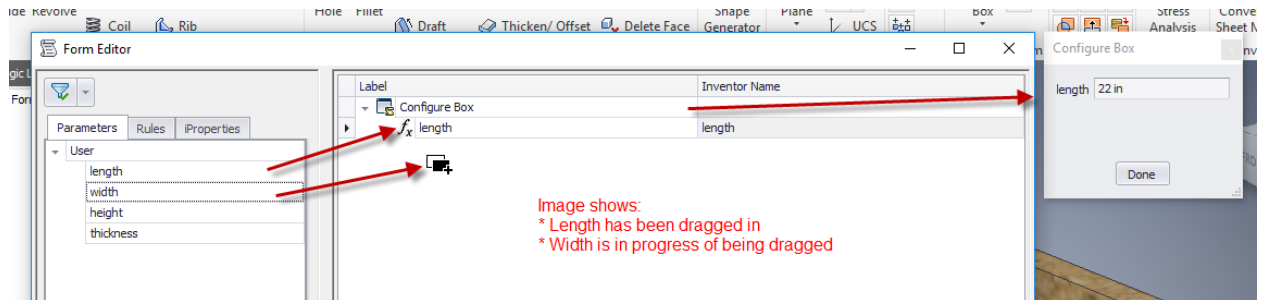


- Change the name of the form to "Configure Box"

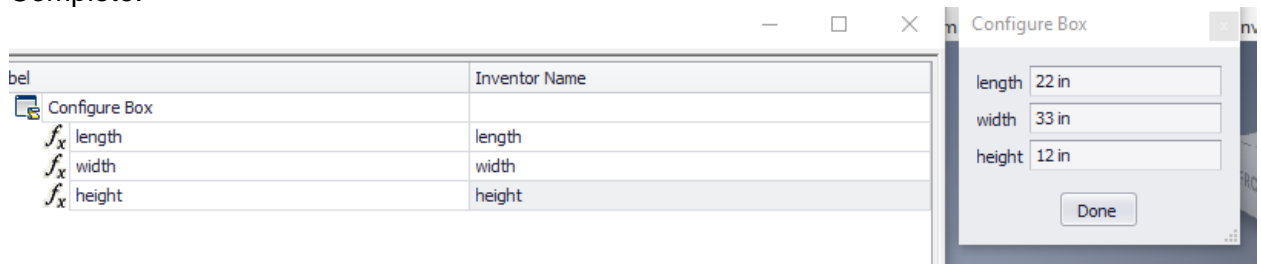


5. Add length, width and height, by dragging from Parameters tab to Control Area:

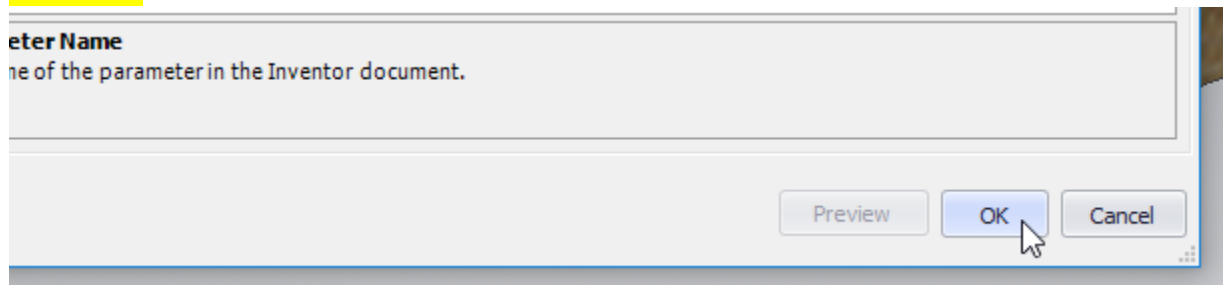
In progress:



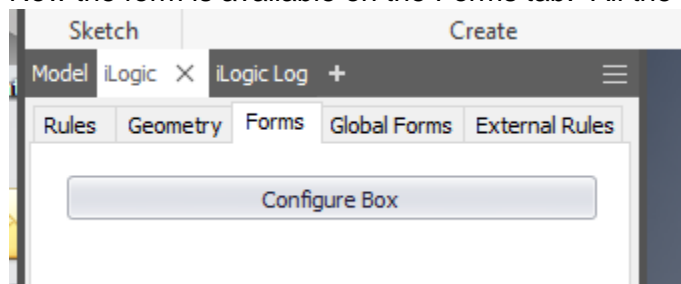
Complete:



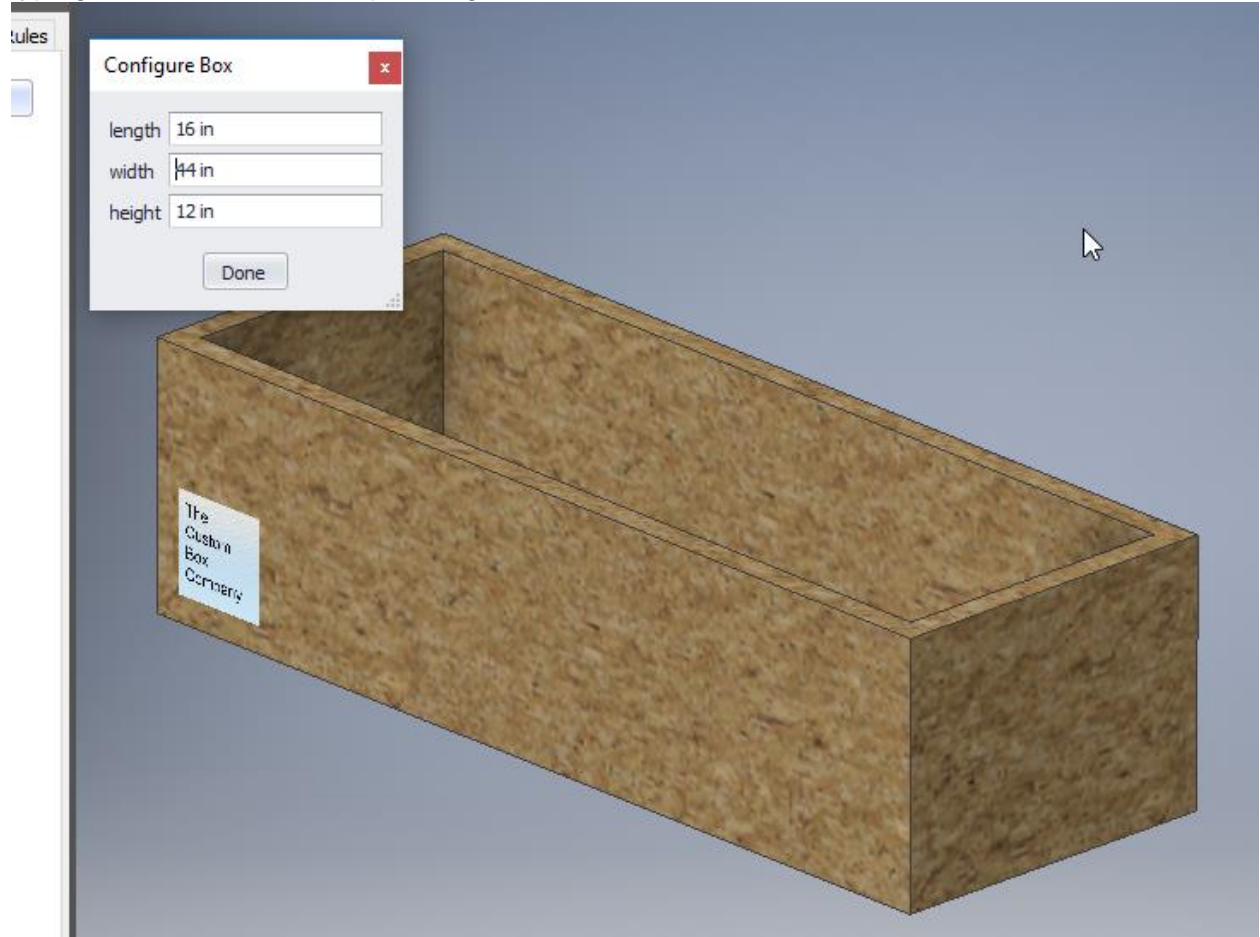
6. Click "OK" to create the form.



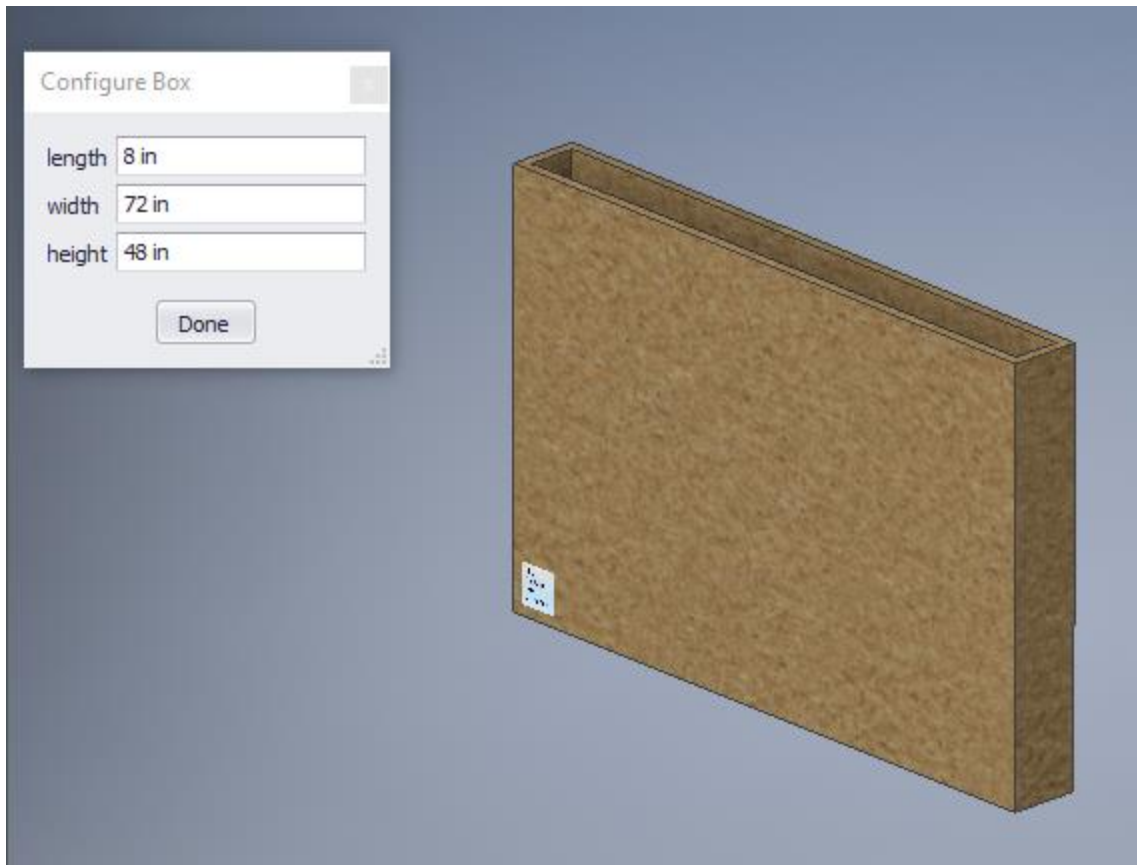
7. Now the form is available on the Forms tab. All the forms in this document will appear here.



8. Click the "Configure Box" button. The form appears. Change the parameter values by typing in new numbers and pressing <enter> after each one.



9. Note that your rule(s) will fire when you change parameter values. See what happens if you try to **change the parameters to a value outside of the valid range**:



Note also that you don't have to have any rules at all, just to use a form.

Click 'Done' on the form when you've tried enough variations.

10. **Save the file.** The form is saved within the file.

11. Let's say the 'thickness' really shouldn't be completely variable, but should be chosen from a handful of standard dimensions:

0.0625

0.125

0.25

0.5

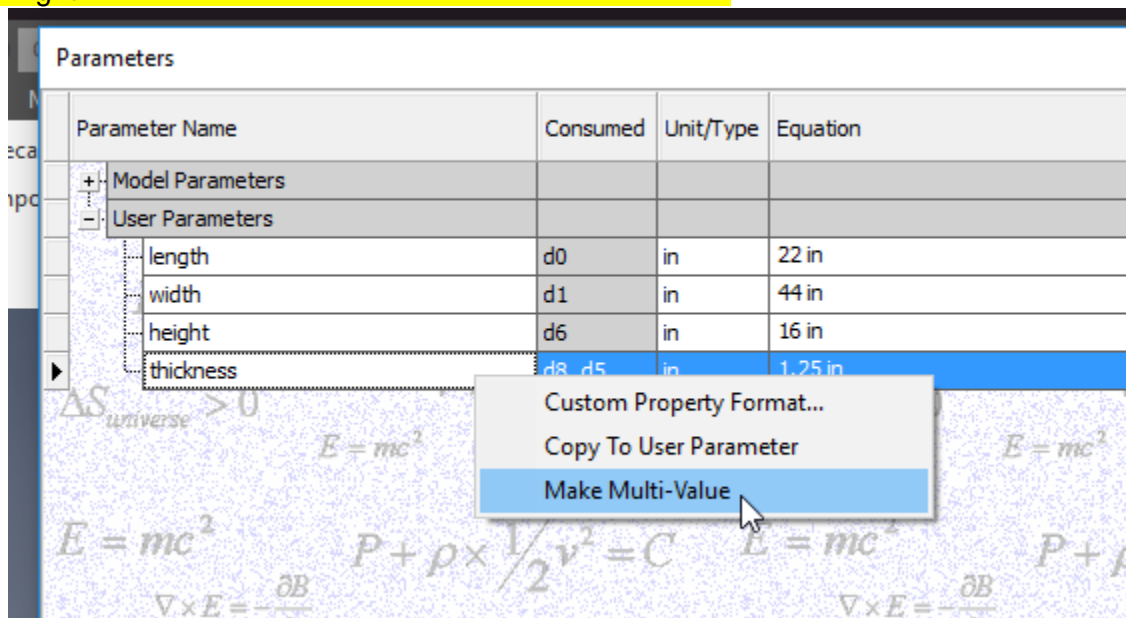
1.0

1.5

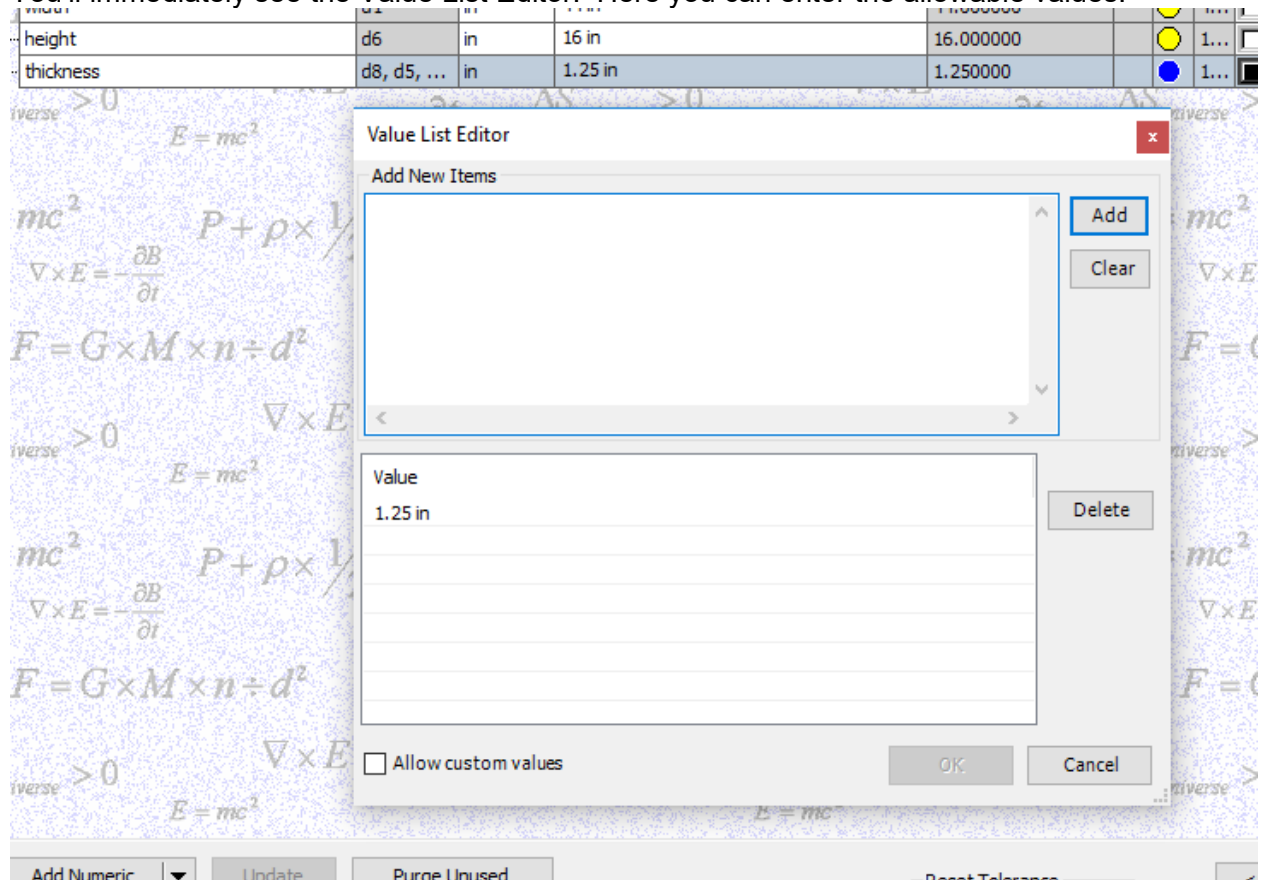
The best way to do that is with Inventor's "multi-value" parameter mechanism.

