MFG502967

iLogic, Your Foundation for CPQ

Don Boresky
SolidCAD
don.boresky@solidcad.ca

Shannon Lundrigan
SolidCAD
shannon.lundrigan@solidcad.ca

Learning Objectives

- Understand what iLogic is within Inventor
- Structure models to enable full product configuration
- Learn how iLogic can directly drive product pricing
- Extend Inventor Models to the WEB for non-technical users

Description

iLogic provides powerful, but underutilized, capabilities for Inventor. In a recent poll of Autodesk users, almost 50% had never heard of iLogic, and only about 30% are actively using it. Many users that are aware of iLogic, limit its use to part design and small utilities. However, that is just scratching the surface of what iLogic can do. In this session, we will introduce how iLogic can be used to configure an entire product, including supporting drawings, documentation, and pricing. We will also show how the configurable model can be made available to non-Inventor users, such as sales reps or customers, to create their own configurations, freeing up time for Engineering to focus on value-add activities.

Speaker(s)

Don Boresky has been working with Manufacturers for over 30 years. With a Bachelor of Science – Computer Science, Don's entire career has been focused on the use of technology to help organizations integrate systems, streamline workflows, and improve business outcomes. In his role with SolidCAD, Don helps manufacturers of configurable products to reduce lead times and errors by streamlining the quote-to-production process using SolidCADs Variant Configurator.

Shannon has a Bachelor of Applied Science in Mechanical Engineering from the University of Ottawa, and is a certified P.Eng registered with AEPGA in Alberta. With over 10 years of industry experience in design and manufacturing, she has developed many custom products for mission critical applications, some of which were granted patents in the US and Canada. Shannon has extensive experience working directly with clients to meet unique requirements and provide technical support throughout long-term & high-profile projects.

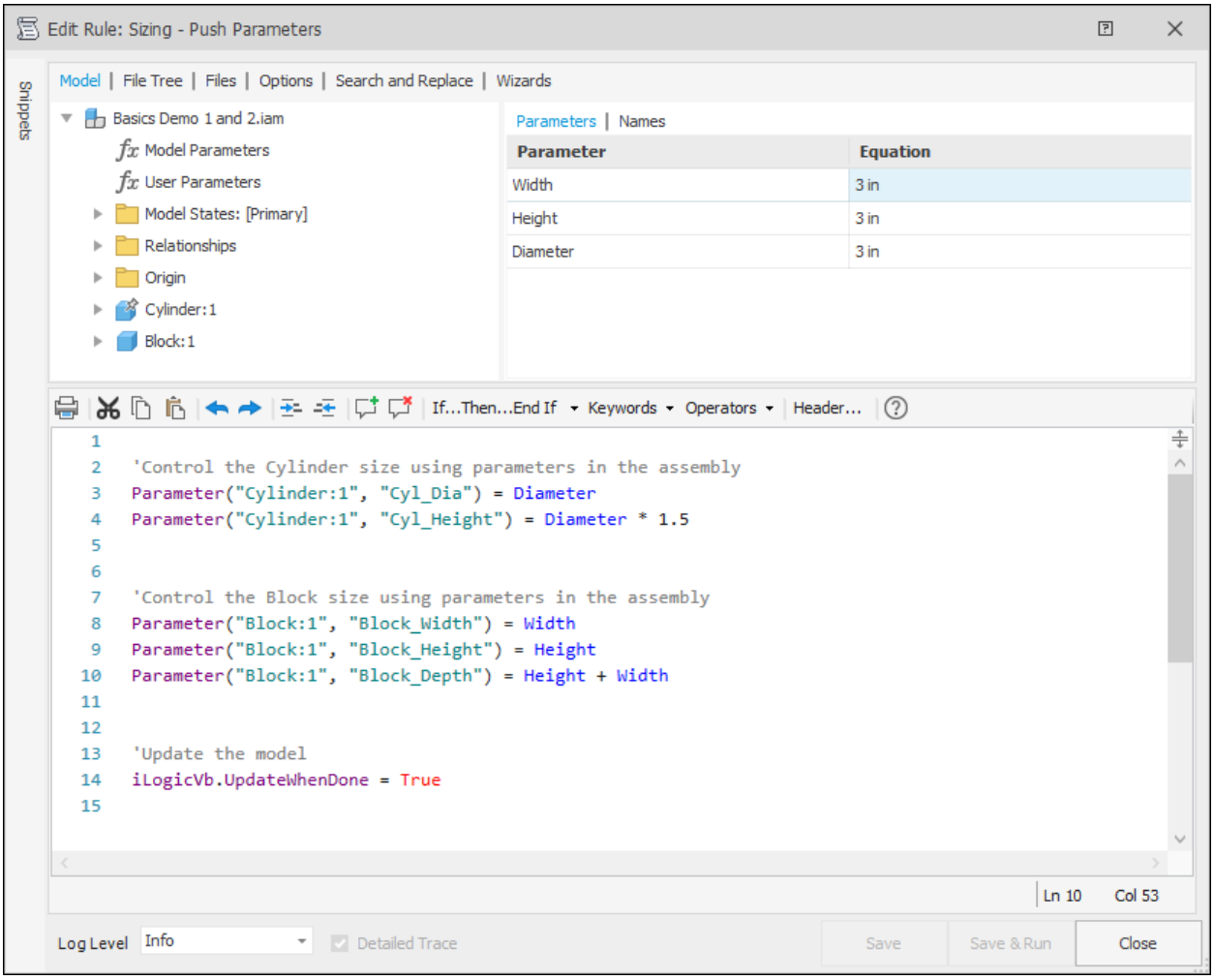
At SolidCAD, Shannon is a Technical Consultant on the Manufacturing Team providing pre- and post-sale consulting services, software implementation, training, and support for our clients.

iLogic Basics

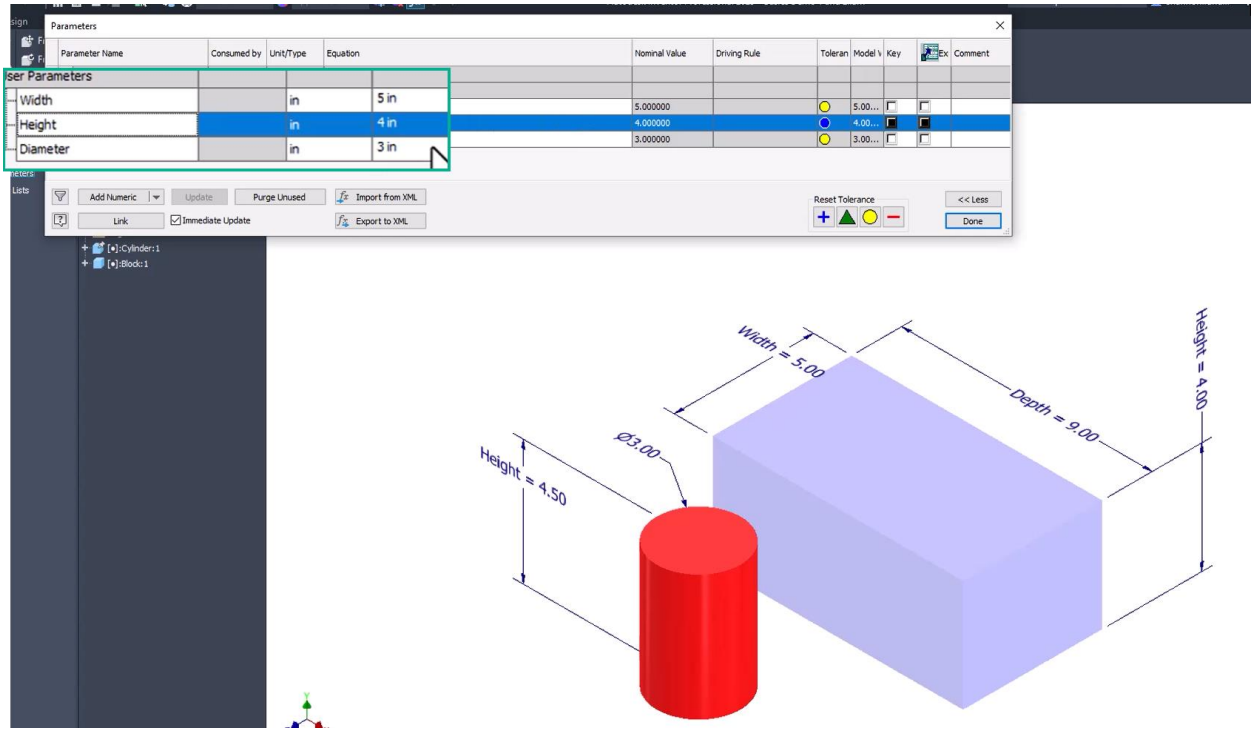
Using parameters to manipulate your designs is at the core of 3D modeling. Using iLogic to add intelligence and eliminate extra steps when changing parameters is just scratching the surface of what iLogic can do, but these fundamentals are essential to kick start your iLogic journey.

Push Parameters

Pushing the parameter values from an assembly into individual part files is an efficient way to manage configurations. This simple rule in the figure below will control both the Cylinder and the Block parts of my assembly using the user parameters inside my assembly file (Width, Height, and Diameter). Using the Parameter function you first call the part instance (Cylinder:1 and Block:1, respectively) and then name the parameter from the part file you want to control (Cyl_Dia, Cyl_Height etc.). Once the parameter has been identified, you control it using a parameter within the assembly (ie. Diameter) or a function or equation. Whatever is necessary to get the correct value.



Push Parameters Code Example

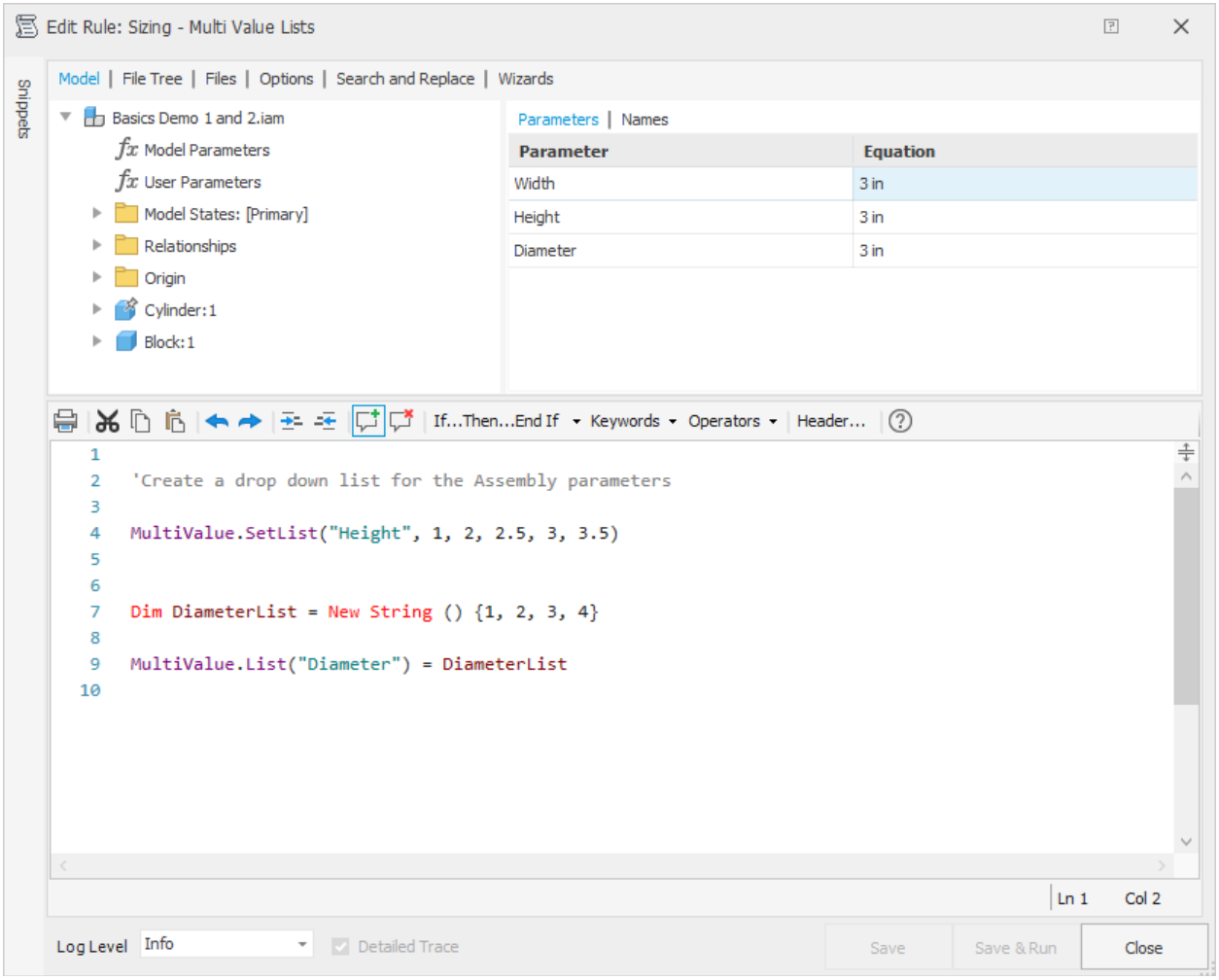


Push Parameters Example

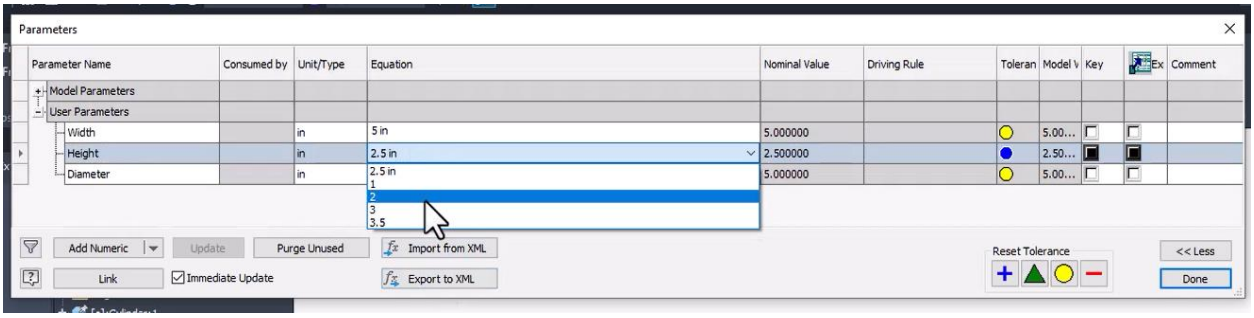
Changing the assembly parameter values now adjusts the parts accordingly. Since this is a direct parameter change, the rule runs automatically.

Multi-Value Drop Down Lists

When you require a finite list of options for a configuration assembly, a multi-value drop down list is typically the best solution. These can easily be created using iLogic as seen below. In this rule, we set the list for height directly as 1, 2, 2.5, 3 and 3.5. Another option is to declare the list as a variable so it can be easily referenced in multiple locations. Here we see “DiameterList” as the variable, setting the assembly diameter to 1, 2, 3, and 4.



Multi-Value Drop Down Code Example



Drop Down List for The Height Parameter

Vent-A-Hood re-used many lists in their code specifically for their material finishes. Depending on the selections the user made, the rule set the finish lists accordingly and each list can be

used in as many places as necessary but can be easily edited in a single location in the rule, which is a best practice for reducing errors and typos.