A. Open the Parameters dialog

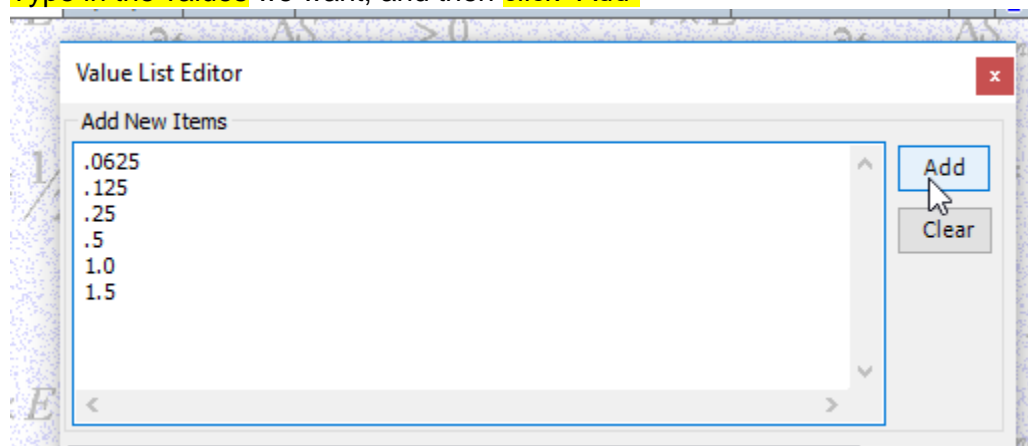
B. Right-click on "thickness" and select "Make Multi-Value"



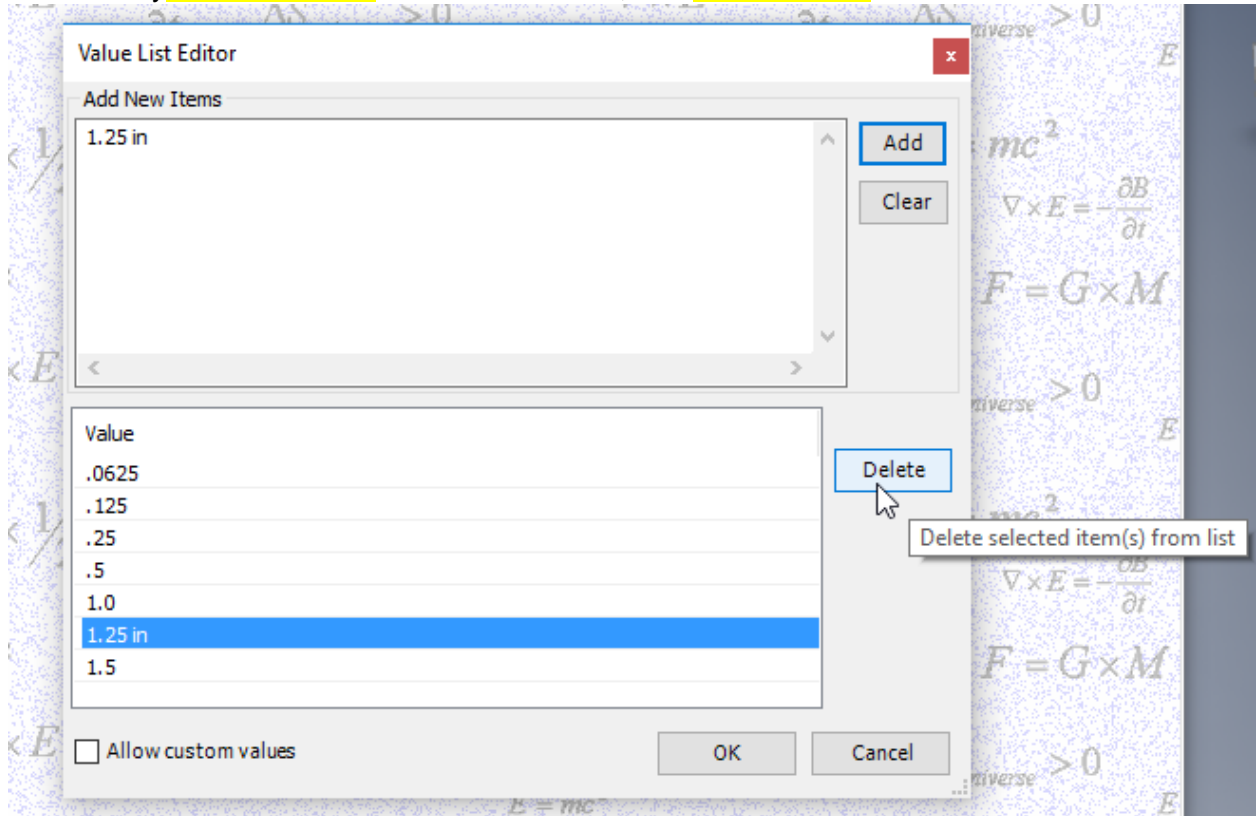
You'll immediately see the Value List Editor. Here you can enter the allowable values:



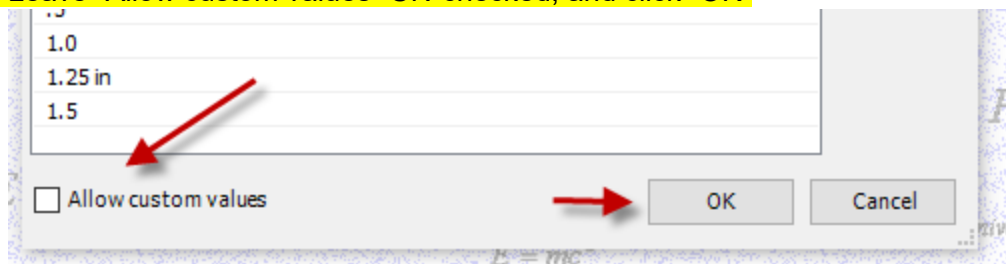
C. Type in the values we want, and then click "Add"



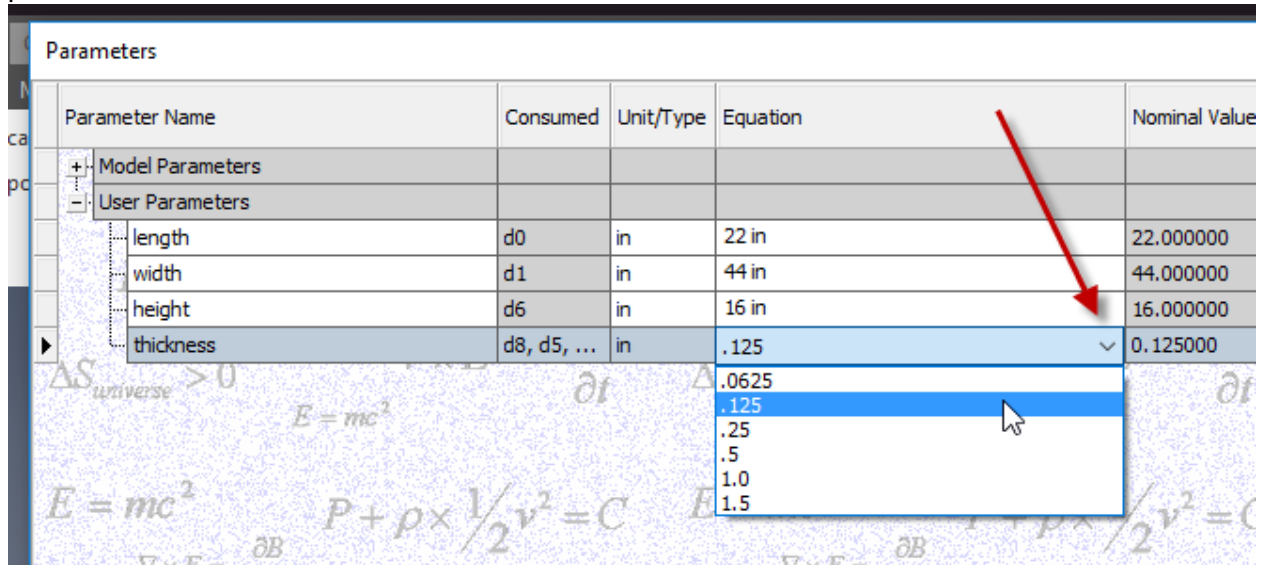
D. If necessary, select “1.25 in” in the bottom area, and click “Delete”:



E. Leave “Allow custom values” UN-checked, and click “OK”



- F. Now you can see that 'thickness' has a drop-down box to enter values, unlike the other parameters.



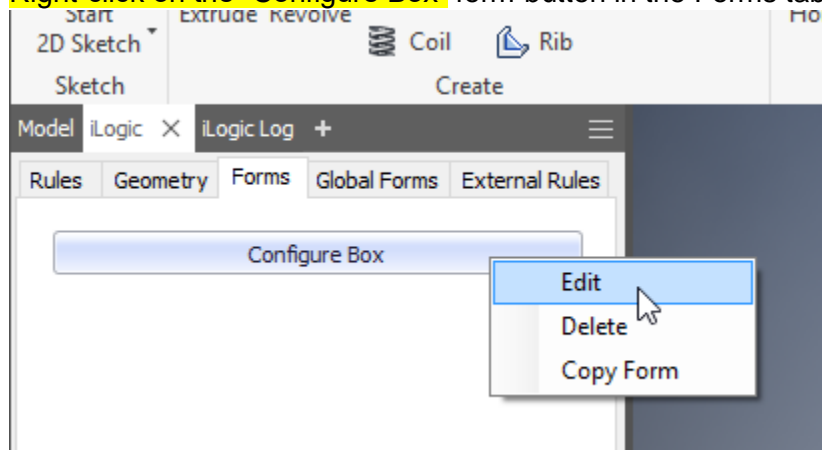
Parameter Name	Consumed	Unit/Type	Equation	Nominal Value
Model Parameters				
User Parameters				
length	d0	in	22 in	22.000000
width	d1	in	44 in	44.000000
height	d6	in	16 in	16.000000
thickness	d8, d5, ...	in	.125	0.125000

Multi-value parameters are a normal aspect of Inventor; you don't have to use iLogic to have a multi-value parameter.

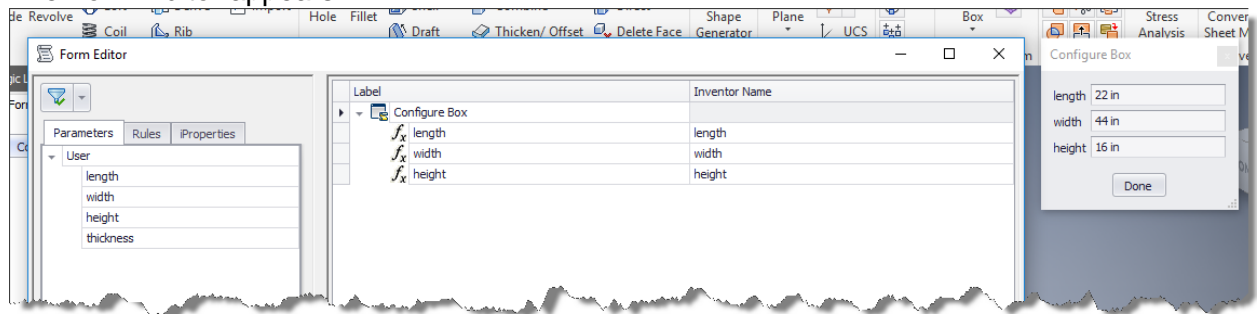
- G. **Close** the Parameters dialog.

12. Place 'thickness' onto the form, as follows:

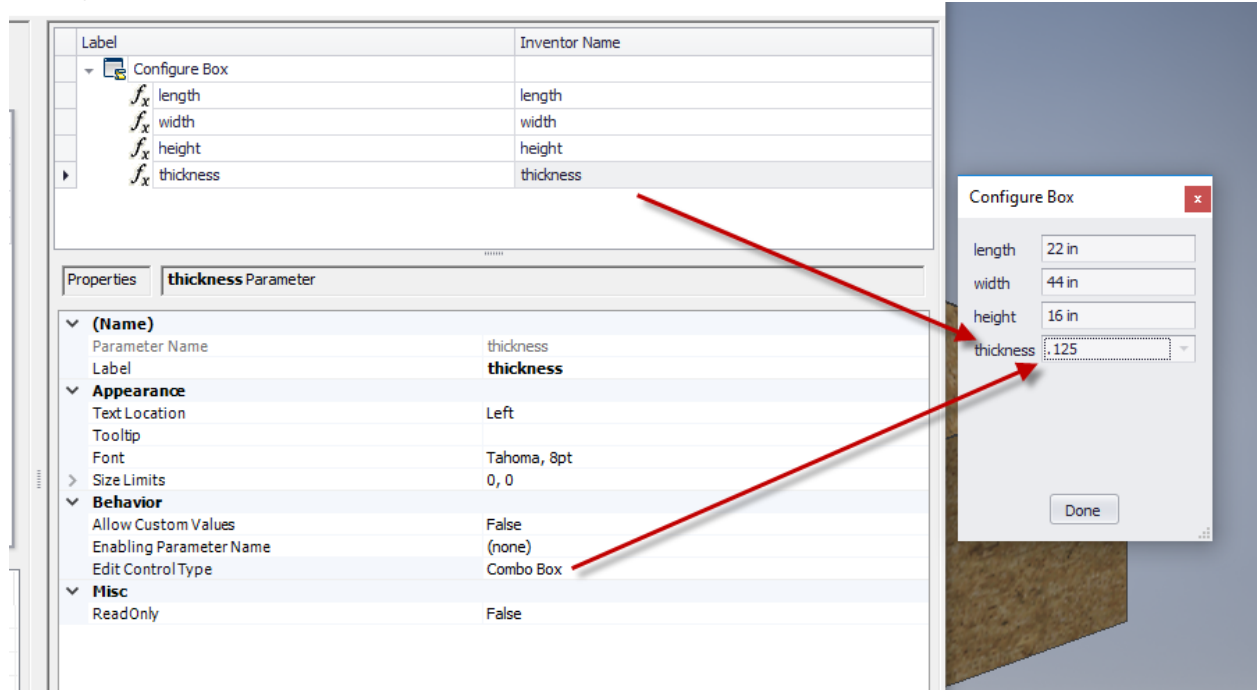
- a. Right-click on the "Configure Box" form-button in the Forms tab, and select Edit:



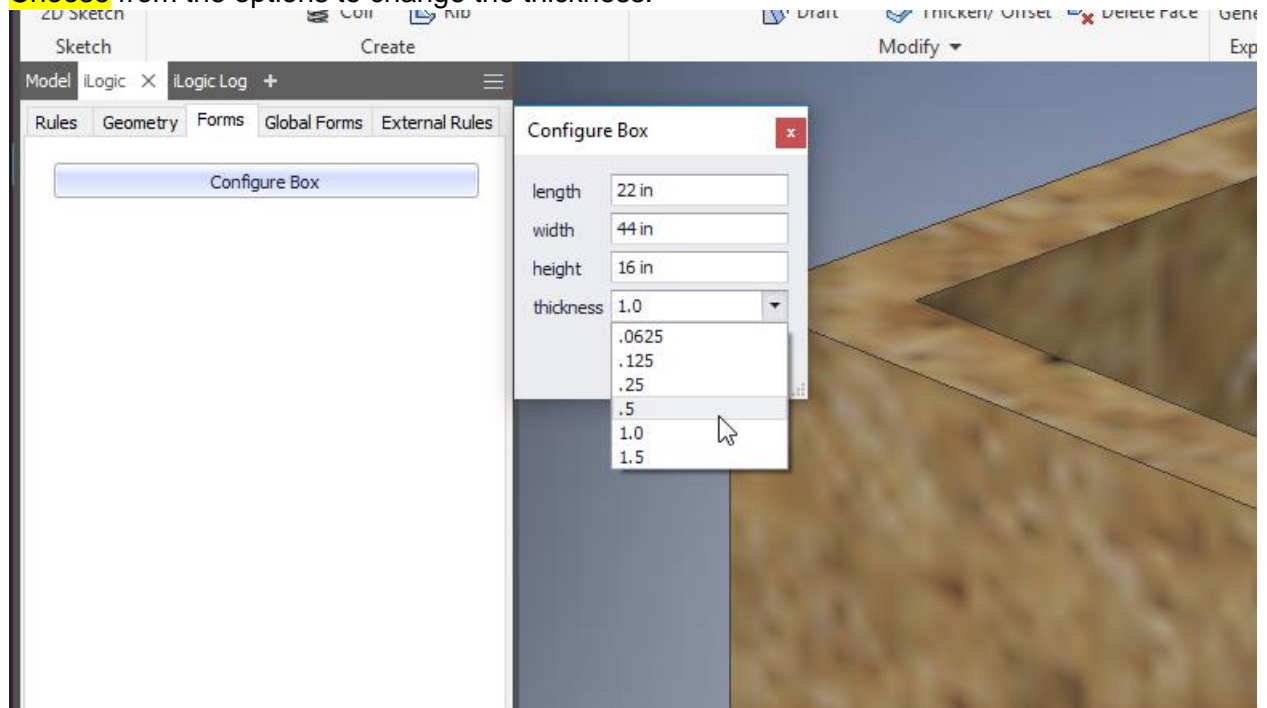
- b. The Form Editor appears:



- c. Drag 'thickness' into the Control Area. By default, it gets a "Combo Box" (drop-down menu) control:



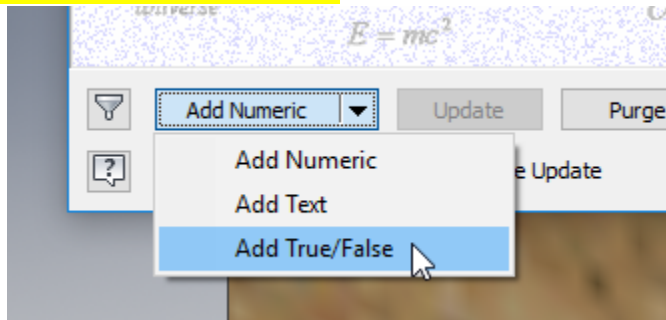
- d. Click on the “Configure Box” button in the Forms tab. The form opens, now with “thickness” listed.
- e. Choose from the options to change the thickness:



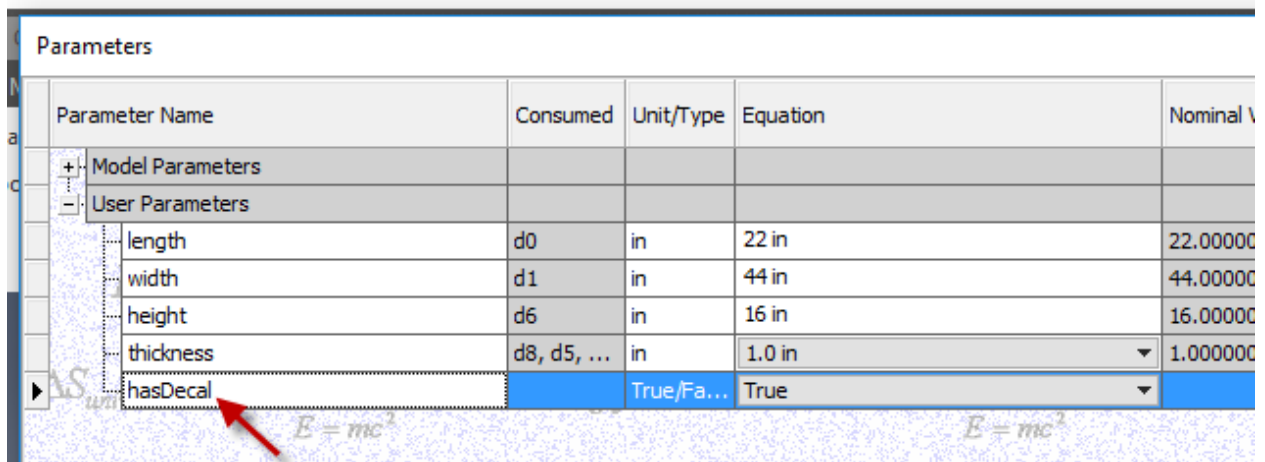
13. Save the file.

14. Now imagine that the “logo” decal is an option. Sometimes we want to turn it off, so that should also be part of the “configuration”. The best way to do that is with a “True/False” (Boolean) parameter.

- a. Open the parameters dialog
- b. Click on “Add True/False”



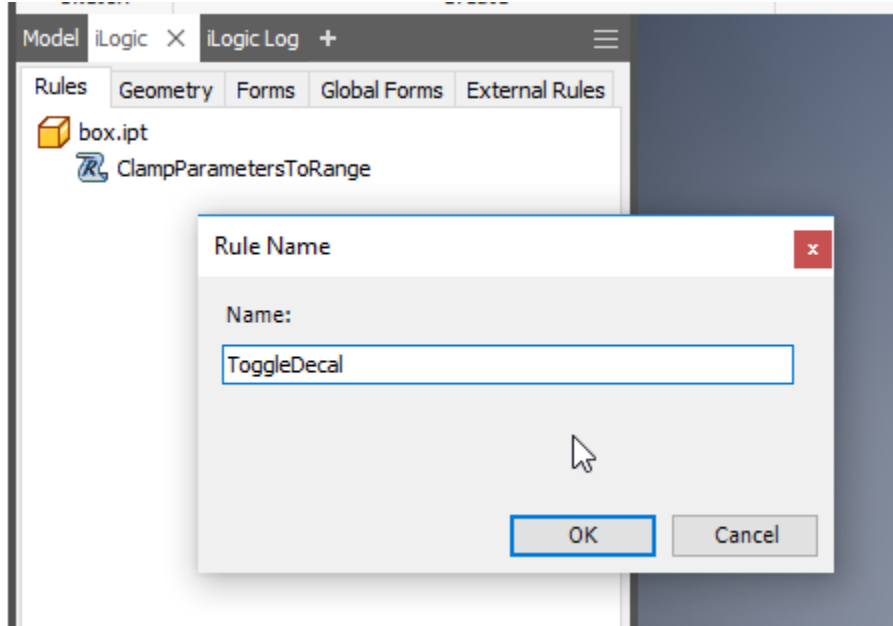
- c. Give it a “Boolean-sounding” name, like “hasDecal”:


 A screenshot of the 'Parameters' dialog box showing a table of parameters. The table has five columns: 'Parameter Name', 'Consumed', 'Unit/Type', 'Equation', and 'Nominal Value'. The 'hasDecal' parameter is highlighted with a red arrow.