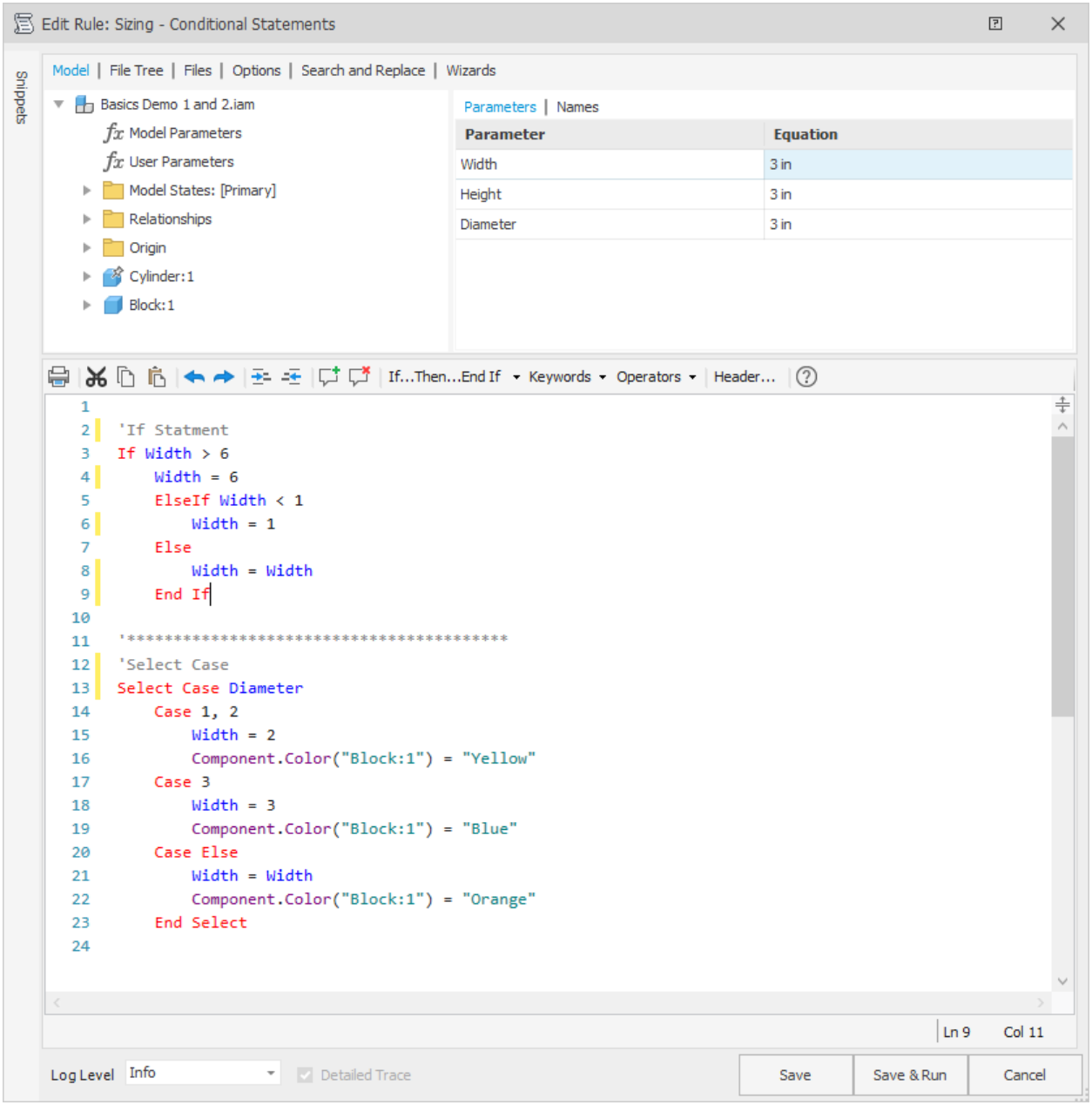
```
Dim hoodFinishes = New String() {"Stainless Steel", "Biscuit", "Black", "Black Carbide",
                                 "Gunsmoke", "White", "Black River", "Copper Vein",
                                 "Oil Rubbed Bronze", "Pewter", "Rust", "Silver Vein",
                                 "Slate", "Antique Brass", "Antique Copper",
                                 "Antique Hammered Copper", "Artisan Stainless Steel",
                                 "Hammered Copper", "Real Brass", "Brushed Brass", "Satin Brass",
                                 "Real Copper" }
Dim standardBandFinishes = New String() {"Mirror Stainless Steel", "Stainless Steel", "Biscuit", "Black",
                                     "Black Carbide", "Gunsmoke", "White", "Black River", "Copper Vein",
                                     "Oil Rubbed Bronze", "Pewter", "Rust", "Silver Vein", "Slate", "Antique Brass",
                                     "Antique Copper", "Antique Hammered Copper", "Artisan Stainless Steel",
                                     "Hammered Copper", "Real Brass", "Brushed Brass", "Satin Brass", "Real Copper" }
Dim standardLTFinishes = New String() {"Mirror Stainless Steel", "Stainless Steel", "Biscuit", "Black", "Black Carbide", "Gunsmoke",
                                     "White", "Black River", "Copper Vein", "Oil Rubbed Bronze", "Pewter", "Rust", "Silver Vein",
                                     "Slate", "Antique Brass", "Antique Copper", "Antique Hammered Copper",
                                     "Artisan Stainless Steel", "Hammered Copper", "Real Brass", "Brushed Brass",
                                     "Satin Brass", "Real Copper" }
Dim mediumLTFinishes = New String() {"Stainless Steel", "Antique Brass", "Antique Copper", "Real Brass", "Real Copper"}
Dim heavyLTFinishes = New String() {"Stainless Steel", "Antique Brass", "Antique Copper", "Real Brass", "Real Copper"}
Dim railFinishes = New String() {"Polished Chrome", "Polished Brass", "Satin Copper"}
```

Conditional Statements

If parameters are the core of 3D modeling, then conditional statements are the core of iLogic. This is where the magic takes form, and you can start to build intelligence into your models.

In the first section of the rule below, we see the parameter Width being given a maximum value of 6 and a minimum value of 1 by checking the entered value against the limit and re-setting it accordingly if necessary.

The second section of the rule is the Case Select method. Once you tell the rule what parameter to look at you can set actions for each case. A case can be an option from a drop down, or simply a value (or equation). As the example shows, more than one action can be performed for a given case.



Conditional Statement Code Example

A best practice when writing conditional statements is to ensure all potential outcomes are covered (either with the “Else” or “Case Else” lines). This way your rule runs without error.

In the Vent-A-Hood models, the Select Case method is used extensively to set dependent parameters and turn features on/off. Notice in the code seen below the use of nesting conditional statements, which can further add intelligence to your configurable designs.


```
Select Top_Width
    Case 24:
        Select Ceiling_Height
            Case 9, 10:
                DC_Top_Width = 15
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case 30:
        Select Ceiling_Height
            Case 9, 10:
                DC_Top_Width = 18
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case 36:
        Select Ceiling_Height
            Case 9:
                DC_Top_Width = 24
            Case 10:
                DC_Top_Width = 18
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case 42:
        Select Ceiling_Height
            Case 9:
                DC_Top_Width = 30
            Case 10:
                DC_Top_Width = 24
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case 48:
        Select Ceiling_Height
            Case 9:
                DC_Top_Width = 36
            Case 10:
                DC_Top_Width = 30
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case 54:
        Select Ceiling_Height
            Case 9:
                DC_Top_Width = 42
            Case 10:
                DC_Top_Width = 36
            Case Else
                DC_Top_Width = Top_Width - 6
        End Select
    Case Else
        DC_Top_Width = Top_Width - 6
End Select

'-----------------------------------------------------------------------------------------------
'   CONFIGURE THE PARAMETER TO DRAWING LINKS
```

Vent-A-Hood Select Case Example

Assembly Control

In order to properly configure an assembly, one thing you will surely need to control is what parts are in each configuration. There are a few different methods that can be employed, each of them has their benefits.

Visibility

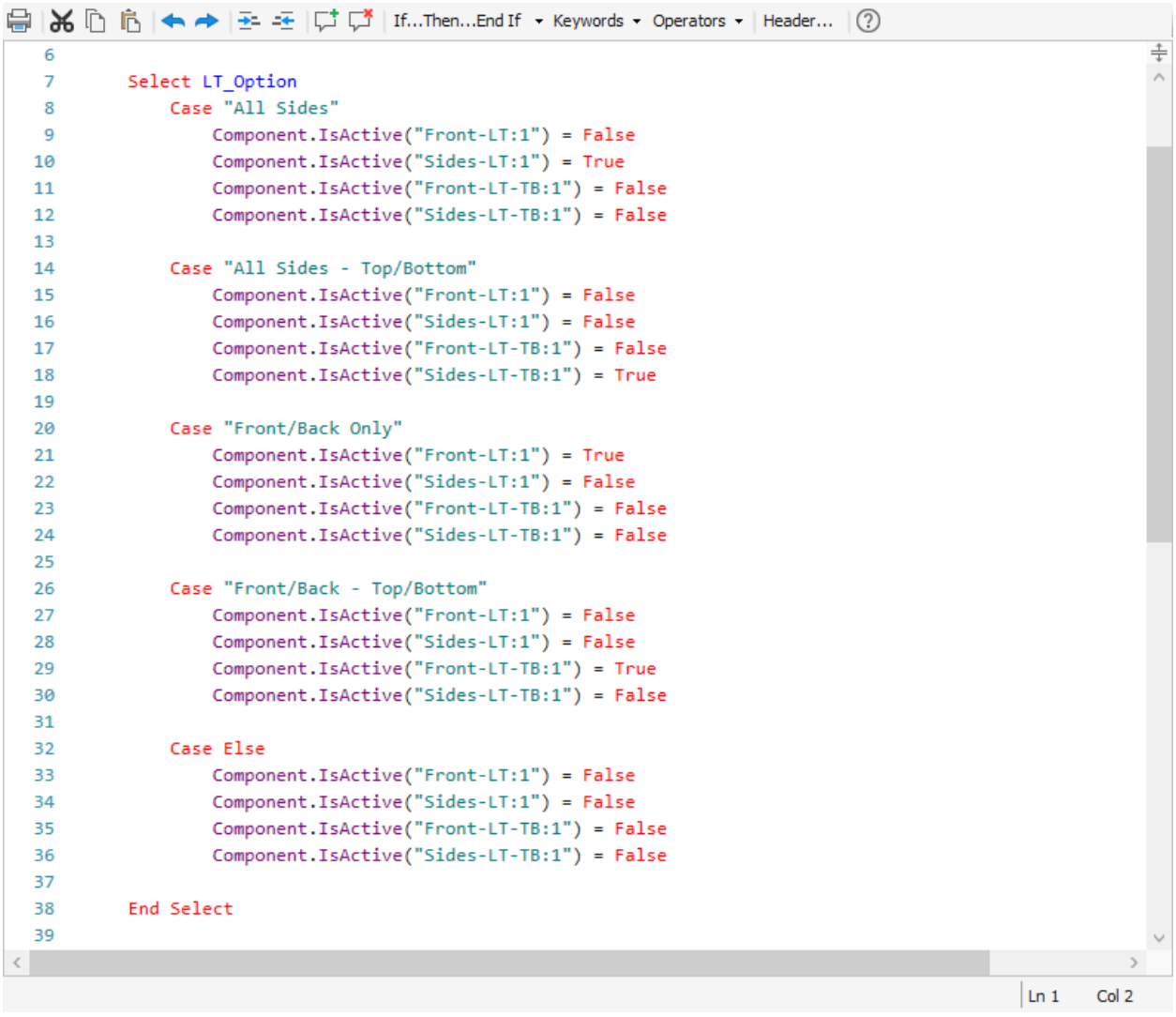
Using conditional statements we can turn the visibility of certain parts on and off. In the example below, using the Boolean function “Component.Visible” we can select any part occurrence to be visible or not visible, depending on the value selected for the parameter “LT_Option”.
This function is very quick when run in the model however, the parts will remain in the BOM as “normal” and their quantity will not change to reflect their visibility. Therefore, if an accurate BOM is required for each configuration then you would either need to set the BOM structure explicitly or use another method.

```
6
7       Select LT_Option
8           Case "All Sides"
9               Component.Visible("Front-LT:1") = False
10              Component.Visible("Sides-LT:1") = True
11              Component.Visible("Front-LT-TB:1") = False
12              Component.Visible("Sides-LT-TB:1") = False
13
14          Case "All Sides - Top/Bottom"
15              Component.Visible("Front-LT:1") = False
16              Component.Visible("Sides-LT:1") = False
17              Component.Visible("Front-LT-TB:1") = False
18              Component.Visible("Sides-LT-TB:1") = True
19
20          Case "Front/Back Only"
21              Component.Visible("Front-LT:1") = True
22              Component.Visible("Sides-LT:1") = False
23              Component.Visible("Front-LT-TB:1") = False
24              Component.Visible("Sides-LT-TB:1") = False
25
26          Case "Front/Back - Top/Bottom"
27              Component.Visible("Front-LT:1") = False
28              Component.Visible("Sides-LT:1") = False
29              Component.Visible("Front-LT-TB:1") = True
30              Component.Visible("Sides-LT-TB:1") = False
31
32          Case "None"
33              Component.Visible("Front-LT:1") = False
34              Component.Visible("Sides-LT:1") = False
35              Component.Visible("Front-LT-TB:1") = False
36              Component.Visible("Sides-LT-TB:1") = False
37
38      End Select
39
                                                    Ln 1    Col 2
```