Parameter Name	Consumed	Unit/Type	Equation	Nominal Value
Model Parameters				
User Parameters				
length	d0	in	22 in	22.00000
width	d1	in	44 in	44.00000
height	d6	in	16 in	16.00000
thickness	d8, d5, ...	in	1.0 in	1.000000
hasDecal		True/Fa...	True	

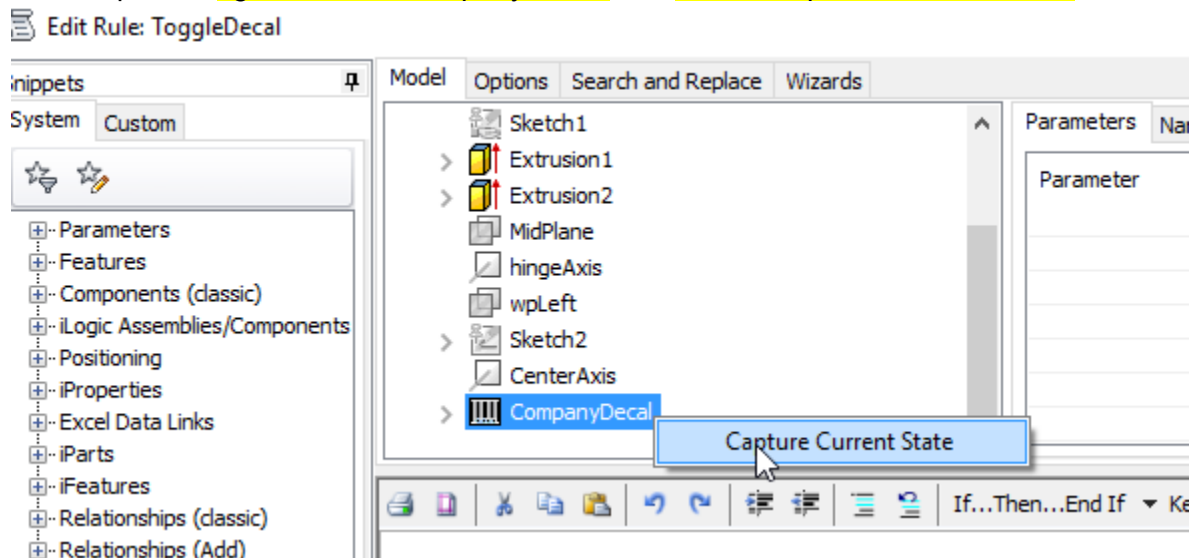
- d. Click “Done” to close the Parameters dialog.
- e. Observation: Switch back to the Model tab, and right-click on the “CompanyDecal” feature. There’s a “Suppress Features” command. If it’s suppressed, then the decal disappears. We want to control that automatically from the new Boolean parameter, so:

- f. Add a rule: If necessary, switch back to the iLogic tab. Select the “Rules” tab. Right-click in blank area, and select “Add Rule”. Give it a nice name, like “ToggleDecal”.

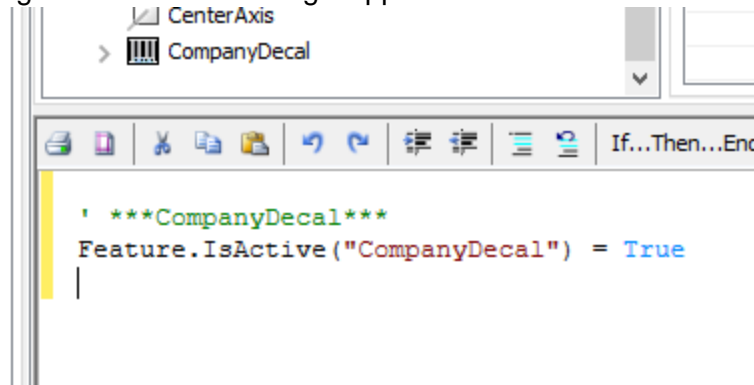


Click “OK” and the Rule Editor appears.

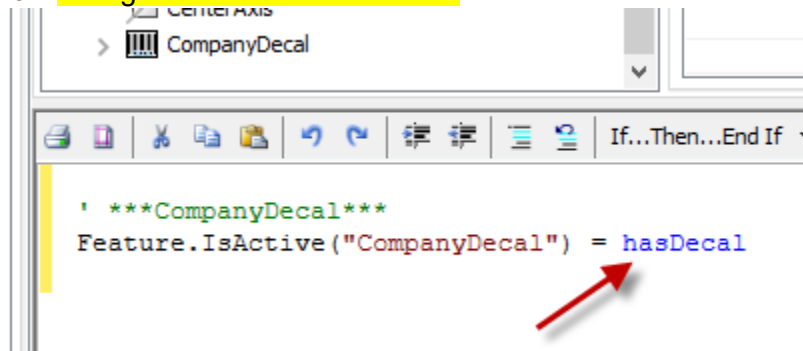
- g. In the top area, right-click on “CompanyDecal” and select “Capture Current State”:



- h. iLogic inserts the following snippet:



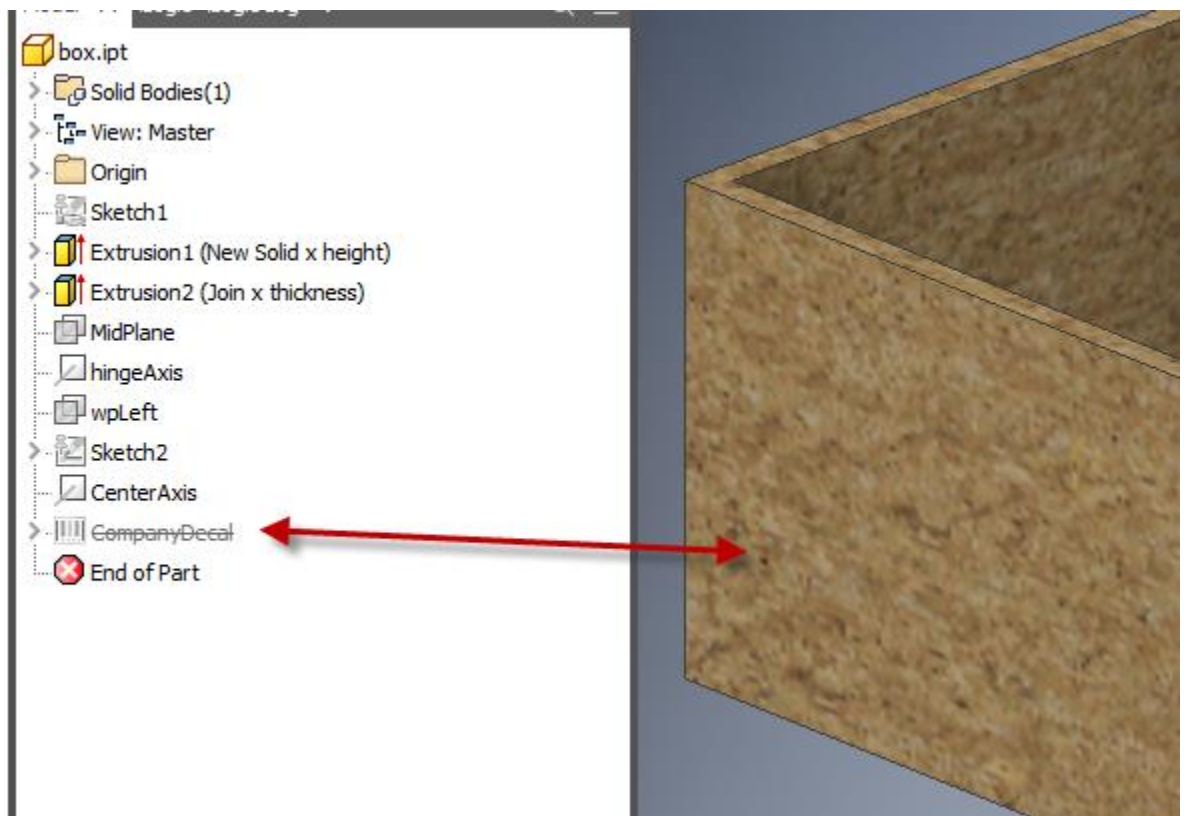
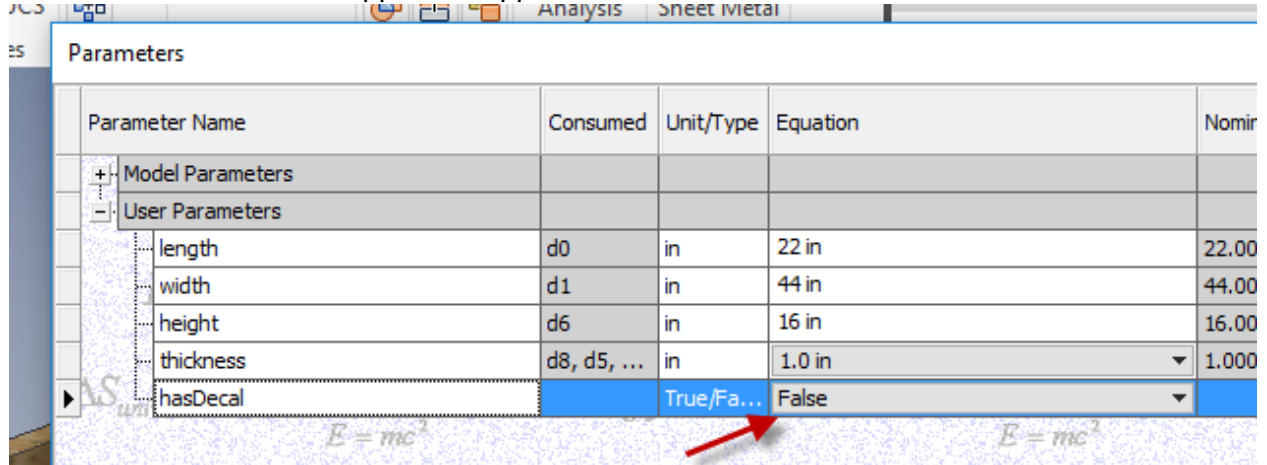
- i. Now change the 'True' to 'hasDecal':



This rule will now be triggered whenever 'hasDecal' changes value. Since the only value it can have is "True" or "False", either one is valid here ... and no "if statement" is required!

- j. Click "Save & Run". The decal will either stay or go away or come back, depending on the value of hasDecal vs. the previous suppression-state.

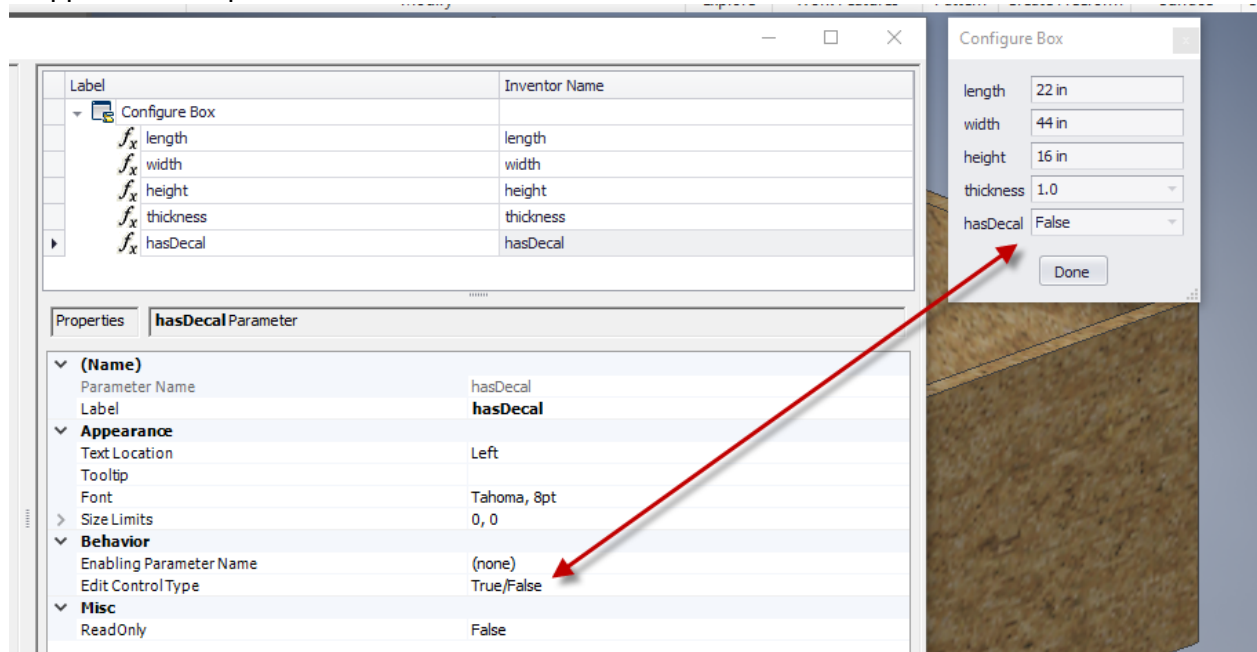
- k. Open the Parameters dialog. Change the value of 'hasDecal' to the opposite setting. You should see the decal appear/disappear to match.



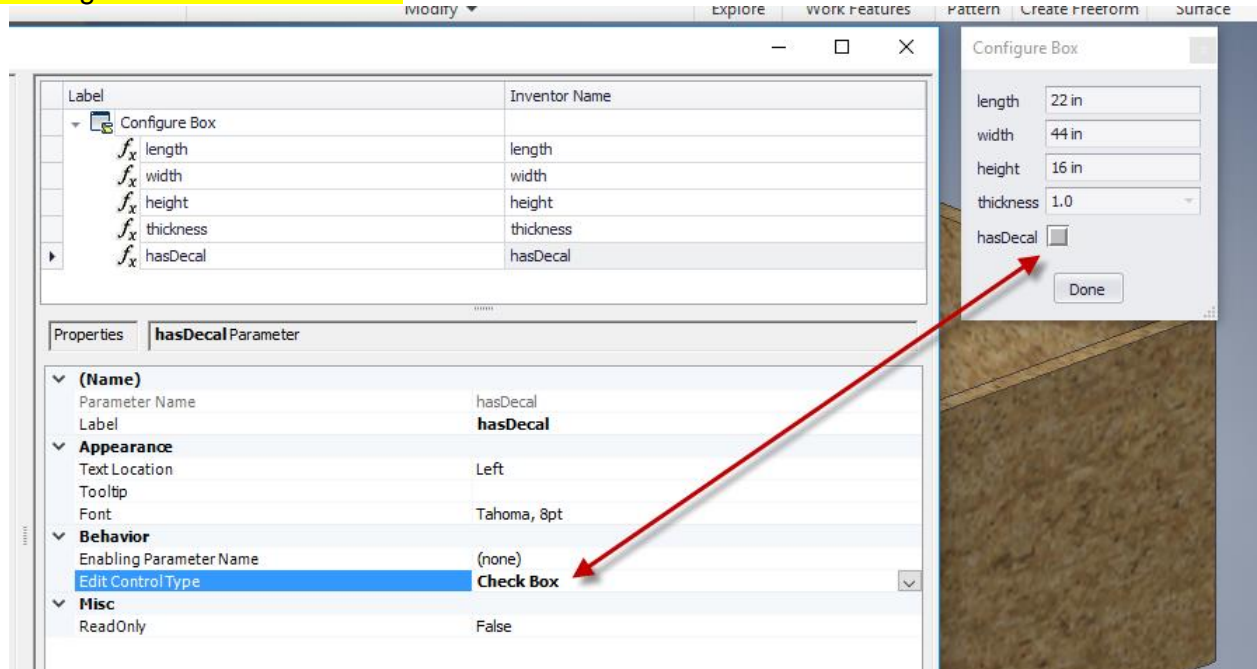
- l. Save the file.

15. Now let's include this on our "Configure Box" form:

- Switch to the iLogic tab, and to the Forms sub-tab.
- Right-click on "Configure Box", and select "Edit". The Form Editor appears.
- Drag "hasDecal" to the Control Area
- It appears in the preview ... but as a True/False Combo Box:

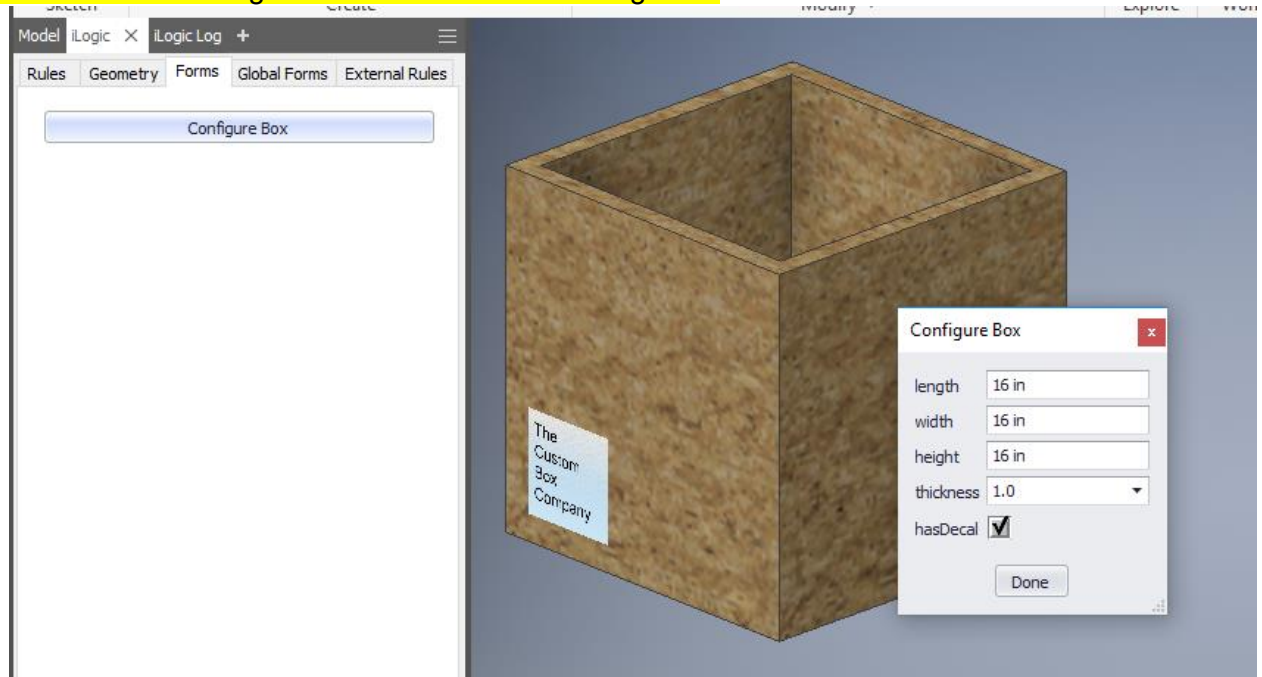


e. Change it to use a Check box:



f. Click "OK" to close the Form Editor.

g. Click on the “Configure Box” button ... and configure it:



h. Save the file.

Additional things to try:

- Change “display names” as shown on form
- Display iProperties on form
- rule-buttons ... run a rule (without implicit triggering) from a form
- Other layout stuff (images, groups)
- Modal / non-modal
- Forms tab is a form! Can add rules-as-buttons and other things.

3. Events / External Rules

1. Another easy way to fire rules.
2. On Open, Save, etc.
3. Internal/external
4. Global events

Exercise 3.1:

Prompt user to initialize a particular iProperty when file is opened/saved.

Phase 1: Do this in a regular “internal” rule, just to test the mechanics.

1. Open project ex3.1.clean
2. Open box.ipt
3. In the iLogic tab, in the Rules sub-tab, double-click on “SetCustomerProject”

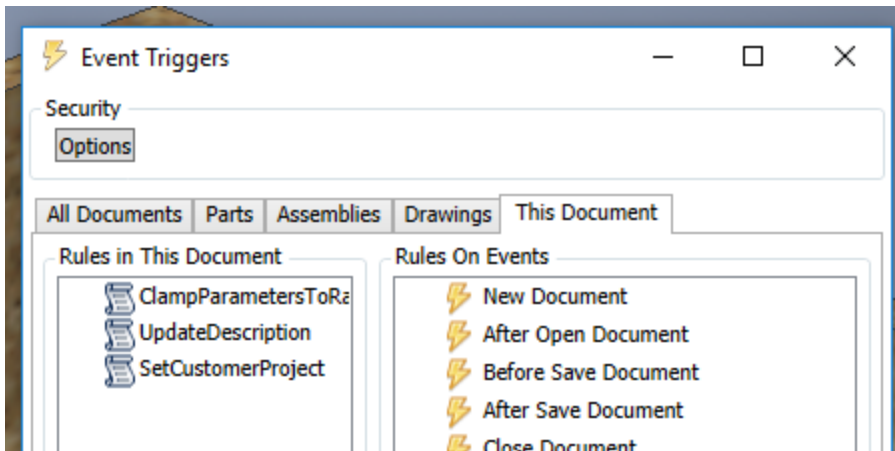
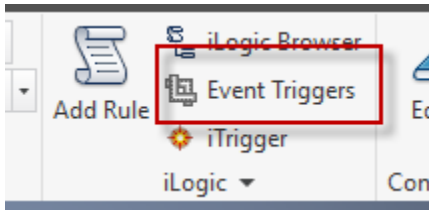
Observe that it has an “if” statement that:

- Checks value of iProperty “Project” (on Project tab). If it is empty string – then prompt user for new value. If user cancels, do nothing, leave it empty.
- “None” or anything else – do nothing, leave it.
- Use “InputBox” to prompt (insert a snippet)

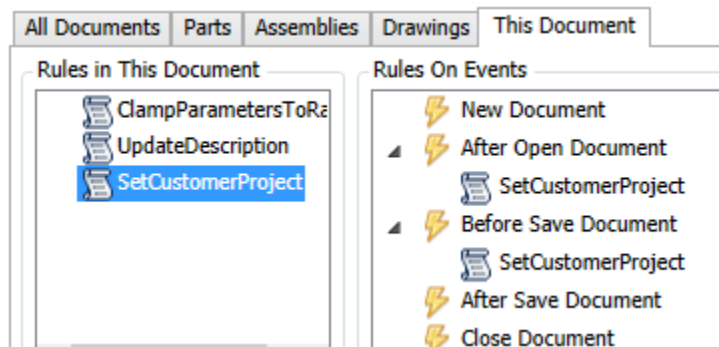
```
If String.IsNullOrEmpty(iProperties.Value("Project", "Project")) Then
    Dim newString = InputBox("Customer project ID, or None", "Project", "")
    If (Not String.IsNullOrEmpty(newString)) Then
        iProperties.Value("Project", "Project") = newString
    End If
End If
```

4. Test the rule:
 - Close the Rule Editor, right-click on the rule, and (since it has no triggers) choose “Run Rule”
 - The “input box” appears. Click “Cancel” The iProperty is *not* changed in this case.