Visibility Control Example

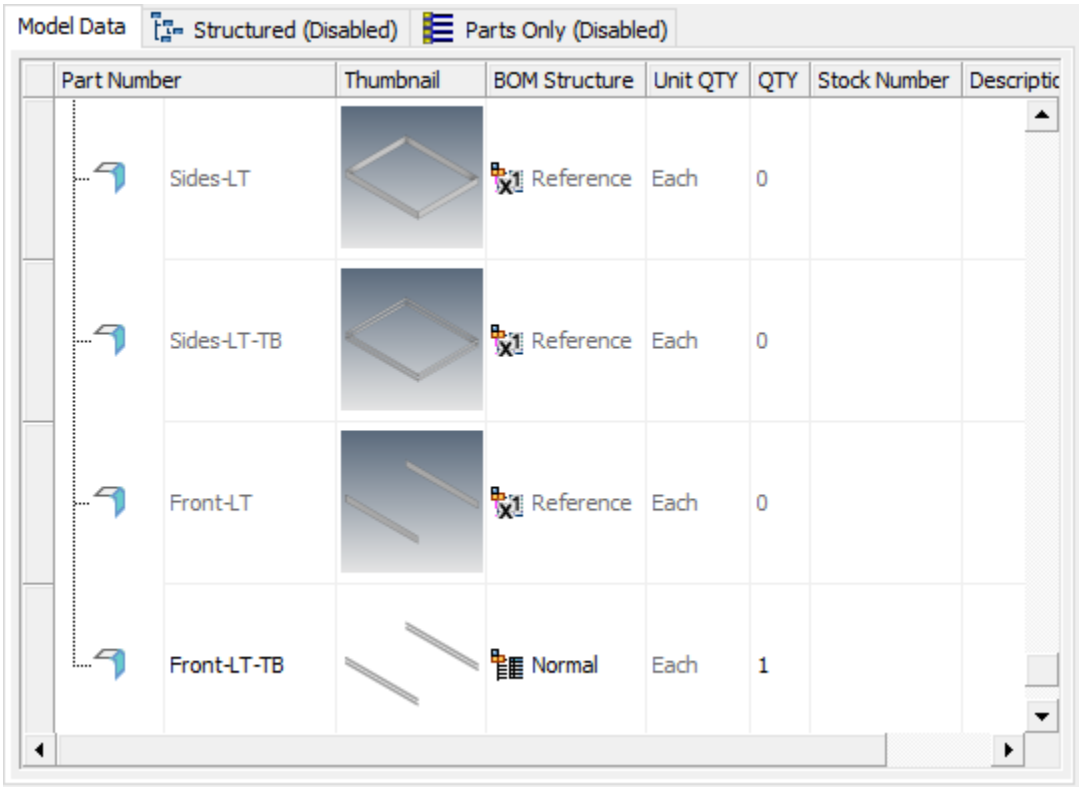
Suppression

Using the function “Component.IsActive” can control the suppression state of each part. Like the visibility method, this is a Boolean statement, using simply true or false to set the suppression. This method is roughly as quick as the visibility method discussed above however, now the BOM will reflect the configuration more accurately with the suppressed parts having their BOM structure set to Reference an the quantity set to 0.




```
Select LT_Option
    Case "All Sides"
        Component.IsActive("Front-LT:1") = False
        Component.IsActive("Sides-LT:1") = True
        Component.IsActive("Front-LT-TB:1") = False
        Component.IsActive("Sides-LT-TB:1") = False

    Case "All Sides - Top/Bottom"
        Component.IsActive("Front-LT:1") = False
        Component.IsActive("Sides-LT:1") = False
        Component.IsActive("Front-LT-TB:1") = False
        Component.IsActive("Sides-LT-TB:1") = True

    Case "Front/Back Only"
        Component.IsActive("Front-LT:1") = True
        Component.IsActive("Sides-LT:1") = False
        Component.IsActive("Front-LT-TB:1") = False
        Component.IsActive("Sides-LT-TB:1") = False

    Case "Front/Back - Top/Bottom"
        Component.IsActive("Front-LT:1") = False
        Component.IsActive("Sides-LT:1") = False
        Component.IsActive("Front-LT-TB:1") = True
        Component.IsActive("Sides-LT-TB:1") = False

    Case Else
        Component.IsActive("Front-LT:1") = False
        Component.IsActive("Sides-LT:1") = False
        Component.IsActive("Front-LT-TB:1") = False
        Component.IsActive("Sides-LT-TB:1") = False

End Select
```

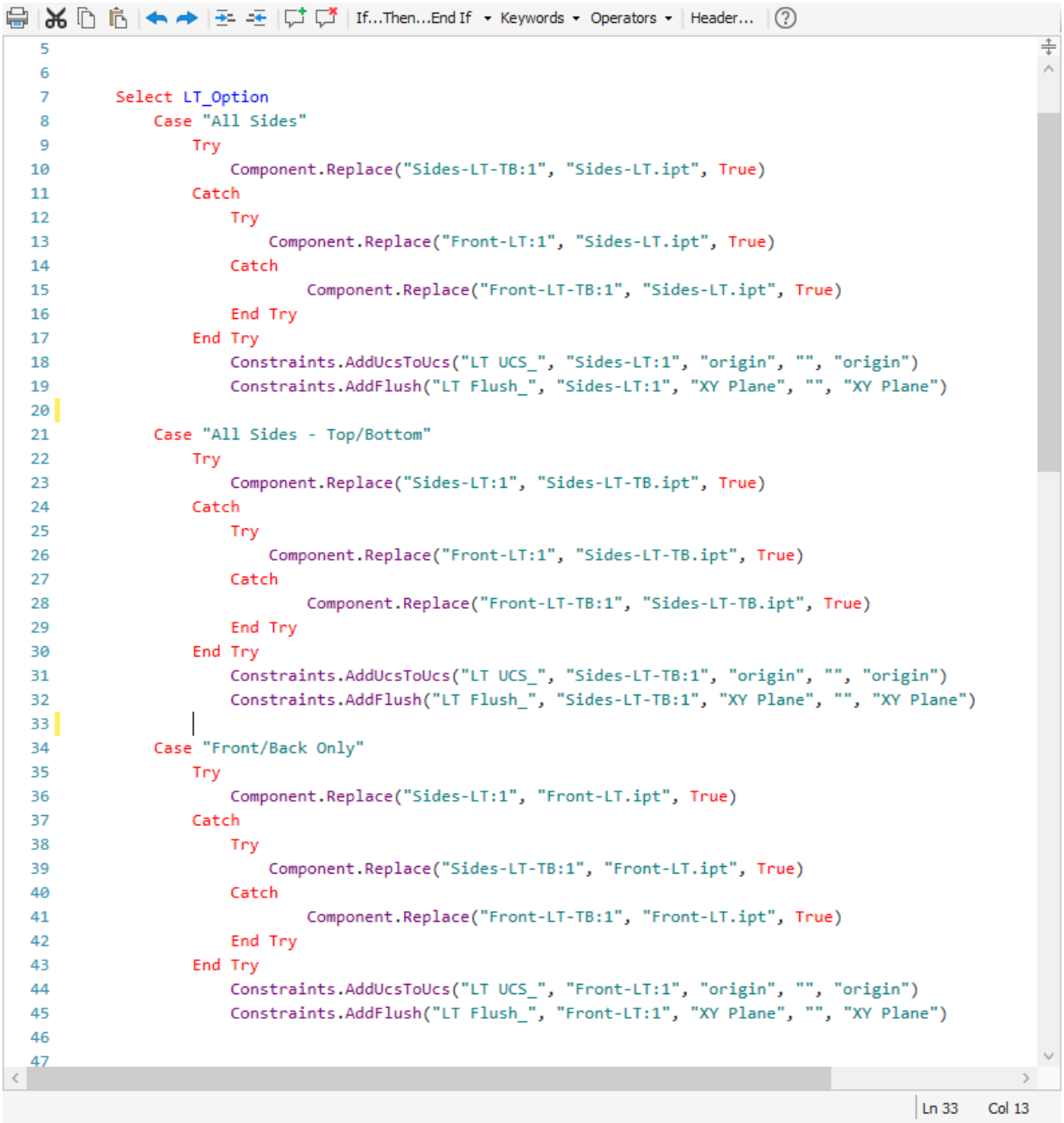
Suppression Control Example



Suppression Control Example BOM

Replace

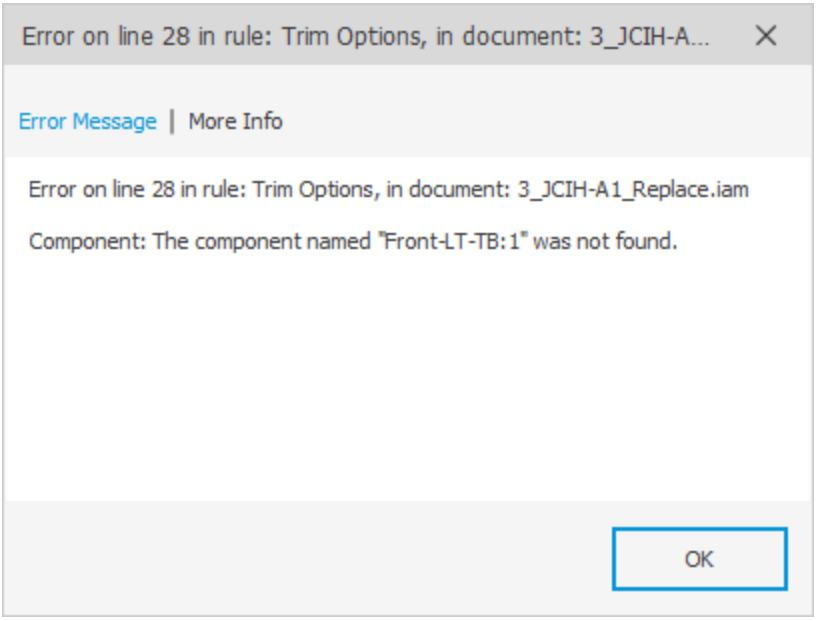
The “Component.Replace” function can be useful in certain scenarios and will keep your model browser “cleaner” in the sense that only the parts being used in the current configuration will appear. This can be useful for larger assembly files. In the example below we see partial code for the LT_Option again. For each available option, the code needs to look for what part is currently in the model, and then replace that part with the correct one. The “Try, Catch” lines allow the rule to look for each one without throwing an error if the part does not exist currently in the model.




```
    Select LT_Option
        Case "All Sides"
            Try
                Component.Replace("Sides-LT-TB:1", "Sides-LT.ipt", True)
            Catch
                Try
                    Component.Replace("Front-LT:1", "Sides-LT.ipt", True)
                Catch
                        Component.Replace("Front-LT-TB:1", "Sides-LT.ipt", True)
                End Try
            End Try
                Constraints.AddUcsToUcs("LT UCS_", "Sides-LT:1", "origin", "", "origin")
                Constraints.AddFlush("LT Flush_", "Sides-LT:1", "XY Plane", "", "XY Plane")

        Case "All Sides - Top/Bottom"
            Try
                Component.Replace("Sides-LT:1", "Sides-LT-TB.ipt", True)
            Catch
                Try
                    Component.Replace("Front-LT:1", "Sides-LT-TB.ipt", True)
                Catch
                        Component.Replace("Front-LT-TB:1", "Sides-LT-TB.ipt", True)
                End Try
            End Try
                Constraints.AddUcsToUcs("LT UCS_", "Sides-LT-TB:1", "origin", "", "origin")
                Constraints.AddFlush("LT Flush_", "Sides-LT-TB:1", "XY Plane", "", "XY Plane")

        Case "Front/Back Only"
            Try
                Component.Replace("Sides-LT:1", "Front-LT.ipt", True)
            Catch
                Try
                    Component.Replace("Sides-LT-TB:1", "Front-LT.ipt", True)
                Catch
                        Component.Replace("Front-LT-TB:1", "Front-LT.ipt", True)
                End Try
            End Try
                Constraints.AddUcsToUcs("LT UCS_", "Front-LT:1", "origin", "", "origin")
                Constraints.AddFlush("LT Flush_", "Front-LT:1", "XY Plane", "", "XY Plane")
```

Replace Component Method Example

If you have ever used the “Replace Part” command in Inventor, you know that in order to replace a part, there needs to be something there to replace. In the case of this particular Vent-A-Hood model, the user has the option to turn off the LT_Option altogether, meaning the parts are removed from the assembly, leaving nothing to replace. Although there are ways around this, it would require additional code to accommodate and for this reason it’s not the most efficient method for this model.

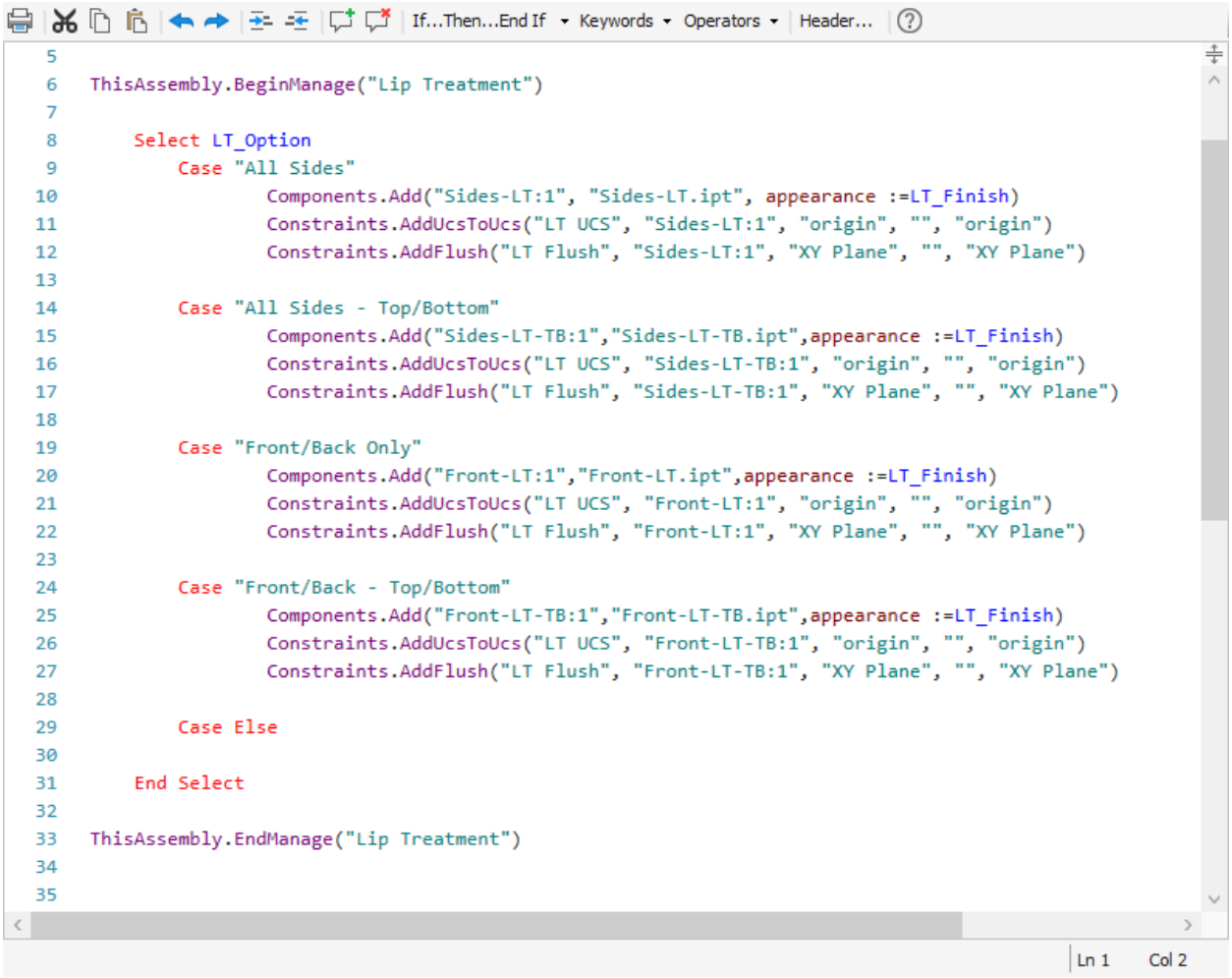


Component Not Found Error

Manage Components

The final method we will look at is what's called "Manage Components" which uses the "Component.Add" function. Here we can use "BeginManage" and "EndManage" to define a group of components we want to control. Added in 2019, this method works in a similar fashion to the replace method but is much more efficient when it comes to writing the code.

In the example below we create a managed group called "Lip Treatment". Inside the Begin and End Manage lines we can add components and their respective constraints using the Select Case method (or any conditional statement method preferred) and their respective constraints. The rule will look at all of the components within the Group and add the component for the satisfied condition if and only if it doesn't already exist, and deletes anything within the Group which currently exists in the model. This is an extremely efficient rule. You no longer need to worry about deleting or hiding components that aren't required for a particular configuration.



Manage Components Method Example

In large top-level assemblies, this is where you may begin to notice some slight changes in speed. Since the parts are not yet in the assembly file, there is a lot more going on in the background. However, since each configuration only has the parts and sub assemblies that are required, and nothing extra; you may also see a benefit in overall file size. Depending on your situation this could be a big factor in choosing your method of assembly control

Ultimately there is no right or wrong answer to how you create your configuration rules. There is always more than one way to write the code to get the same result. The most important thing is to make it robust and error proof, because you need the rules to run perfectly every time, without the need for secondary input (when we are talking in the context of web configurators)

In the case of Vent-A-Hood, they chose to toggle visibility because they didn't need the Inventor BOM output for their online configurator, they wanted to maximize the speed and have everything contained in the master model assembly file.