5. Open the “Event Triggers” dialog, (“Manage” ribbon-tab) and be sure it’s on the “This Document” tab,



6. Drag the “SetCustomerProject” rule to both: “After Open Document” and “Before Save Document”.

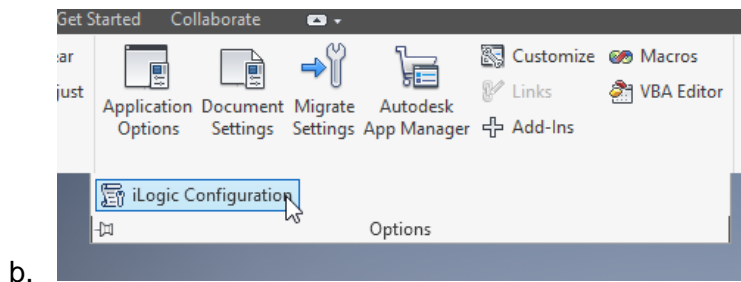
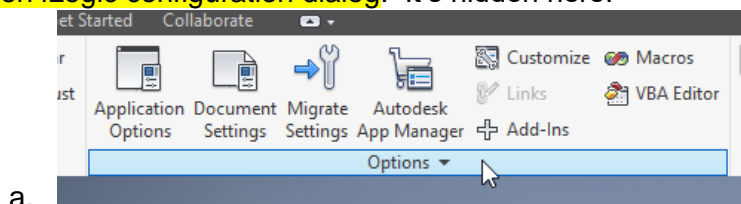


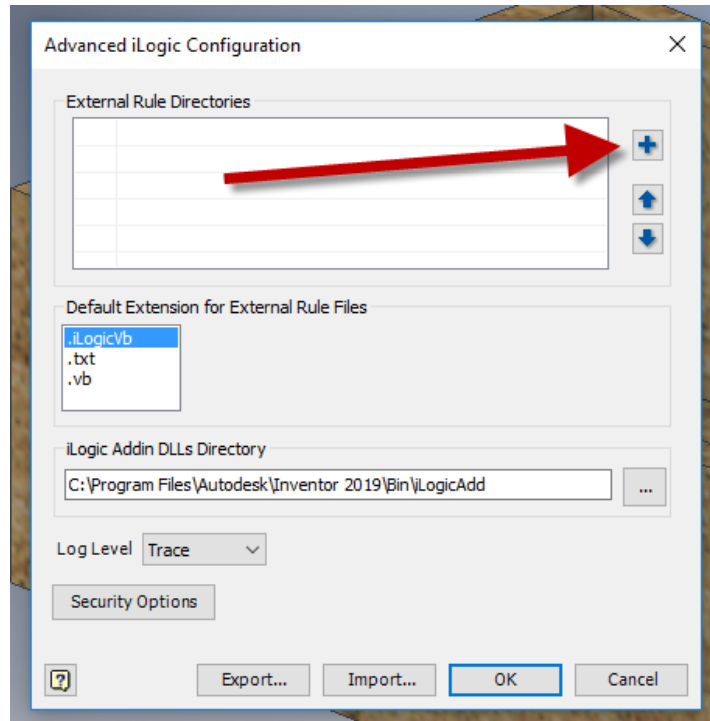
7. Test (save, open, see what happens with OK vs. Cancel buttons)

Phase 2: Make this be the standard behavior for all files:

In “Phase 1”, you learned the basics of using Event Triggers for internal rules. Now we’re going to apply the same concepts to external rules and “global” events.

1. Open Event Triggers dialog, remove the rule from the two events (on “This Document”). Close the dialog.
2. Open iLogic configuration dialog. It’s hidden here:



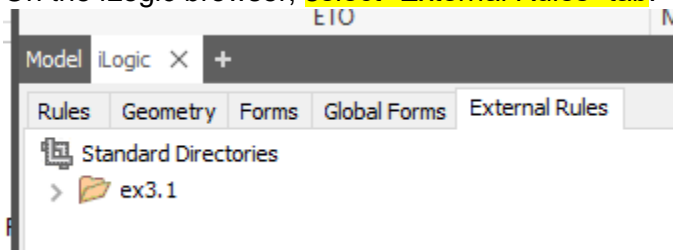


C.

3. In the first box, “External Rule Directories”, use the “+” button to add “this” folder (i.e., the folder with box.ipt) as the (only) external rules folder. (Or any folder is OK, just remember the name.) Click OK.

This tells iLogic where to look for External Rules. Normally/typically, this would be a shared network folder.

4. On the iLogic browser, select “External Rules” tab. The directory is now listed.



(seems to be some cosmetic defect with displaying folder names with multiple “dots”!!)

5. Convert this internal rule to an external rule: (not as easy as it could be, but this isn’t a common thing to do)
 - Go back to the “Rules” tab.

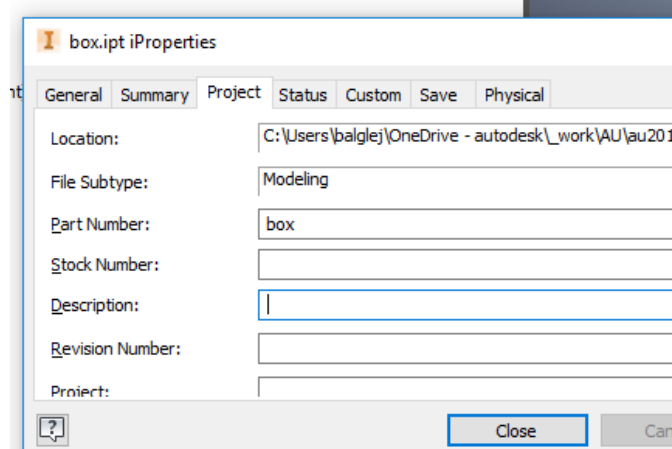
- Edit the SetCustomerProject rule.
 - Select all the text of the rule.
 - Right-click anywhere on the selected text, and select "Save Selected Text". Be sure you're in the correct folder (typically "3.1.clean")
 - Save as file named "SetCustomerProject.iLogicVb"
 - Close the Rule Editor
 - Check that the rule is now present in the **External Rules** tab
 - On the "Rules" tab, right-click and delete the original internal rule.
6. Attach the new external rule to global events:
- Open Event Triggers dialog.
 - Select "All Documents" tab. Note that only External rules are available (in contrast to the "This Document" tab).
 - Drag SetCustomerProject to all three:
 - "New Document"
 - "After Open Document"
 - "Before Save Document"
 - Click OK

7. **Save the document**. You should see the prompt.
8. Test the “new file” behavior:
 - Now **start a new IPT** document. You should see the prompt.
 - **Click OK with no text**.
 - **Make some changes** to the document.
 - **Save**. You should see the prompt again.

Exercise 3.2:

Using iLogic for formatting text for iProperties:

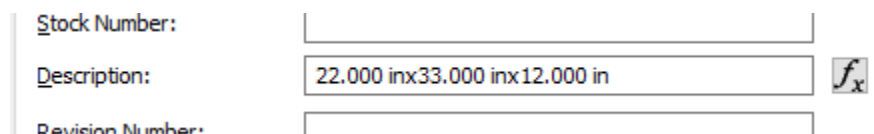
1. Select project ex3.1.clean, if necessary.
2. Open box.ipt if necessary.
3. Open the iProperties dialog. Change to the Project tab:



4. Enter =<length>x<width>x<height> (with the “=” and “<” and “>”), as shown:



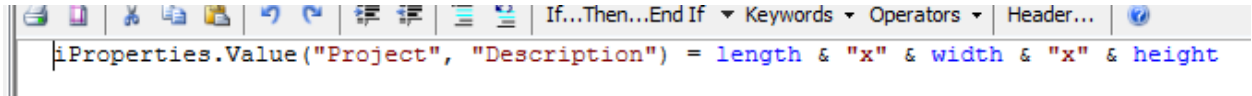
You'll see the results once you press “Enter”. You can edit the “formula” using the “fx” button.



Now the iProperty will always stay up-to-date with respect to the Inventor parameter values. But this “=<param>” mechanism has limited string-formatting capabilities.

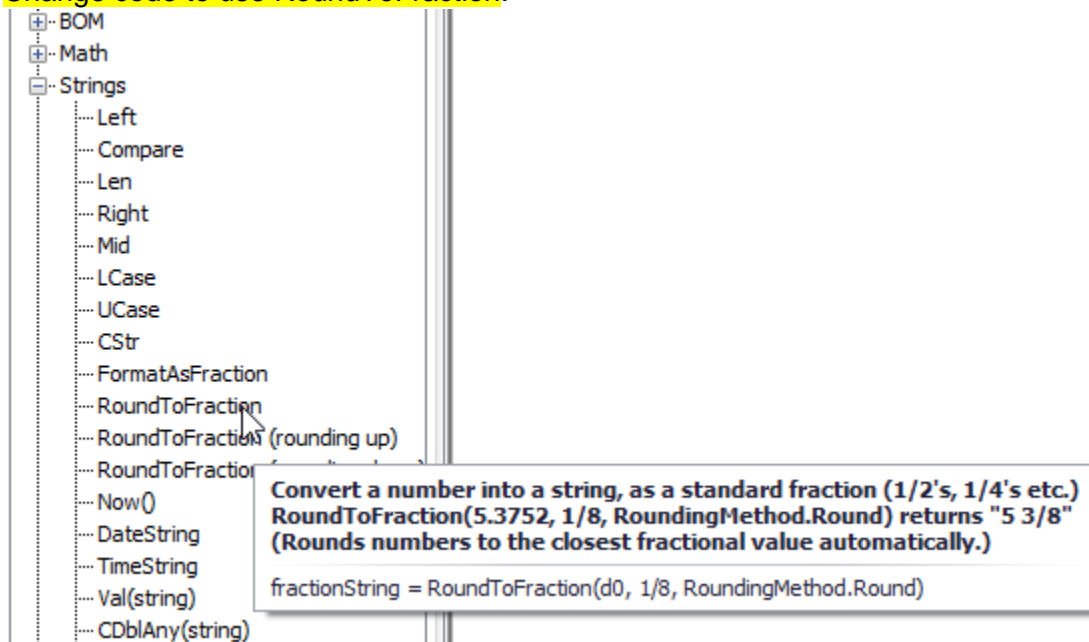
5. Add new rule to update the “Description” iProperty. (You can do this without instructions, right???) Call it “UpdateDescription” or something similar.

6. **Insert code with snippet.** (See the last step in Exercise 1 if you've forgotten how to do this)
7. **Change the value** to refer to parameters: Use "&" to concatenate strings.



```
iProperties.Value("Project", "Description") = length & "x" & width & "x" & height
```

8. **Test**
9. **Change code to use RoundToFraction:**



Convert a number into a string, as a standard fraction (1/2's, 1/4's etc.)
RoundToFraction(5.3752, 1/8, RoundingMethod.Round) returns "5 3/8"
(Rounds numbers to the closest fractional value automatically.)

```
fractionString = RoundToFraction(d0, 1/8, RoundingMethod.Round)
```

```
Dim rndLength = RoundToFraction(length, 1 / 8, RoundingMethod.Round)
Dim rndWidth = RoundToFraction(width, 1 / 8, RoundingMethod.Round)
Dim rndHeight = RoundToFraction(height, 1/8, RoundingMethod.Round)
```

```
iProperties.Value("Project", "Description") = rndLength & " x " & rndWidth &
" x " & rndHeight
```

10. **Test**
11. **Save** file

4. Configuration & GoExcel

1. Configuration of assemblies
 - iLogic enables “Top-down design”
 - Two styles of managing components:
 - “IsActive” and LODs
 - “Managed” components
 - LOD style – suppression, etc.
 - Managed – actual deletion of components
 - Capture Current State (2x)
2. GoExcel

Other topics for exploration, without specific exercises

- Drawings
- Configurator 360 & Design Automation API for Inventor
- For programmers:
 - Programming (loop at the heart)
 - API
 - .Net classes
 - Custom class libs

NOTE: THERE WILL ONLY BE ENOUGH TIME TO DO ONE ADVANCED EXERCISE.

PLEASE CHOOSE EITHER 4.1 OR 4.2 OR 4.3.

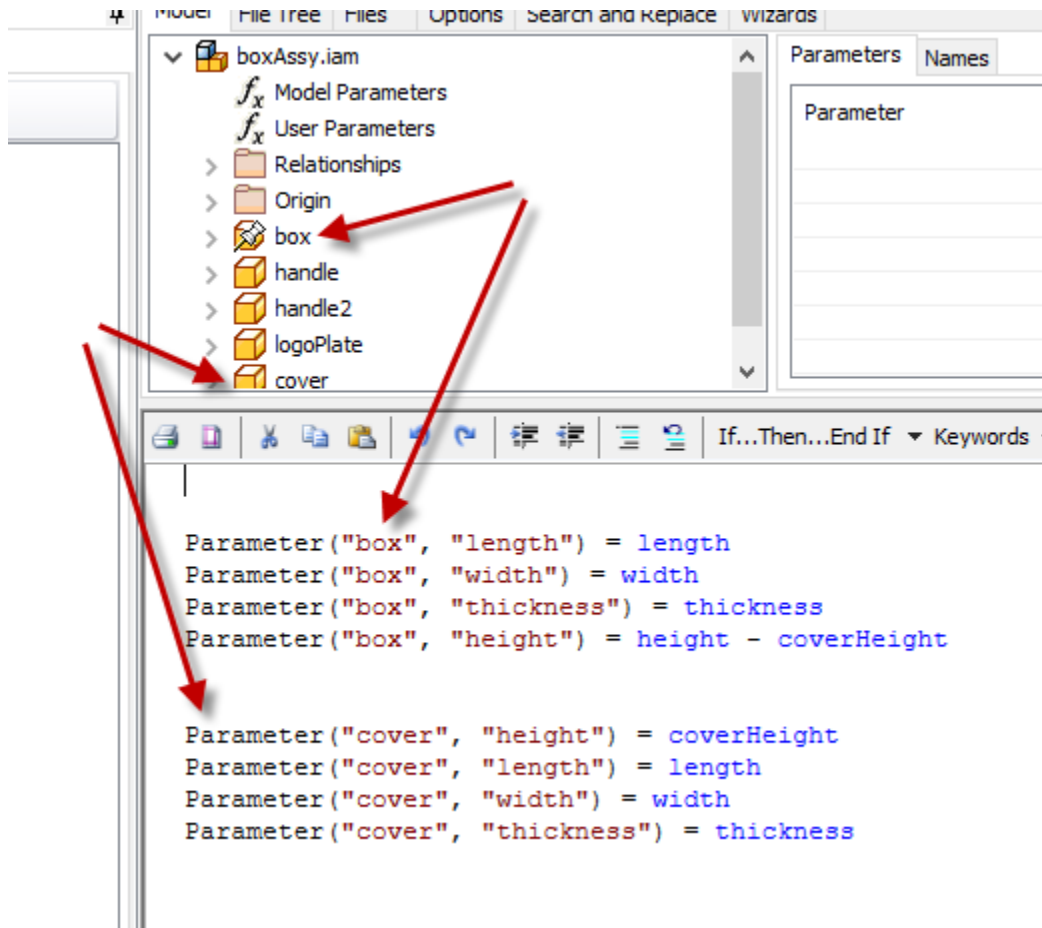
You can do the others at another time.

- 4.1 Assembly configuration: Uses the older, legacy style. If you're more interested in reading and understand existing rules, perhaps from your colleagues, choose this one. Page 49
- 4.2 Assembly configuration: Uses the newer, "managed" style of assembly configuration. Choose this one if you want to learn about the modern style. Page 59
- 4.3 Excel as a data file: If you're more interested in Excel than assembly-configuration, skip right to 4.3. Page 68

Exercise 4.1: Box assembly with conditional cover, LOD-style

1. Open project ex4.1.clean.
2. Open box.iam (**box.ipt is different, iLogic-wise**)
 - You might need to remove your rule from the global events (from last exercise!!)
3. In iLogic Rules tab, edit DimRangeCheck ... just examine it; it should be familiar. Close rule editor.

4. In iLogic Rules tab, edit the “ConfigureComponents” rule. In this rule, we “push” values from parameters of the IAM into the appropriate parameters of the child components. This is a kind of “top-down design”, enabled though the use of iLogic rules.



Please notice:

- We are referring to the child by the name of the occurrence (“box” or “cover”). But we are actually changing the parameter value in the occurrence’s *file* (“box.ipt” or “cover.ipt”). If there was more than one occurrence of the file in the IAM, then they would *all* change size.
- You can also use the filename instead of the occurrence name, but there must be an active occurrence of the file. This will be an issue later.
- It’s a one-way transfer. If you go into the IPT files, you can “manually” change the parameter values and the IAM won’t be aware of it.

Close the Rule Editor

5. Test configuration:

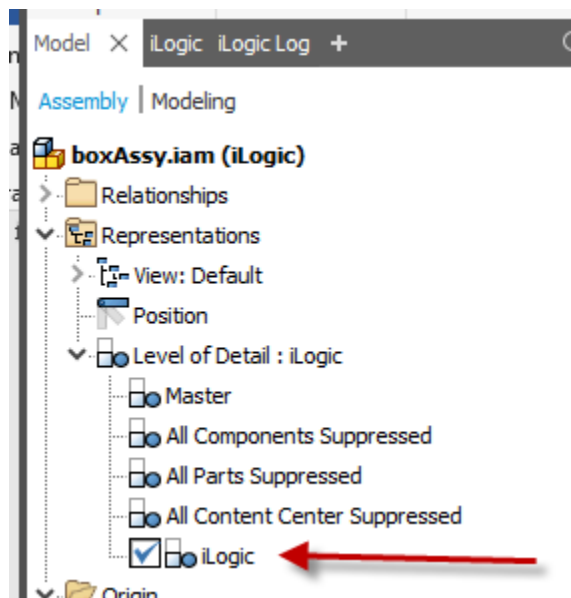
- In the iLogic Forms tab, click on the Configure form.
- Change some of the dimensions.
- Click 'Done' to close the form.
- Save the assembly. Notice: the IPT files were modified too.

Implication: "in real life" you should be working on a copy, or use some other technique (e.g., iParts).

Now we're going to implement the option of having a cover or no cover. We're going to use the "LOD style"

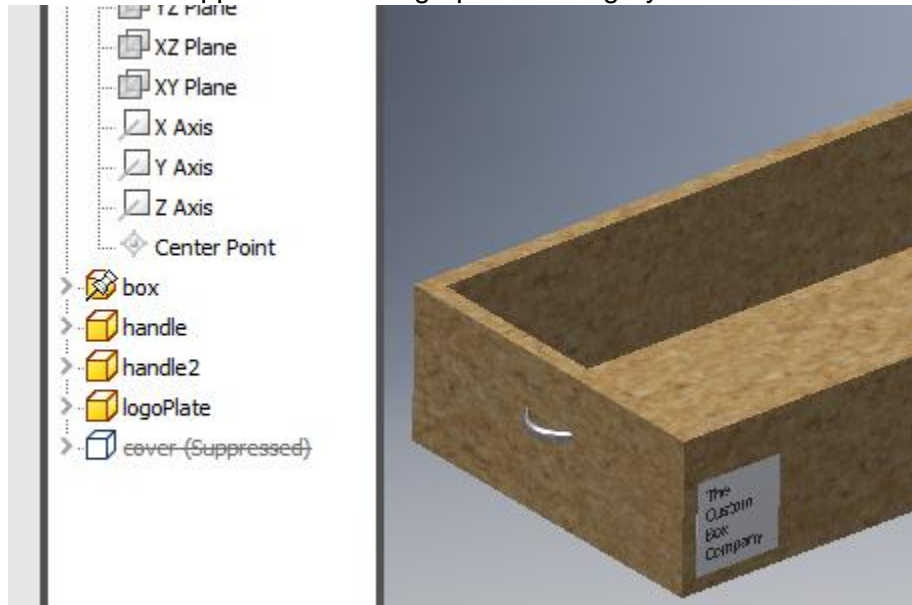
6. Create LOD:

- In the "Model" browser, find & expand "Representations".
- Right-click on "Level of Detail: Master", and select "New Level of Detail".
- A new LOD is created.
- Slow-double-click on the new LOD, to change the name. Change the name to "iLogic" (the name can be whatever you want, but using "iLogic" is common)
- It should automatically be activated (checked), but if not, check the box now to activate it.



7. Just to test, try “manually” suppressing the cover now:

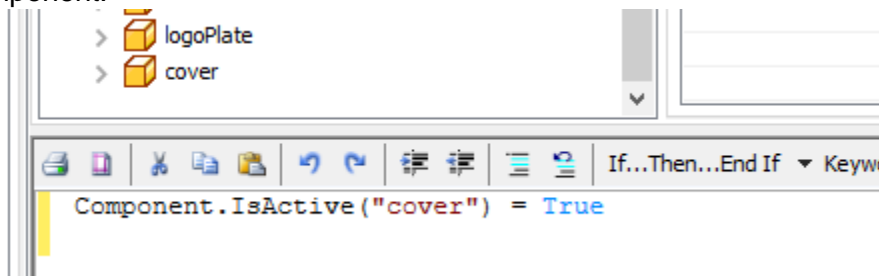
- In the **Model browser**, right click on the “cover” component.
- **Select “Suppress”**
- The cover disappears from the graphics. It is grayed-out in the Model tree.



- Now bring it back: **Right-click on the “cover (Suppressed)” component in the browser and select (uncheck) “Suppress”.**
- You might notice that the number of open files changes (in lower right corner)

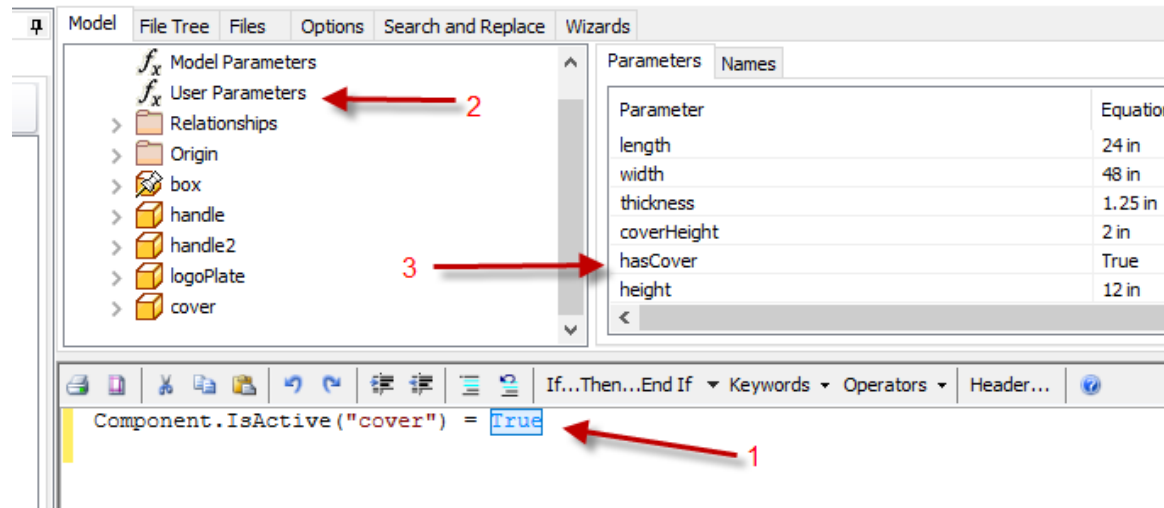
8. Now we’re going to do this with iLogic, based on a parameter.

- In the **iLogic Rules tab**, double-click on **ConfigureComponents** to edit the rule. The rule editor appears.
- Make sure the **cursor is on the blank line at the top** of the rule.
- In the rule editor, there is a “Model” tab at the top. Find the “cover” component, **right-click on it, and select “Capture Current State (IsActive)”**.
- iLogic inserts code into the rule that represents the current state of that component:

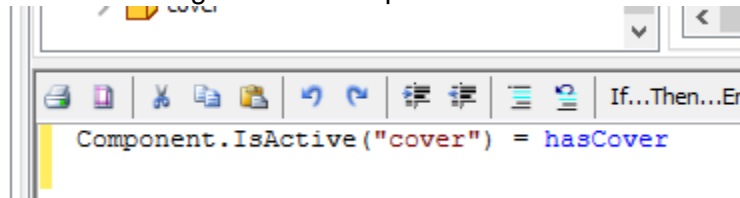


(“IsActive” is the opposite of “Suppressed”, with some additional BOM-related behavior, see below.)

- Now we want to make the “active” status to be equal to the hasCover parameter value. We can use iLogic helpers to make this change:
 - Double-click on “True” to select it.
 - Click on “User Parameters” to show them in the adjacent tab.
 - Double-click on “hasCover”

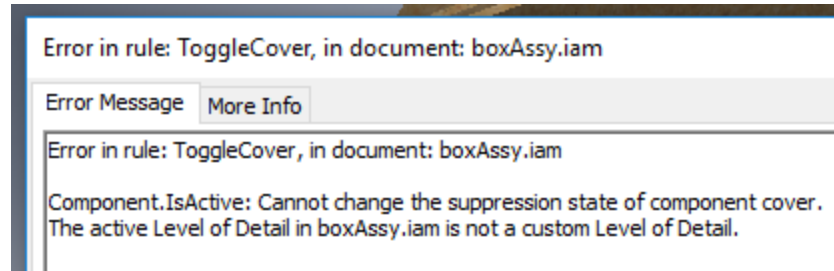


The rule is changed to use the parameter value:



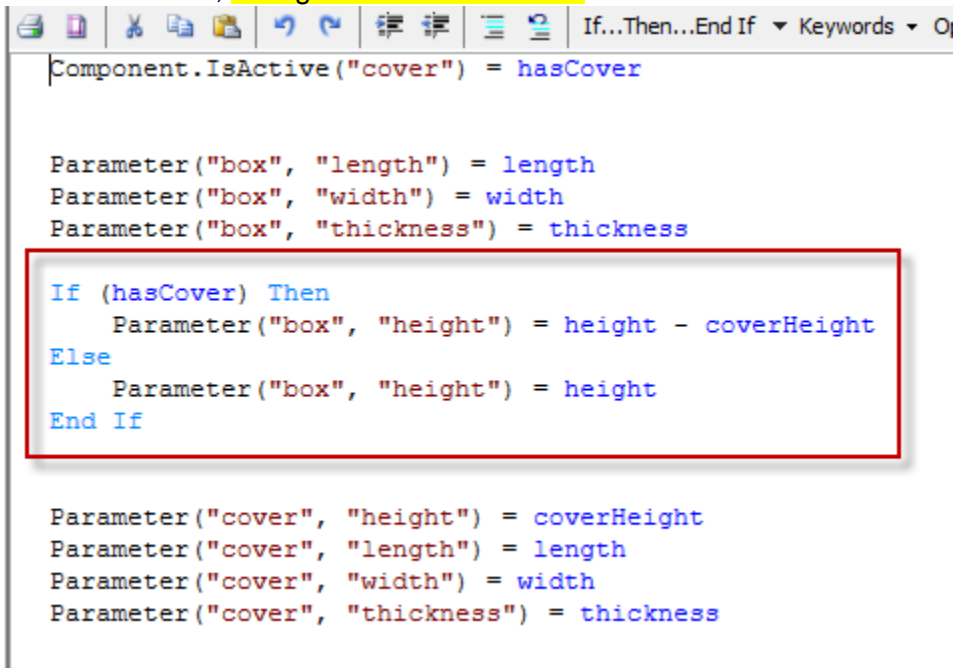
- Save & Run (nothing happens yet)
- Open the Parameters dialog. (Move it so you can see the box in the graphics.) Change “hasCover” from True to False. The cover goes away. Change it from False back to True. The cover comes back.
- Close the Parameters dialog.
- Save the file.

Note that you might see this error message when running the rule, if you change the active LOD back to Master:



9. One more small thing. When the cover is NOT present, we want the box's height to be the total height (i.e., not subtract the coverHeight). So we need to change the ConfigureComponents to account for that:

- On iLogic "Rules" tab, double-click ConfigureComponents to edit.
- In the rule editor, change the code as follows:



```

Component.IsActive("cover") = hasCover

Parameter("box", "length") = length
Parameter("box", "width") = width
Parameter("box", "thickness") = thickness

If (hasCover) Then
    Parameter("box", "height") = height - coverHeight
Else
    Parameter("box", "height") = height
End If

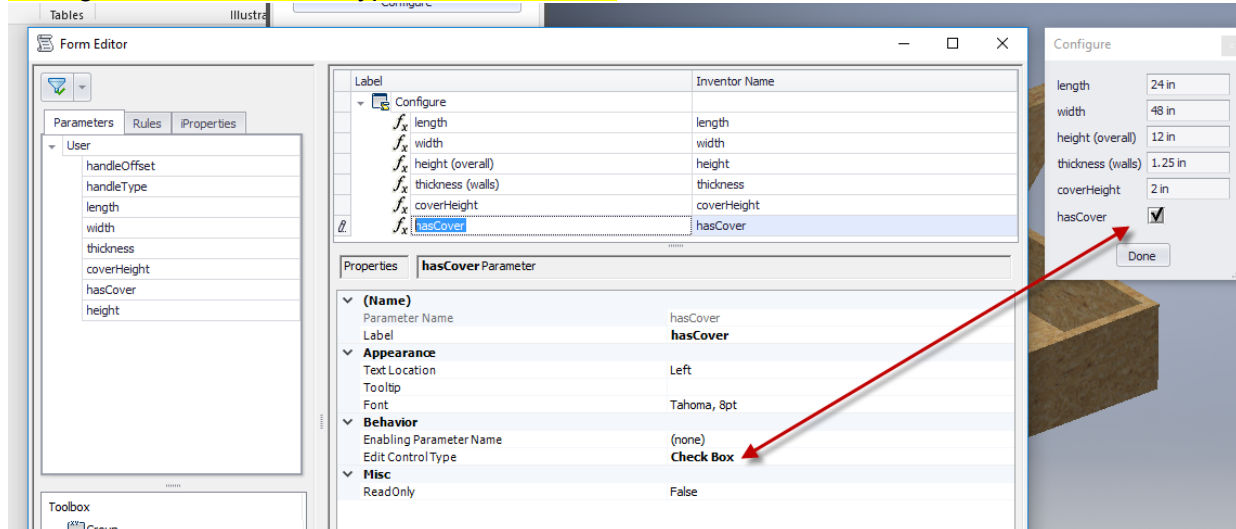
Parameter("cover", "height") = coverHeight
Parameter("cover", "length") = length
Parameter("cover", "width") = width
Parameter("cover", "thickness") = thickness
  
```

This means the "height" of the cover part no longer contributes to the height of the box part (when inactive).

- Save and Run

10. Now let's update the "Configure" form with this behavior:

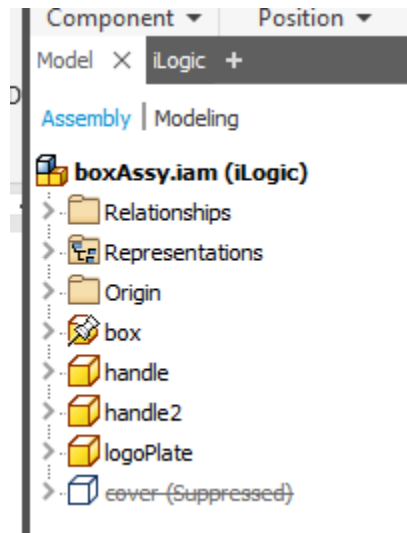
- On iLogic "Forms" tab, Right-click on "Configure", select "Edit"
- Drag the "hasCover" parameter to the form layout
- Change the "Edit Control Type" to "Check Box"



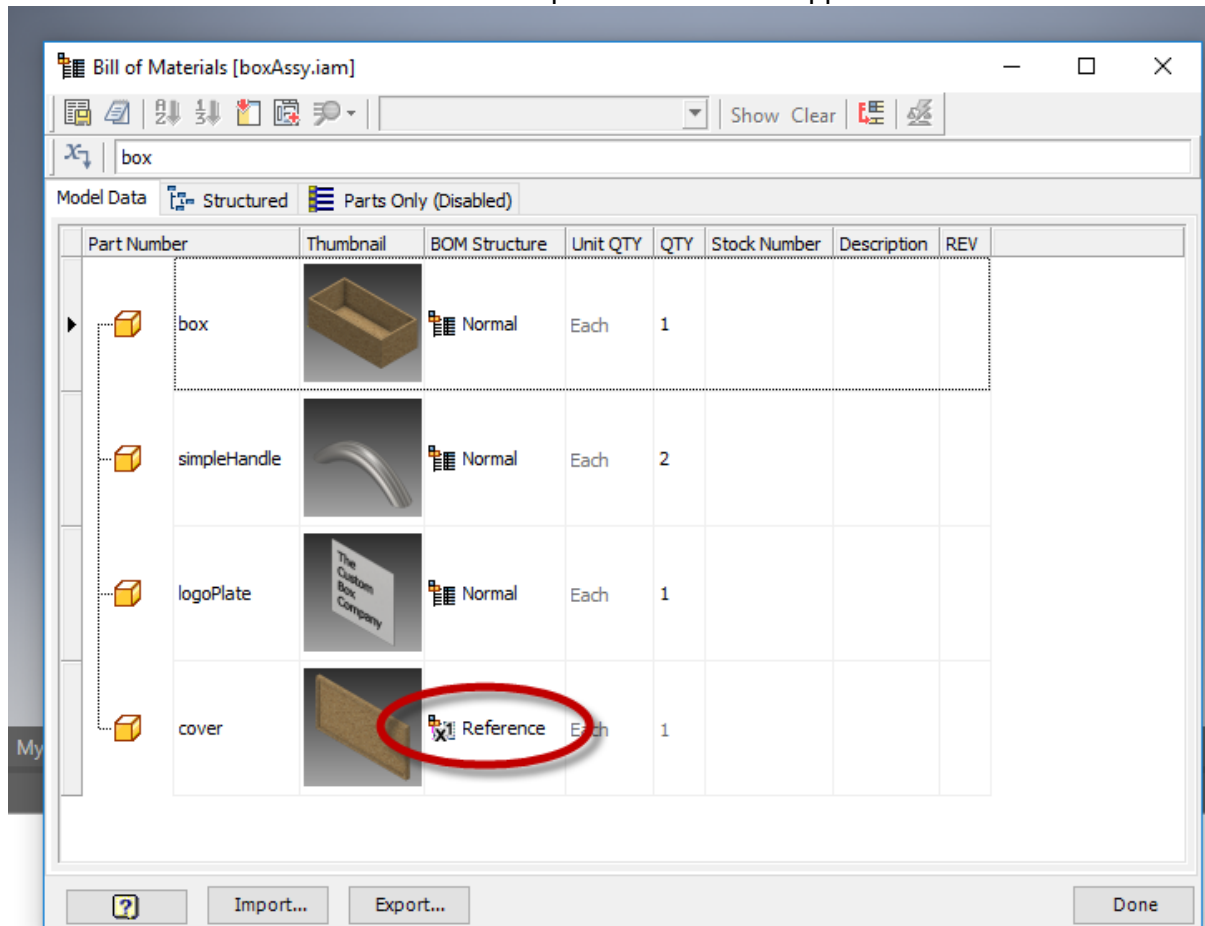
- Click OK to close the Form Editor.
- Click on Configure, and test the check box on the form.
- Save the file.

11. Observations about this technique:

- All suppressed components must actually be listed in the model, even if Inventor can save some memory and kinda-sorta make them not visible. If you have a lot of potential variations, you'll have a huge "master model" (all the suppressed components are still listed in the model browser).



- BOM. The BOM always comes from the “Master” LOD. iLogic’s “isActive” function tricks the BOM by setting the inactive parts’ BOM Structure to “Reference”. “Reference” means the component should not appear in the BOM.

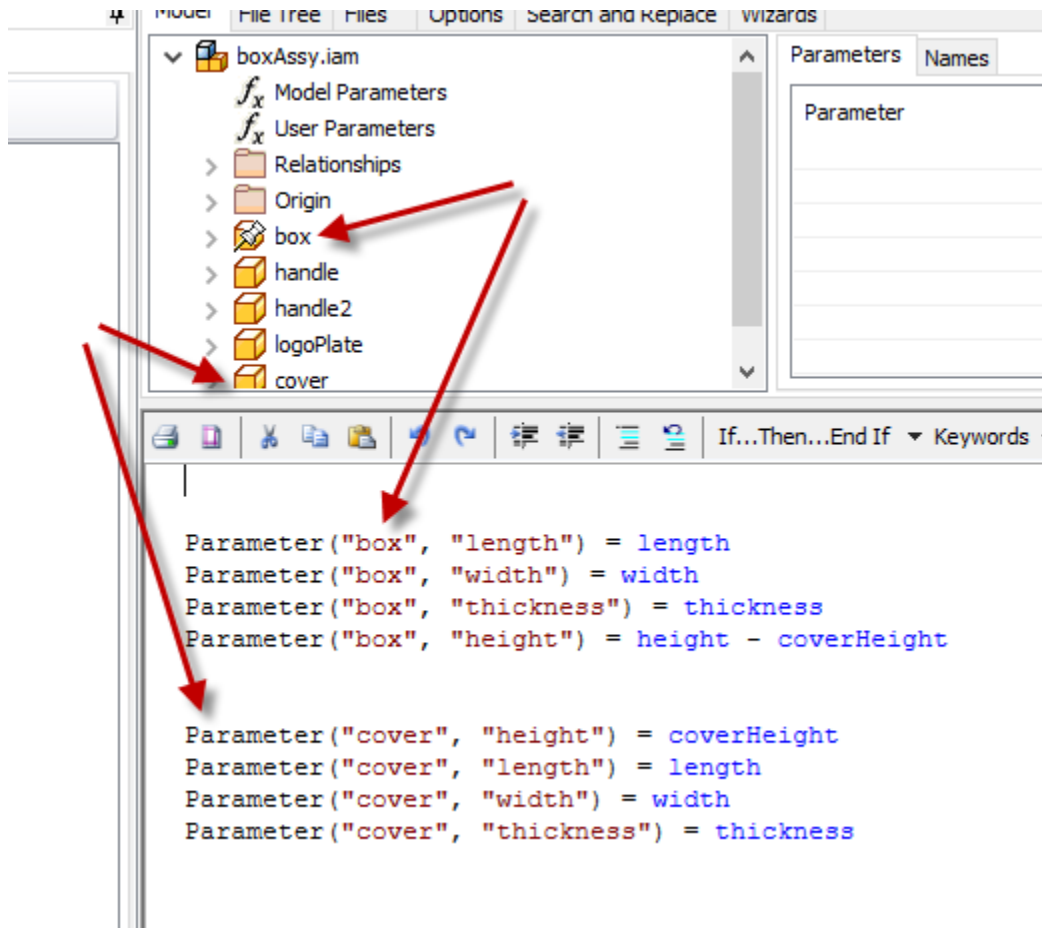


Exercise 4.2: Box assembly with conditional cover, Managed-style

Same thing with the “Managed” technique.

1. Save and close all documents.
2. Open the ex4.2.clean project.
3. Open box.iam (**box.ipt is different, iLogic-wise**)
 - You might need to remove your rule from the global events (from last exercise!!)
4. In iLogic Rules tab, edit DimRangeCheck ... just examine it; it should be familiar. Close rule editor.