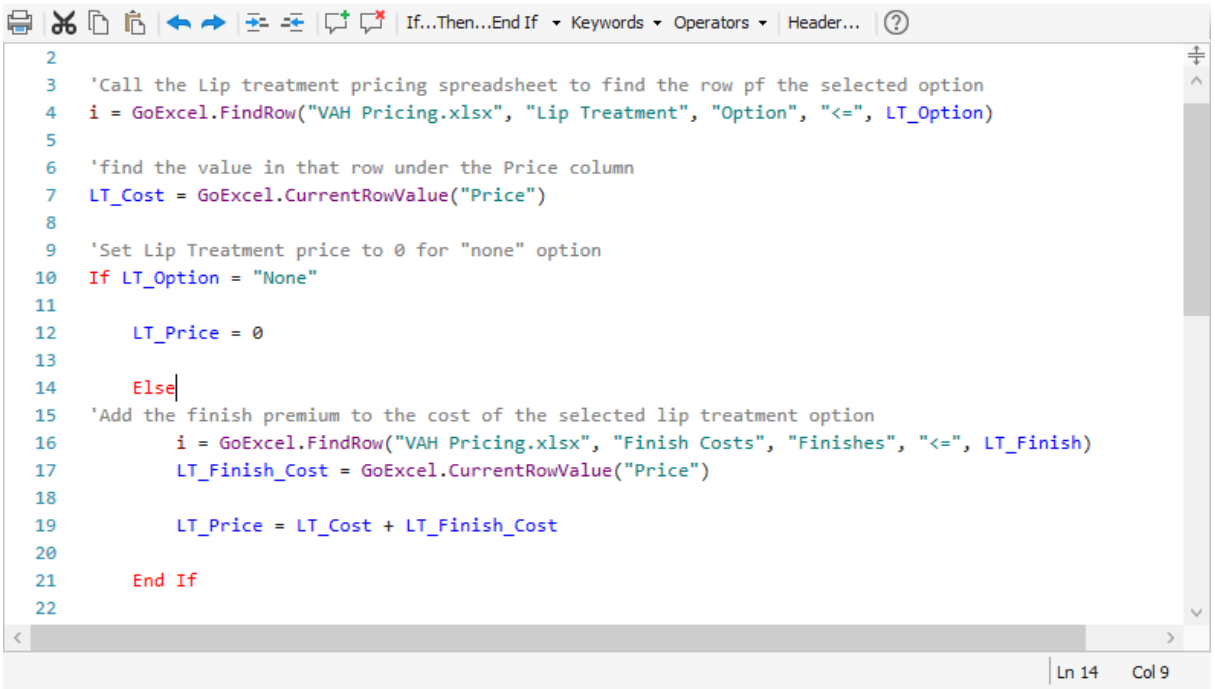
Pricing

When it comes to pricing, we typically see about 50% of our web configurator clients generate quotes directly from the site, and the other half will simply have the "request quote" option. Vent-A-Hood has a very extensive price quoting calculation built into their configuration models. In this section we will look at two options for calculating price; using external data, and calculation directly using iLogic.

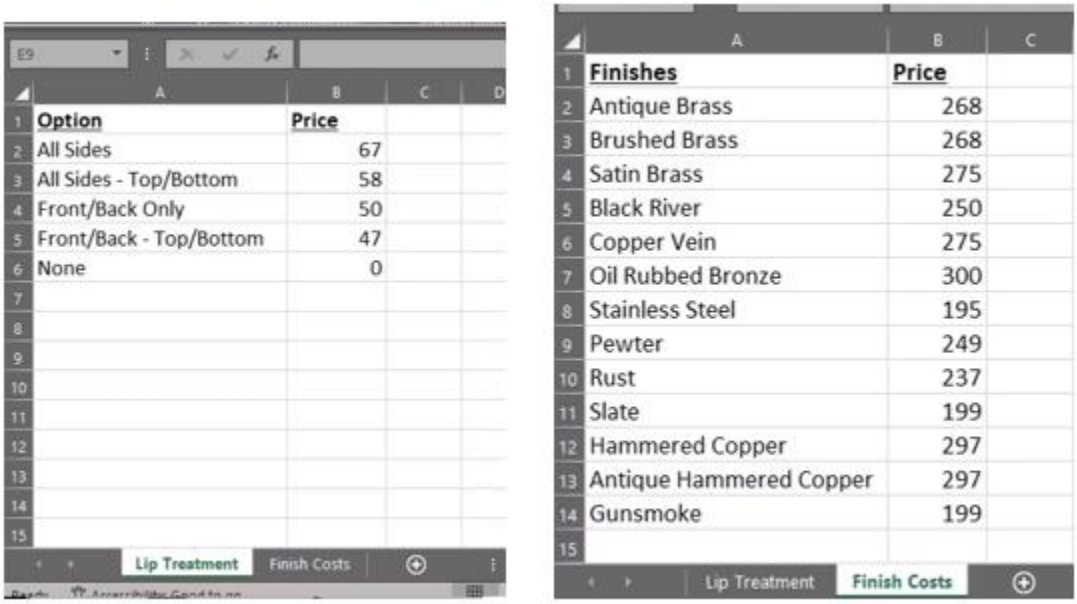
Pulling External Data

Some clients prefer to keep the price information outside of the model. One benefit of this is non-technical personnel (ie. Procurement) can edit the data without needing access to the Inventor model. The data is pulled from a spreadsheet via iLogic. The easiest is to have the spreadsheet saved in the same location as the assembly file, then there are just a few lines of code needed to pull the appropriate values.

Here we are telling the rule to look at the spreadsheet named “VAH Pricing.xlsx”, on the sheet “Lip Treatment” and then it compares the values in the column “Option” to the parameter in the assembly called “LT_Option”. This finds the row that corresponds to the select LT_Option. Next the rule looks at the “Price” column in that row to assign that value to the parameter “LT_Cost”. Likewise, we can pull the values for the finish that was selected by the user, and at the end we add the two cost values together to get our final LT_Price.



Pricing Rule Pulling External Data



	A	B	C	D
1	Option	Price		
2	All Sides	67		
3	All Sides - Top/Bottom	58		
4	Front/Back Only	50		
5	Front/Back - Top/Bottom	47		
6	None	0		
7				
8				
9				
10				
11				
12				
13				
14				
15				

Lip Treatment | Finish Costs

	A	B	C
1	Finishes	Price	
2	Antique Brass	268	
3	Brushed Brass	268	
4	Satin Brass	275	
5	Black River	250	
6	Copper Vein	275	
7	Oil Rubbed Bronze	300	
8	Stainless Steel	195	
9	Pewter	249	
10	Rust	237	
11	Slate	199	
12	Hammered Copper	297	
13	Antique Hammered Copper	297	
14	Gunsmoke	199	
15			