5. In iLogic Rules tab, edit the “ConfigureComponents” rule. In this rule, we “push” values from parameters of the IAM into the appropriate parameters of the child components. This is a kind of “top-down design”, enabled though the use of iLogic rules.



Please notice:

- We are referring to the child by the name of the occurrence (“box” or “cover”). But we are actually changing the parameter value in the occurrence’s *file* (“box.ipt” or “cover.ipt”). If there was more than one occurrence of the file in the IAM, then they would *all* change size.
- You can also use the filename instead of the occurrence name, but there must be an active occurrence of the file. This will be an issue later.
- It’s a one-way transfer. If you go into the IPT files, you can “manually” change the parameter values and the IAM won’t be aware of it.

Close the Rule Editor

6. Test configuration:

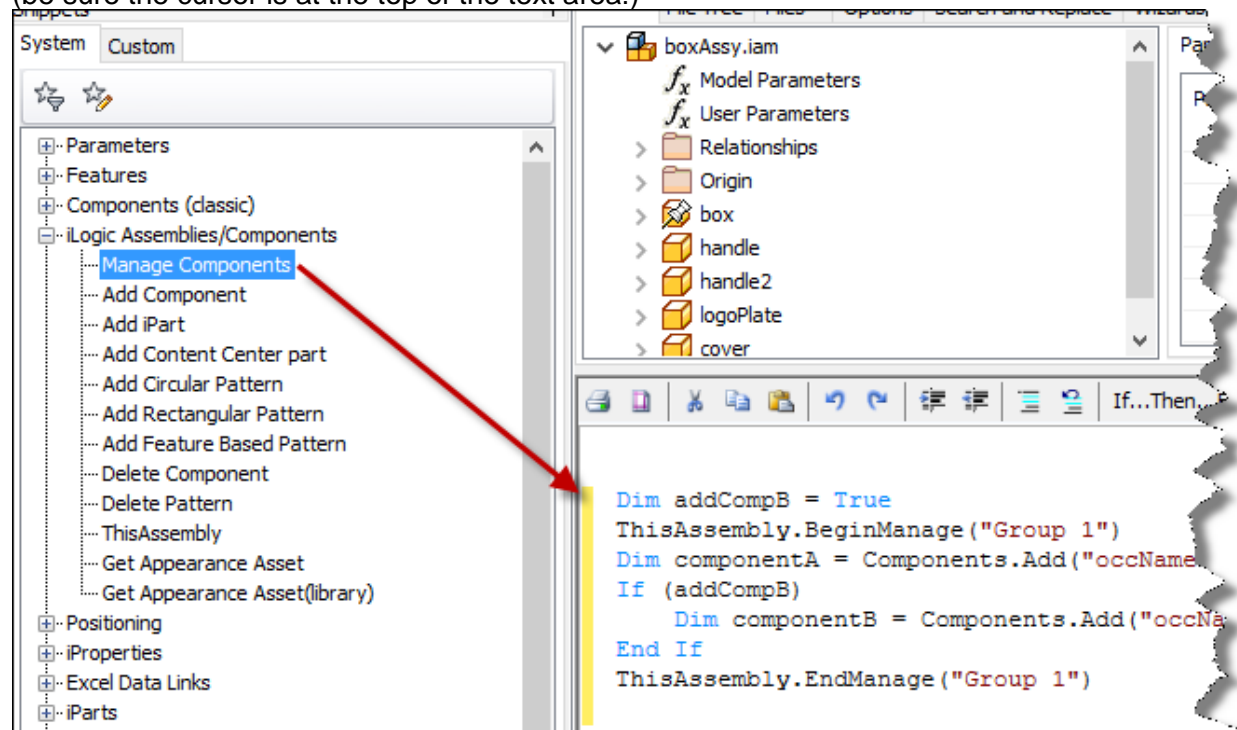
- In the iLogic Forms tab, click on the Configure form.
- Change some of the dimensions. (Don't try the "hasCover" checkbox.)
- Click 'Done' to close the form.
- Save the assembly. Notice: the IPT files were modified too.

Implication: "in real life" you should be working on a copy, or use some other technique (e.g., iParts).

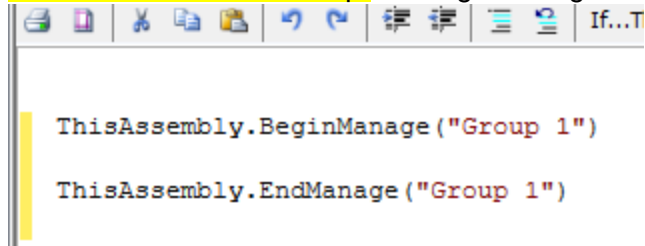
7. Edit the rule "ConfigureComponents"

8. Add code to the rule:

- a. Use the "Manage Components" snippet to insert some code as shown:
 (be sure the cursor is at the top of the text area!)



- b. Remove all the code except for BeginManage/EndManage:



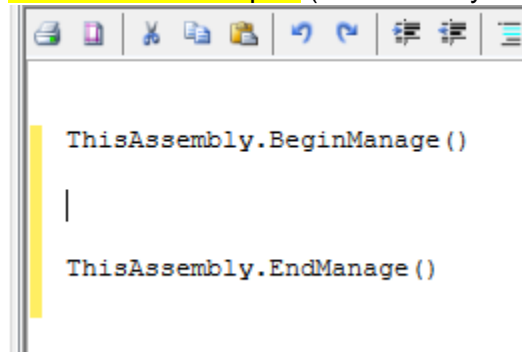
```

ThisAssembly.BeginManage("Group 1")

ThisAssembly.EndManage("Group 1")

```

- c. Remove the "Group 1" (unnecessary except in very complex scenarios)



```

ThisAssembly.BeginManage()

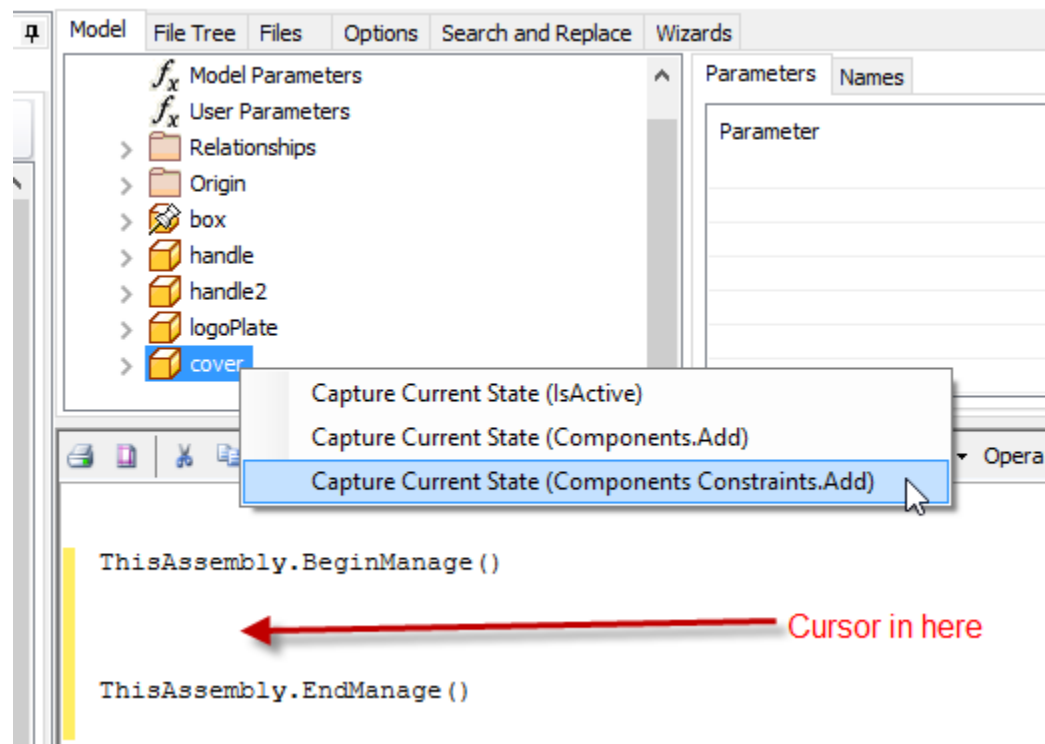
|

ThisAssembly.EndManage()

```

This is the basic structure of "Managed" assemblies. Within this pair, you state the conditions under which you want components and constraints created. iLogic takes care of cleaning up under any other conditions.

- d. Capture current state of the "cover":



The screenshot shows the iLogic interface with the 'Model' browser on the left. The 'cover' component is selected. A context menu is open over the 'cover' component, showing three options: 'Capture Current State (IsActive)', 'Capture Current State (Components.Add)', and 'Capture Current State (Components Constraints.Add)'. The third option is highlighted. Below the browser, the iLogic code editor shows the following code:

```

ThisAssembly.BeginManage()

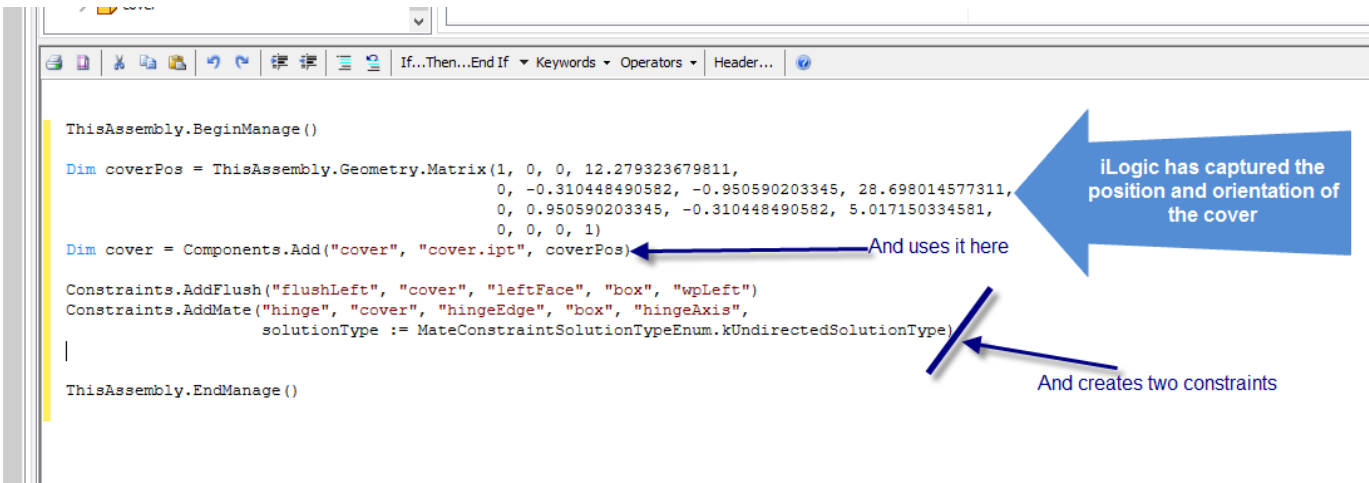
|

ThisAssembly.EndManage()

```

A red arrow points to the vertical bar (|) in the code editor, with the text "Cursor in here" next to it.

e. The code is inserted:



```

ThisAssembly.BeginManage()

Dim coverPos = ThisAssembly.Geometry.Matrix(1, 0, 0, 12.279323679811,
                                             0, -0.310448490582, -0.950590203345, 28.698014577311,
                                             0, 0.950590203345, -0.310448490582, 5.017150334581,
                                             0, 0, 0, 1)

Dim cover = Components.Add("cover", "cover.ipt", coverPos)

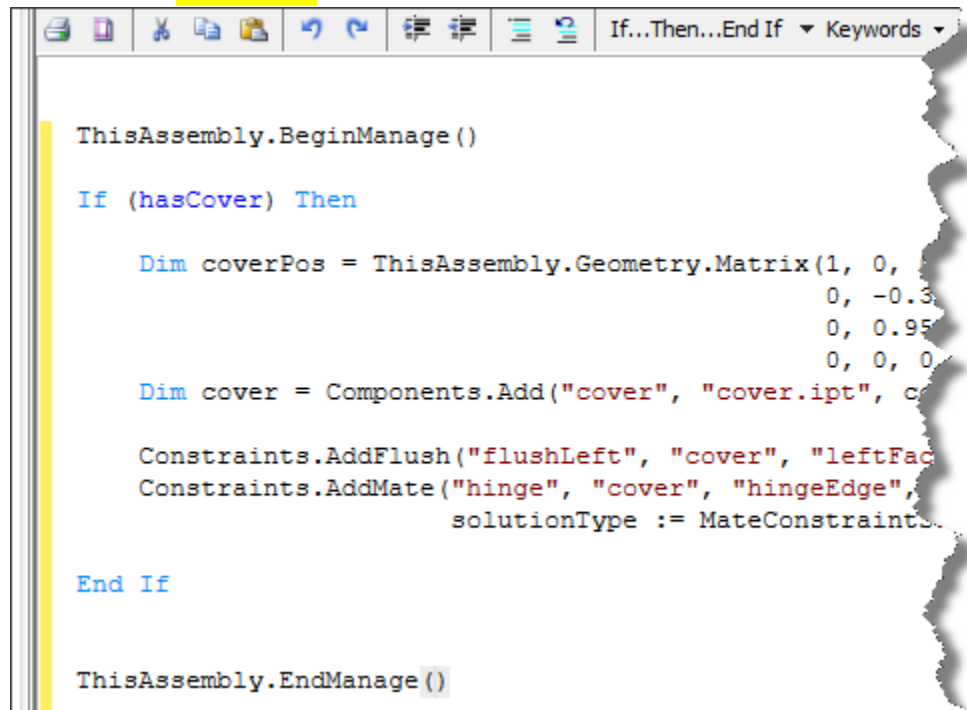
Constraints.AddFlush("flushLeft", "cover", "leftFace", "box", "wpLeft")
Constraints.AddMate("hinge", "cover", "hingeEdge", "box", "hingeAxis",
                   solutionType := MateConstraintSolutionTypeEnum.kUndirectedSolutionType)

ThisAssembly.EndManage()
  
```

Annotations:

- Blue arrow pointing to the `coverPos` matrix: "iLogic has captured the position and orientation of the cover"
- Blue arrow pointing to the `Components.Add` line: "And uses it here"
- Blue arrow pointing to the `Constraints.Add` lines: "And creates two constraints"

f. But that's not very interesting yet; it has just reproduced the same state we had. So now let's add an "if":



```

ThisAssembly.BeginManage()

If (hasCover) Then

    Dim coverPos = ThisAssembly.Geometry.Matrix(1, 0, 0, 12.279323679811,
                                                0, -0.310448490582, -0.950590203345, 28.698014577311,
                                                0, 0.950590203345, -0.310448490582, 5.017150334581,
                                                0, 0, 0, 1)

    Dim cover = Components.Add("cover", "cover.ipt", coverPos)

    Constraints.AddFlush("flushLeft", "cover", "leftFace", "box", "wpLeft")
    Constraints.AddMate("hinge", "cover", "hingeEdge", "box", "hingeAxis",
                      solutionType := MateConstraintSolutionTypeEnum.kUndirectedSolutionType)

End If

ThisAssembly.EndManage()
  
```

Now the component and constraints will only be created when "hasCover" is true. When "hasCover" is False, *iLogic* takes care of making sure the component and constraints are removed.

g. Click "Save & Run" and see what happens (nothing).

- h. **Use the Form to toggle the cover** (don't change other parameters yet). Things to notice:
- i. Look at the Model browser to see what happens to the components.
 - ii. There's no iLogic custom LOD (as there would be in the other style).
 - iii. And of course the BOM works normally; no additional "reference" components. (as there would be in the other style)

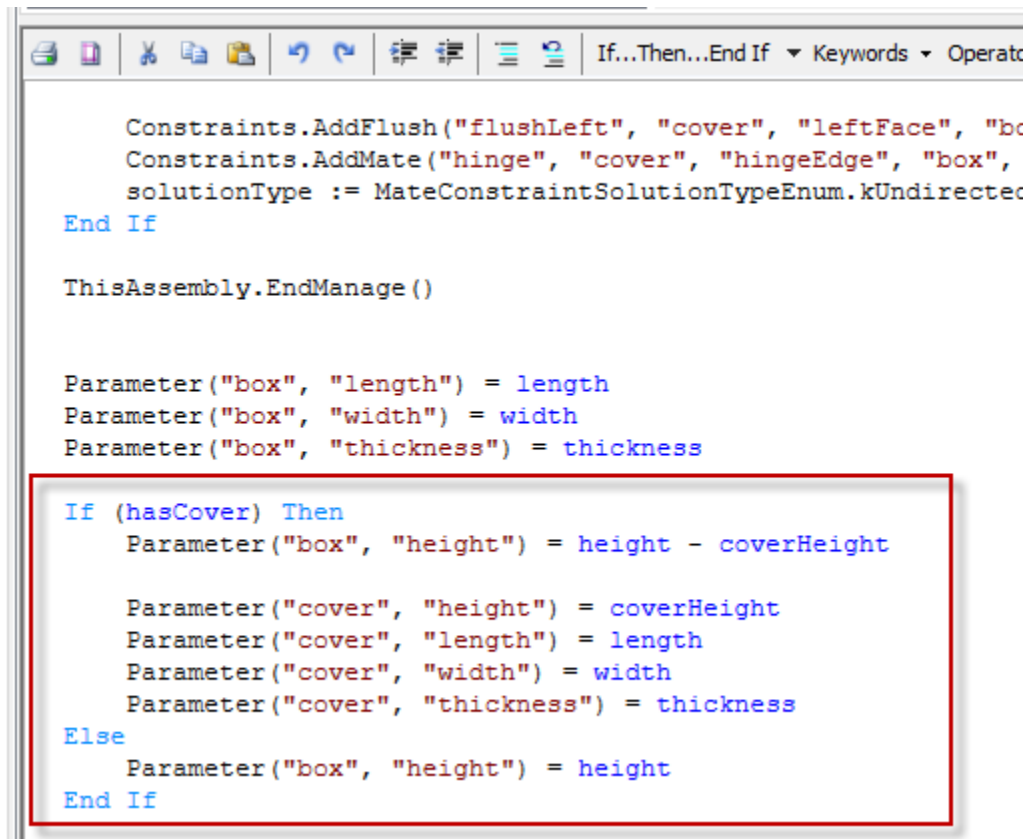
Close the Form.

9. About the other parameters: With the “managed” technique, when there is no-cover, there’s really no “cover” component at all! It has really been deleted. So all of the parameter-setting statements, like “Parameter(“cover”, “height”) = coverHeight”, will fail. The solution is only to run those statements when the cover is active. So:
- Edit the ConfigureComponents rule.**
 - Add an “if” statement to enclose the cover-related statements,** like this:

```
If (hasCover) Then
    Parameter("box", "height") = height - coverHeight

    Parameter("cover", "height") = coverHeight
    Parameter("cover", "length") = length
    Parameter("cover", "width") = width
    Parameter("cover", "thickness") = thickness
Else
    Parameter("box", "height") = height
End If
```

Image showing this code in context:

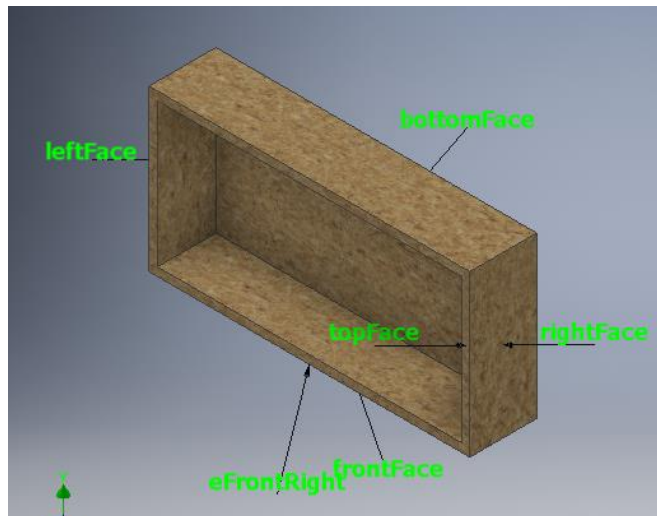
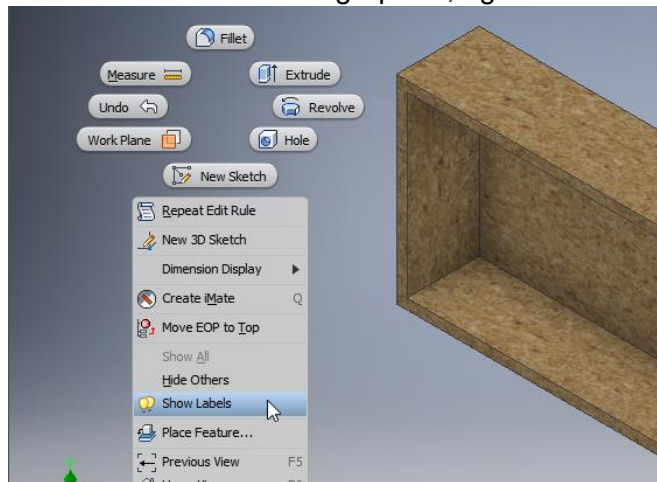


Now we have a fully customizable “box”!

10. (The following section is just for reading & learning. You don't have to actually do any of it to complete the exercise.)

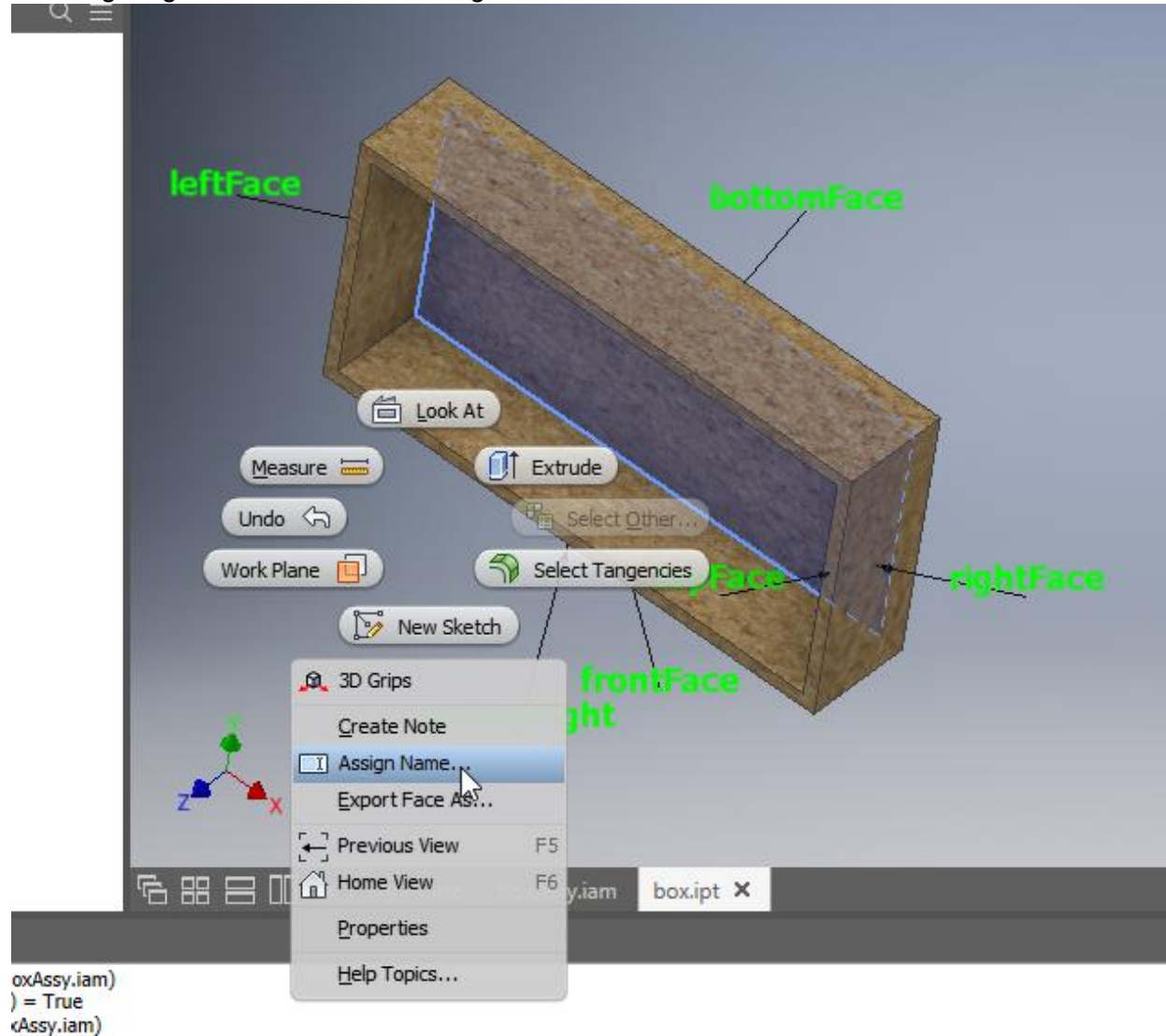
About those face & edge names: If necessary, iLogic will create face and edge names. But you can also specify “nicer” names yourself. As you can see in the rule, there are names like “leftFace” and “hingeEdge”. Those come from the model in the IPT itself:

- Open box.ipt
- Show the “Model” tab if not visible.
- Notice that there are workfeatures with “good” names, such as “hingeAxis”. This is one source of the names.
- In the “blank area” of the graphics, right-click and then select “Show Labels”:



This view shows the named faces and edges. There is a corresponding “Hide Labels” command.

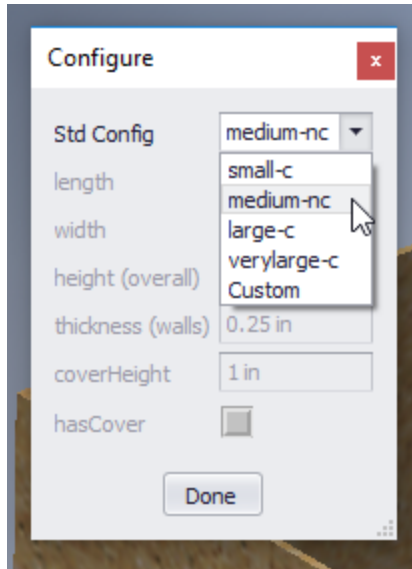
- e. When you are making a part, you can name a face or edge. Select the face/edge, right click, and select “Assign Name”:



Note that if you change the name of a face/edge, you'll have to manually update any rules that used the old name.

Exercise 4.3: Predefined configurations stored in Excel file

In this exercise, we'll use an Excel spreadsheet to provide predefined options for "standard configurations" for our box. The user should be able to select from the available configurations from a drop-down menu, like this:



This is similar to using an iAssembly, but the configuration is defined externally in the spreadsheet. (Using iAssembly is tricky for controlling subcomponents; all components have to be iParts and you need to specify iPart-rows for each one ... which is OK, just ... tricky).

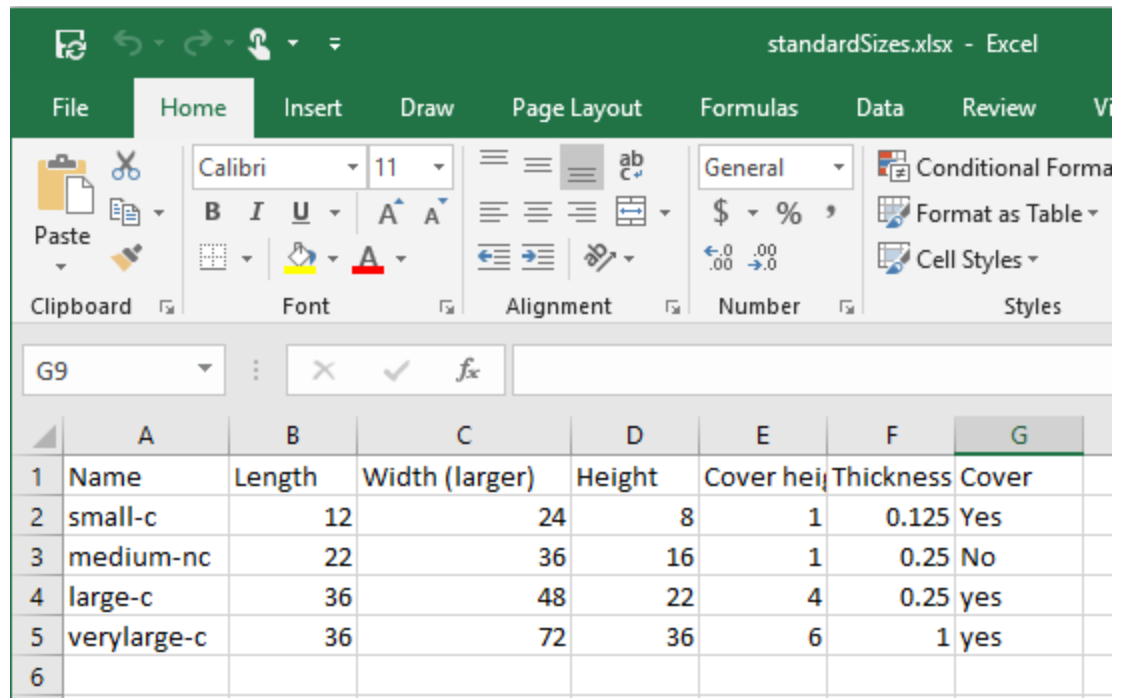
Since the "standard configurations" are being saved in an external spreadsheet file, the first thing we want to do is to retrieve (update) the list of all the standard configurations, since someone else might have updated it since the last time we saved the IAM file. We should do that when we first open the IAM.

In summary, we're reading Column A from the spreadsheet, and using it to populate the "multi-value" of a parameter. Plus adding "Custom". Then we'll disable the parameter-controls on the form, unless the user selects "Custom".

1. If necessary, save and close all files.
2. Select project ex4.3.clean.ipj
3. Open boxAssy.iam (same as 4.2 finished ... if you're just continuing with same files, be sure to copy standardSizes.xlsx file in 4.3.clean folder)
4. Create these parameters:
 - a. Create new TEXT parameter, "ConfigurationName". Value = "Custom"
 - b. Create new TRUE/FALSE parameter, "IsCustom". Value = True

Note: Watch the capitalization ... Inventor parameter names are case-sensitive.

- c. Open the spreadsheet with Excel, ("standardSizes.xlsx") just to see what's in it. Note the column-names.



	A	B	C	D	E	F	G
1	Name	Length	Width (larger)	Height	Cover hei	Thickness	Cover
2	small-c	12	24	8	1	0.125	Yes
3	medium-nc	22	36	16	1	0.25	No
4	large-c	36	48	22	4	0.25	yes
5	verylarge-c	36	72	36	6	1	yes
6							

Close Excel.

- d. Create a new rule called "UpdateConfigurationList"

- e. **Add the following contents to the rule** (copy-and-paste from the code below ... watch out for line-wrapping problems):

```
Dim configs As New ArrayList

Dim dbpath As String = System.IO.Path.Combine(ThisDoc.Path,
"standardSizes.xlsx")
GoExcel.Open(dbpath, "Sheet1")

For i = 2 To 100 'Skip header row. And don't go on an infinite
loop, just in case there's some error.
    Dim m As String
    m = GoExcel.CellValue("A" & i)
    If (m = "") Then
        Exit For
    End If
    configs.Add(m)
    Logger.Debug(m)
Next i
configs.Add("Custom")

MultiValue.List("ConfigurationName") = configs
```

- f. Let's review what this code does:

- i. The first and last lines go together, to address the ultimate purpose of setting the “multi-value” for the ConfigurationName parameter:

```
Dim configs As New ArrayList
```

And

```
MultiValue.List("ConfigurationName") = configs
```

MultiValue.List expects an ArrayList, so we need to construct one.

- ii. Open the spreadsheet:

```
Dim dbpath As String = System.IO.Path.Combine(ThisDoc.Path, "standardSizes.xls")
GoExcel.Open(dbpath, "Sheet1")
```

System.IO.Path.Combine is a .Net library method. It takes a directory-string and a file-string and combines them together. You don't have to worry about which one has the “/”, or not.

ThisDoc.Path returns a string containing the directory path to the current file.

GoExcel.Open makes the given spreadsheet be the “active” spreadsheet for subsequent GoExcel method calls.

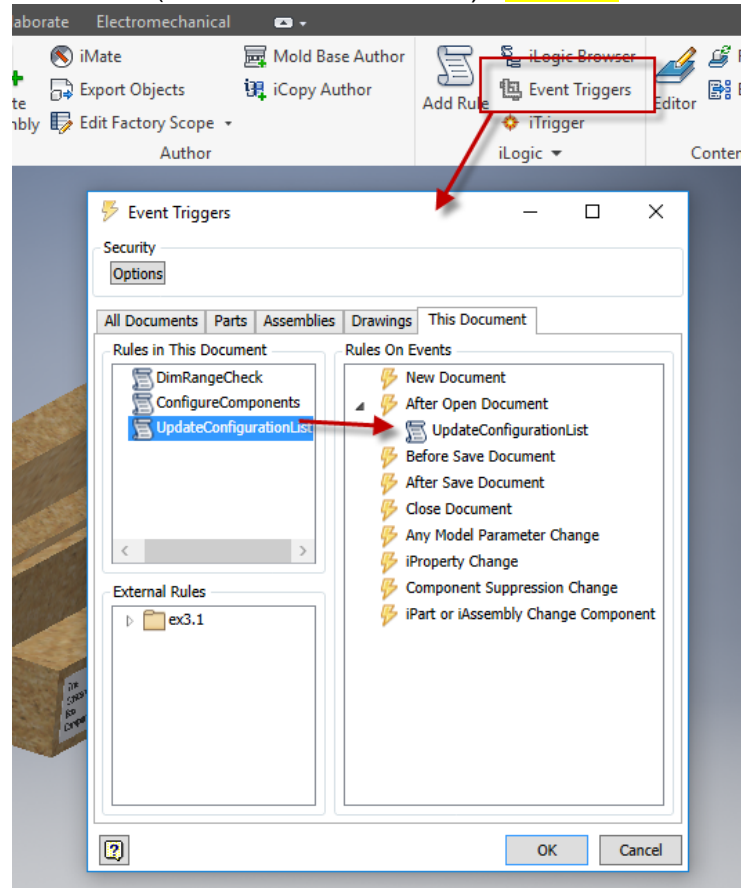
- iii. Get the configuration names from the spreadsheet.
 - The ‘for’ loop starts at 2 in order to skip the header row.
 - The ‘for’ loop stops at 100 at max. It’s common to make some mistake in loops (especially “while” loops). If your iLogic rule goes into an infinite loop, there’s no way to stop it gracefully. You have to kill Inventor using the Task Manager. And you’ll lose everything since your last save. So be careful.
 - GoExcel.CellValue – gets the value of the specified cell, from the “active” spreadsheet. Note that we’re using string concatenation (“&”) to build the cell reference.
 - If the cell is empty, then exit the loop. This allows us to have an arbitrary number of configurations (up to 99).
 - Configs.Add – this adds the name (from the cell) into the arrayList.
 - Logger.Debug – shows this message in the iLogic Log window, IF you’ve set the Log Level to Debug or Trace. Otherwise, the message doesn’t appear.
 - Configs.Add(“Custom”) -- we always want this pseudo-configuration.

g. **Save & Run** the rule, if you haven’t already.

h. **Open the Parameters** dialog, and look at the **choices for ConfigurationName**. Leave it set on “Custom” for now.

i. **Save the file**.

- j. Open the iLogic Event Triggers dialog, and drag the new rule under After Open Document (on “This Document” tab). Click OK



- k. Save and close the IAM file.
- l. Update the spreadsheet with a new configuration:
- Open the spreadsheet in Excel.
 - Add another configuration name (don't worry about the other parameters).
 - Save the spreadsheet and **exit from Excel**.
- m. Back in Inventor, open the IAM file. Open the Parameters dialog. See if your new configuration name is listed in the choices under ConfigurationName. If you get an error of some kind, maybe you forgot to close the file in Excel? In that case, you may need to restart Inventor too.
- n. Open Excel again, and remove your new configuration name. Save the spreadsheet file, and close Excel.
- o. Back in Inventor, close the IAM, and then open it again. Open the Parameters dialog and look at the ConfigurationName parameter. Is your new config-name now gone?
- p. Wow!

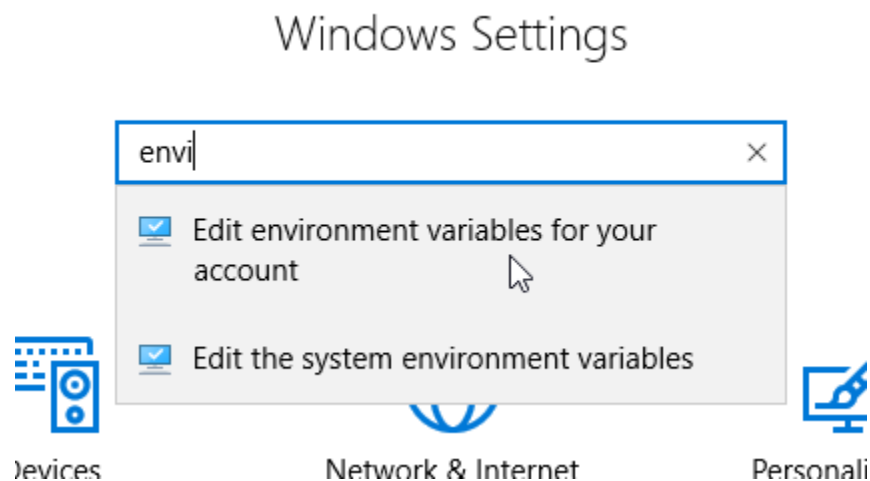
5. You might notice that it's a little slow to open the IAM file now, since it also starts Excel to read the spreadsheet. There is an alternative mechanism that is faster; however it has certain limitations: (a) It can only read cells with data in them – it doesn't execute any formulas. (b) It doesn't provide access to the Excel COM interface, so you can only access the spreadsheet using the iLogic GoExcel functions. (c) It applies to the whole Inventor session, so all other uses of spreadsheets will also be subject to these restrictions.

This behavior is controlled by an "environment variable". To change it:

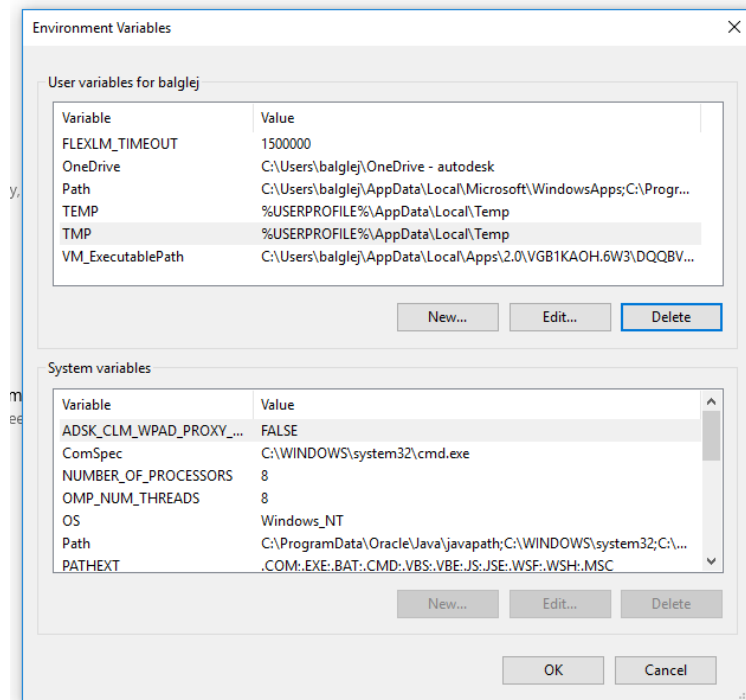
- a. From "Start" button, click on "Settings":



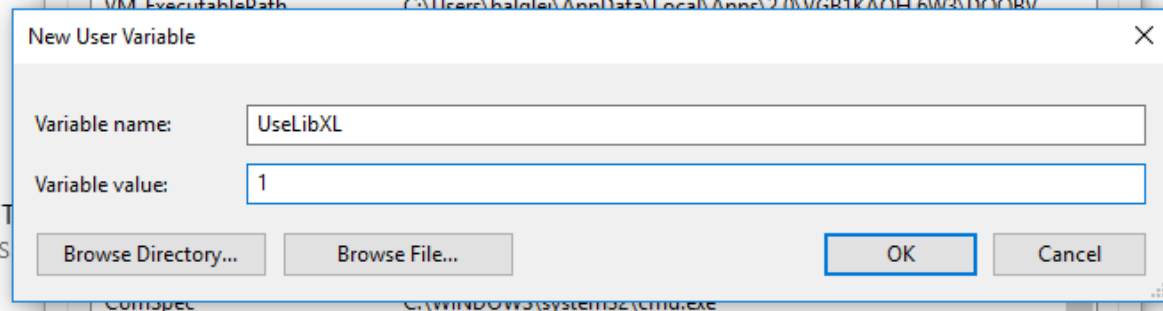
- b. In Search box, type "environment"



- c. You get the Environment Variables dialog:



- d. In the upper area, click "New...". Enter UseLibXL and "1":



- e. If necessary, restart Inventor.
 f. Open the IAM. Faster now?
 g. Note that this is the default behavior when using **Forge Design Automation API for Inventor** (Pubic Beta was announced Monday, 11-Nov-2018) and also when using **Configurator 360**.