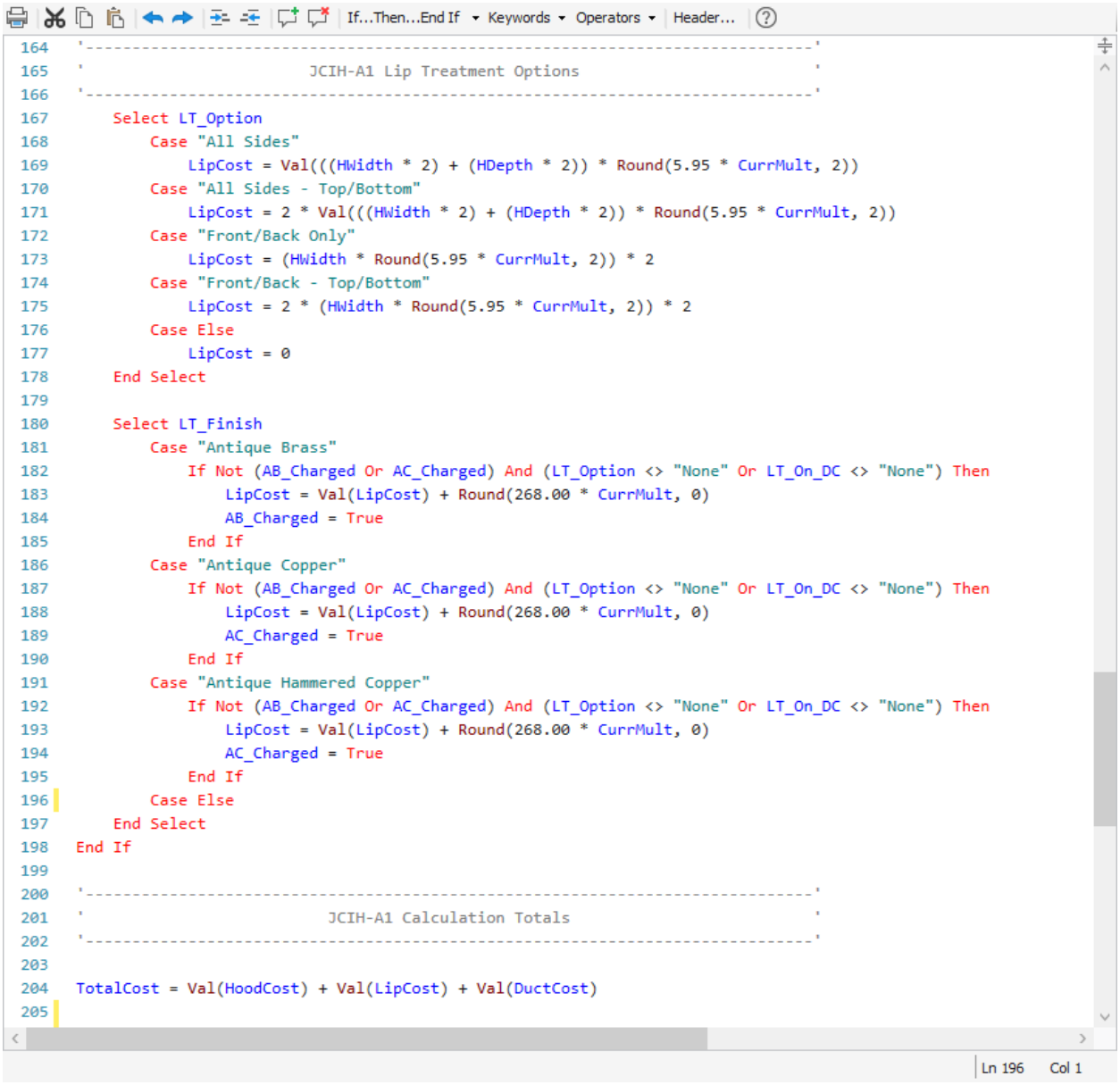
Lip Treatment | Finish Costs

Spreadsheet Data

Provided the referenced names remain the same (file name, sheet, columns), the cost data can be updated as necessary over time all without opening the 3D models.

Price Calculations using iLogic

The code below is an example of what Vent-A-Hood uses to determine their prices for the LT_Option and LT_Finish. They use a combination of If/ElseIf and Select Case conditional statements to ensure each option is accounted for and assign a value to each cost parameter accordingly. Using “Val” and “Round” functions they are able to ensure the calculated values are set to two decimal places, and the “CurrMult” parameter is a multiplier to account for different currencies (in this case USD and CAD). At the bottom of the rule, the TotalCost is added up using the parameters calculated above.



Price Calculation using iLogic Example

This is another case where there is no correct or incorrect method to use. Understanding the differences will help you create your configurable model to suit your needs.

iLogic Best Practices

- Constraining using the origin or work geometry helps make your rule more robust, especially when bringing parts in using iLogic, or replacing parts in general. This is because these elements themselves are less likely to change and cause errors in your rules

- Using meaningful names for your components, parameters, and features will help you not only when coding, but just in general with modeling. (ie. CornerBracketLeft as opposed to Part1)

- Try to avoid explicit values in your rules. Wherever you can, use variables or named parameters in your rules so you can make changes in a single location

- Comments are an underrated feature in iLogic. Use them to your advantage! Explaining what each rule or section of rule is doing in plain terms will allow you or your colleagues to understand much quicker what is happening

- Input prompts, forms, or message boxes should be avoided.* While they are great for troubleshooting your configuration model, keep in mind you'll need to make those particular elements inactive prior to uploading, so you shouldn't rely on them for your user experience or configuration requirements.
  **Specifically when modeling for a web configurator*

Extend to the WEB

Using the Autodesk Forge Design Automation API for Inventor enables iLogic driven Inventor models to be deployed to the WEB. This is a link to a previous Autodesk University on getting started with Design Automation on Forge.

https://www.autodesk.com/autodesk-university/class/Getting-Started-Design-Automation-Inventor-Forge-2019

It is important to note that Autodesk Forge Design Automation API for Inventor is very powerful but does have some limitations and requirements to be used effectively.

- Programming Experience is required
  While iLogic is a guided development environment, Forge is not. Effectively using the Forge Design Automation API for Inventor requires deep knowledge in programming language(s) and techniques.

- Not Interactive
  Unlike iLogic on the desktop, Forge is not an interactive environment. That means, that code will be required in the configurator user interface to handle conditions such as size limits and component dependencies so that valid parameters are submitted to Forge.

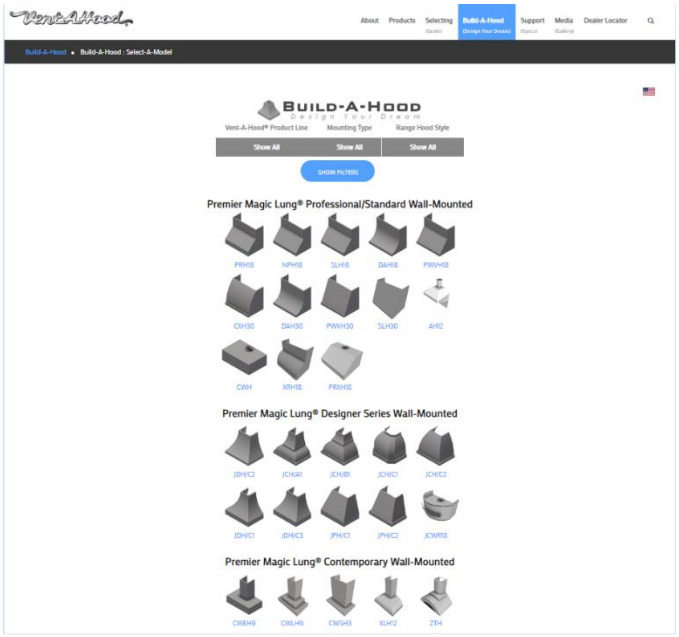
- Supporting Infrastructure
  There are significant setup and management requirements for Forge and the WEB interface environment, such as AWS.

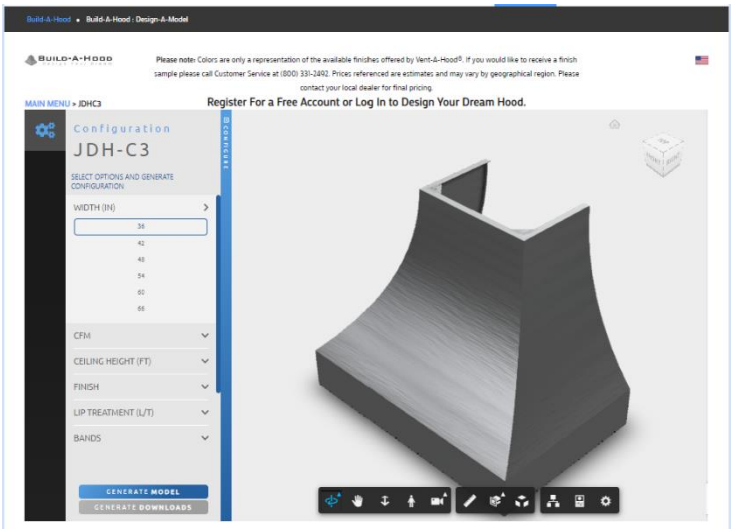
These considerations are not meant to discourage the use of Forge but, rather, to encourage anyone considering a Forge based project to confirm in advance the full technical and personal competencies that will be required.

SolidCAD Variant

After evaluating several options, Vent A Hood decided to base the Build-A-Hood configurator on SolidCAD Variant. The driving factors included:

- Variant provided Vent A Hood with the Power of iLgoic and the Forge Design Automation API for Inventor without the complexity of building their own applications.
- Variant is fully WEB based, making it available to the general public without complex infrastructure to manage.
- Variant made WEB based configuration quickly attainable with a simple annual subscription, letting the engineering and sales teams continue to focus on what they do best.





Build-A-Hood can be accessed at
https://www.ventahood.com/index.php/build-a-hood

Details on SolidCAD Variant can be found at
https://www.solidcad.ca/products/solidcad-products/variant/