6. The next step is to apply the settings (parameter values) from a given configuration.
 - a. **Start a new rule** named “ApplyStandardConfig”
 - b. Content of the rule should be: (**copy-and-paste**, watch out for line-wrapping)

```

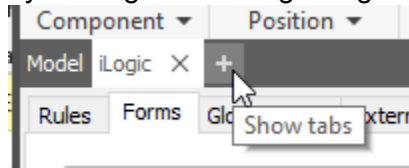
If (ConfigurationName = "Custom") Then
    Logger.Debug("Custom settings -- no changes applied.")
    IsCustom = True
Else
    Dim xl_file As String = System.IO.Path.Combine(ThisDoc.Path,
"standardSizes.xlsx")
    Logger.Trace("Excel file is: " & xl_file)
    GoExcel.FindRowStart = 2
    i = GoExcel.FindRow(xl_file, "Sheet1", "Name", "=", ConfigurationName)
'case-sensitive
    If (i < 0) Then
        MessageBox.Show("Standard configuration named '" &
ConfigurationName & "' was not found.")
    Else
        IsCustom = False
        Dim yesno As String = GoExcel.CellValue("G" & i)
        hasCover = (String.Compare(yesno, "Yes", True) = 0)
        Dim vals = GoExcel.CellValues("B" & i, "F" & i)
        length = vals(0)
        width = vals(1)
        height = vals(2)
        coverHeight = vals(3)
        thickness = vals(4)
    End If
End If

```

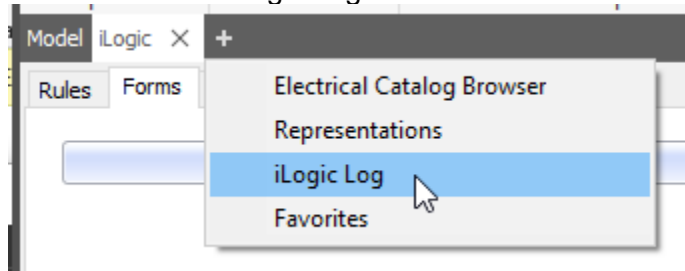
- c. Let's review what this code does, more-or-less line-by-line:
 - i. **If (ConfigurationName = "Custom")** ... In this scenario, we leave all the other parameters in their current state, except we set IsCustom to true. Later, we'll use this parameter in a Form.
 - ii. **Logger.Debug** ... print out this message to the log window, when log level is at Trace or Debug level.
 - iii. **xl_file** ... same trick as earlier rule.
 - iv. **Logger.trace** ... hey, it's easy to get confused about which folder/project you're using; this statement helps us be sure we're using the correct copy. Note the message only appears at the lower “trace” level; it doesn't appear if you're only at “debug” level.
 - v. **GoExcel.FindRowStart** ... we're going to use “FindRow” on the next line. This tells FindRow to skip the “header” row.
 - vi. **GoExcel.FindRow** ... this iLogic method allows you to specify some search criteria, and then it will find the first matching row. Return value is the row number, or -1 if not found.

- vii. `GoExcel.CellValue` ... takes a cell reference (using the “active” spreadsheet file and sheet as set by `FindRow`), and returns the value. In this case, the value is a string.
Note you can also use `GoExcel.CurrentRowValue(“ColumnName”)`, but using `CellValue()` will reset the current row, so probably best not to mix the two techniques.
 - viii. `hasCover = (String.Compare ... String.Compare` returns -1, 0, or 1 depending on whether the string is less than, equal to, or greater than the other string, in alphabetical order. So “= 0” means they’re equal, and the whole parenthetical expression returns `True` or `False`, which can then be assigned to `hasCover`. The third argument to `String.Compare` says whether to “ignore case”. `True` means ignore case, `False` means case-sensitive.
 - ix. `GoExcel.CellValues()` ... (with an “s”) gets the values of a range of cells. The cells should all have the same type (all numbers or all strings). Returns an array.
 - x. Remaining lines just pull the values out of the array, and assign to the appropriate parameter.
- d. **Save & Run** the rule. You may see some changes to the model based on your previous state.
 - e. **Open the Parameters dialog**, and **change the value of “ConfigurationName”** (something other than “Custom”). As you change it, you get different sets of values applied to the model.

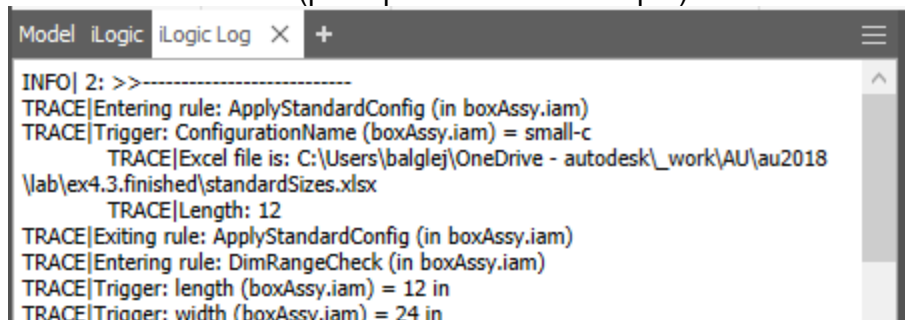
7. (Optional – skip to #8 if short on time) iLogic “Logging”. We’ve already seen some places where the rule invokes “Logger.Trace” or “Logger.Debug”.
- a. **iLogic Log Window.** If you haven’t already figured it out, you can see the output by looking at the “iLogic Log” window. Click the “+” next to the “iLogic” tab:



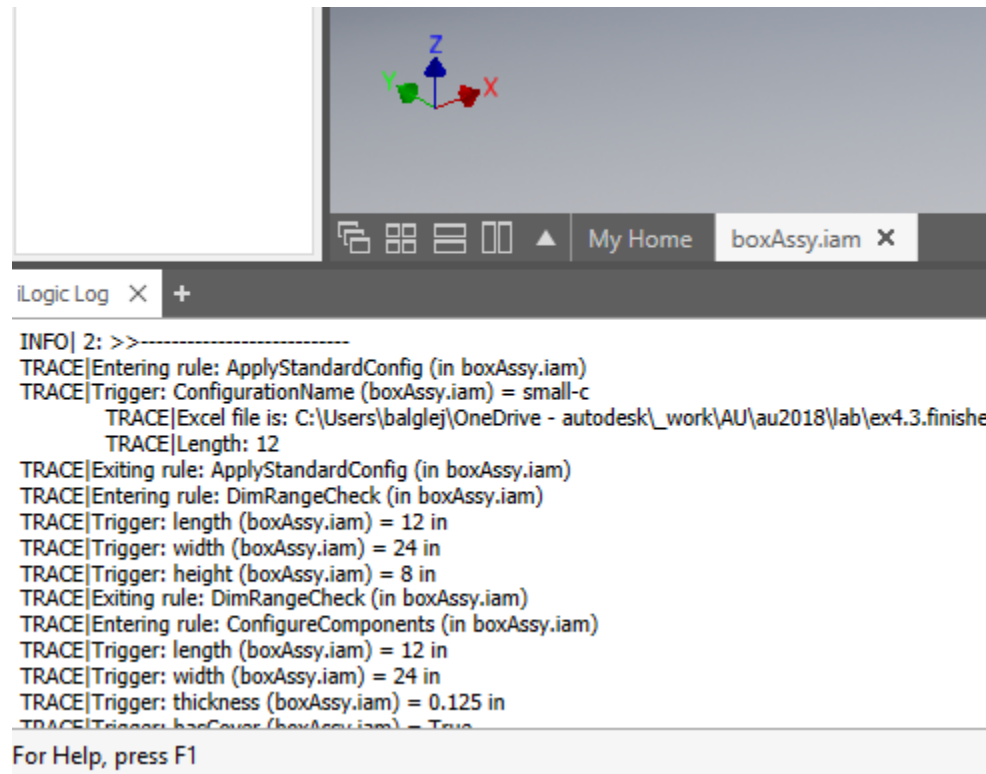
And then select “iLogic Log”



You’ll see the window (perhaps with different output):



You can drag the iLogic window to “float” it, or you can dock it against an edge of the Inventor window. Here it is at the bottom:



- b. Log output is always performed at a certain priority level. “Trace” is the lowest level. “Fatal” is the highest level. You can set the “log level” to control what output is generated. Log output is only performed/generated when the “log level” is at or “above” (higher priority) the requested type.

You can set the log level using the drop-down menu at the bottom of the Rule Editor.

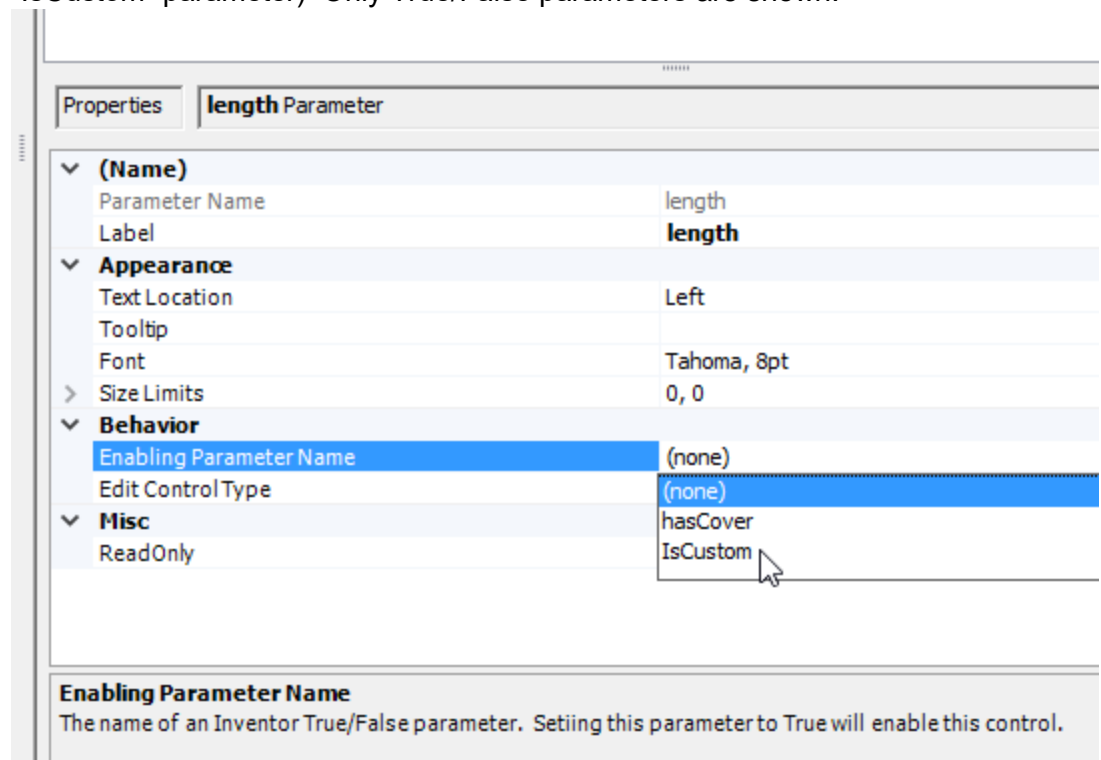


The default log level is “Info”. “Debug” and “Trace” are lower-priority. So if your rules do “Logger.Debug” or “Logger.Trace”, you ... or another user ... will only see those messages if you go out of your way to change the log-level. “Logger.Debug” should be used for most debugging “print statements”. “Logger.Trace” should be used when you have extremely detailed information that you don’t need all the time. But there’s no real restriction here.

The ApplyStandardConfig rule already has Logger.Debug and Logger.Trace statements.

- c. Rule Tracing. At the “Trace” log-level, there is an option (default=on), to get “Detailed Trace” messages from iLogic. iLogic will automatically generate trace output when it enters and exits a rule, and also what parameters triggered the rule (if any). You can use this information to debug certain types of problems. It might also show you when your rules are being executed multiple times (e.g., because a rule is changing some parameters, which triggers other rules that change parameters, etc.)
 - d. Set the trace level to “Debug” and use the Parameters dialog to change the ConfigurationName parameter to any non-custom value. See what output is generated. You can right-click on the iLogic Log window to clear the contents.
 - e. Now set the trace level to “Trace” (Detailed Trace on). Change the ConfigurationName parameter again. See what output is generated.
 - f. Now set the trace level to “Info”. Change the ConfigurationName parameter again. See what output is generated.
8. Add a “Configuration Name” control to the “Configure” form.
- a. Go to the “Forms” tab. Click right on the form-button, select “Edit”
 - b. Drag the “ConfigurationName” parameter to the form-layout. Change the name to “Std Config”.
 - c. Click OK.
 - d. Test the form:
 - i. Change the configuration name to anything except “Custom”. You should see the model update, as before.
 - ii. Try changing any of the dimensional parameters. It should automatically force the parameter back to the “standard” value.
 - iii. Change the configuration to “Custom”.
 - iv. Now change a dimensional parameter. You should see the change take effect.

- e. OK, that works. But really, it would be best if the dimensional parameters were only enabled when the user selects “Custom”:
- Edit the form.
 - Select the “length” parameter to see its properties
 - Change the “Enabling Parameter Name” (that’s why we have this “IsCustom” parameter) Only True/False parameters are shown.

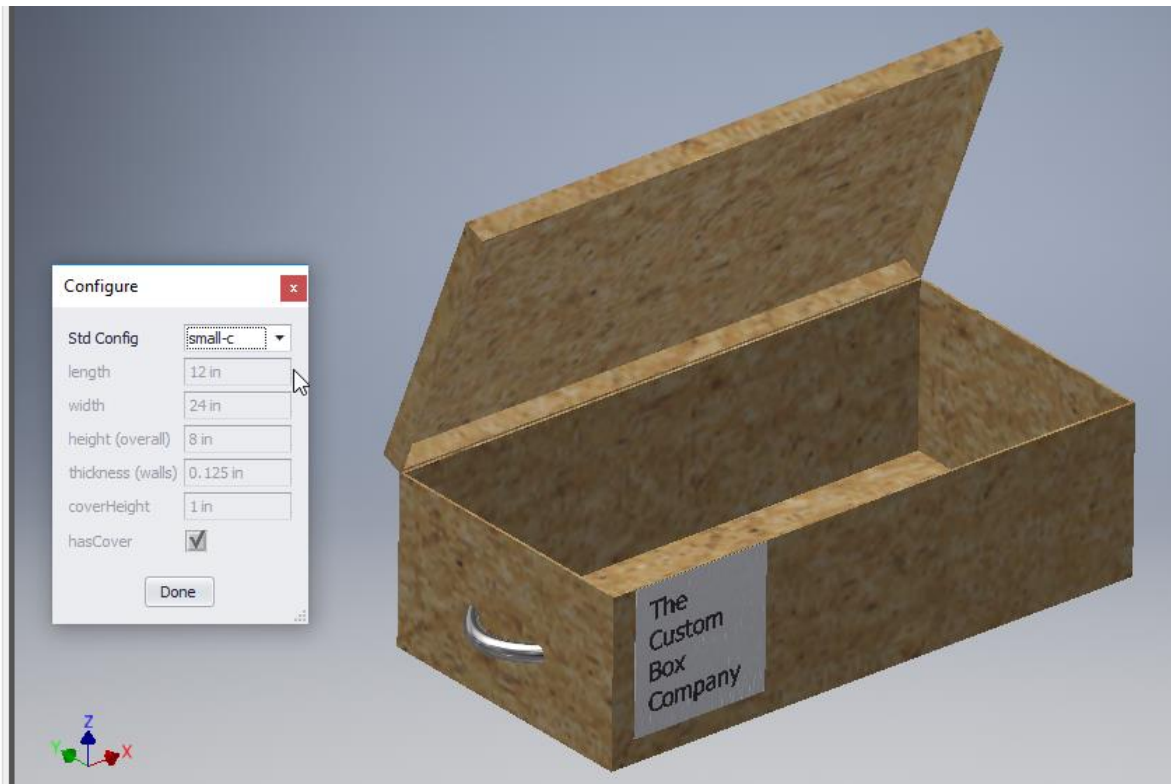


length Parameter	
(Name)	
Parameter Name	length
Label	length
Appearance	
Text Location	Left
Tooltip	
Font	Tahoma, 8pt
Size Limits	0, 0
Behavior	
Enabling Parameter Name	(none)
Edit Control Type	(none)
Misc	
ReadOnly	IsCustom

Enabling Parameter Name
 The name of an Inventor True/False parameter. Setting this parameter to True will enable this control.

- Repeat for all the rest of the parameters.
- Test the form
- Save the file

Wow!!! You can pick standard configurations, or choose custom and change sizes!



You're done!



What experienced iLogic users know

1. They would say: Ask yourself: What do you want to do?
 - a. *Someone has probably done it before.* Avoid writing it from scratch: Search with “site:autodesk.com” ... Customization forum. YouTube. Autodesk University. Guided tutorials in Inventor.
 - b. Have a “spec” for what you want to do.
 - c. Other Inventor capabilities have their place; don’t use iLogic gratuitously. (iParts, addins, skeleton modeling, etc.)
 - d. Other Inventor capabilities have limits and/or require human interaction where it’s not necessary; that’s where iLogic shines. E.g., moderately complex assembly configuration (iAssemblies are hard to use; many standard configurations don’t really require much thought ... iLogic to the rescue).
2. Document Updates.
 - a. If the model isn’t updated after your rule completes, there are iLogic snippets for doing an update. Best is probably “iLogicVb.UpdateWhenDone = True”. You can put this anywhere in your rule and iLogic will update the model when done. But indiscriminate use of this and similar snippets will cause lots of unnecessary updates, slowing down the whole process. Most Inventor commands will perform an update if needed, so why does your rule need to do it also? If rules are triggered from Inventor commands, the command has an update, and no extra update is required. But if just running rule, then rule must ask for update. But not necessarily every dependent/triggered rule.
 - b. Intra-rule update. iLogic normally waits until the end of the rule to “really” update the parameters with their values. During the processing of the rule, the code in the rule can refer to temporary things (local variables) with the same name as the parameters. This is much faster. But if you do something within the rule that accesses the parameters indirectly (e.g., via the API or by computing the volume or other dependent geometry), you may need to “push” the parameter values out to the “real” Inventor model. Use “RuleParametersOutput()”.
3. Using VB.Net (e.g., extra Sub’s and Function’s, try/catch, etc.) and .Net libraries (e.g., System.IO). Keep your code modular. Don’t Repeat Yourself (the D.R.Y. principle). Take advantage of the best tool for the job.
4. Copy Design. Know your workflow w.r.t. copying files or not. I.e., should un-modified parts be copied? When are parts copied? What file names (naming convention) to use.
5. Difference between a parameter reference (triggers) and Parameter(“foo”) (doesn’t trigger, but is dynamic -- string can change, e.g., in a loop). Case-sensitivity of parameter names. Weird names (“return”, “case”). Non-triggering. File must be open.
6. Units, especially when mixing API and iLogic. API centimeters, iLogic is doc-units.

7. SharedVariables (global data accessible to all files, but only within Inventor session)
8. OK to dive into API when snippets not sufficient. Discerning between iLogic objects and their properties, vs. API objects and their properties. (especially important if “units” come into play)
9. Drawings need API usage. Drawings are hard, unless merely poking in strings / changing parameters.
10. Cloud options:
 - a. Configurator 360 – Configure Inventor models in a browser, using iLogic forms
 - b. “Forge Design Automation API for Inventor” (just entering “Public Beta”). Use the Inventor API (and iLogic, of course) totally programmatically.

Additional Resources

1. Inventor [Customization Forum](#)
Note you can just use Google to search for any relevant phrase, and add "site:autodesk.com" to limit the results.
2. YouTube ... search for "iLogic" (some great videos from Autodesk partners/resellers)
3. Autodesk University previous years
4. Inventor Guided Tutorials
5. [Inventor documentation](#) There is a whole section dedicated to iLogic.