FAB466294

Anybody Can Do It! Easily Build Revit Content in Inventor

Pete Strycharske
D3 Technologies

Learning Objectives

- Differentiate unique BIM enabled surfaces using custom Inventor Appearances
- Utilize component library approach to quickly build assembly models
- Create a single Shrinkwrap model representing the final Revit model
- Automate the creation of BIM connectors using the custom Inventor Appearances

Description

There is an increasing desire to build custom designs that can more easily convert from traditional mechanical / industrial CAD tools into BIM ready content that can be utilized inside of Revit. Autodesk Inventor has the capability to generate Revit-ready content, but often this requires specialized knowledge of the process. What if things didn't have to be this way? What if… any CAD designer could quickly assemble components with BIM enabled features, any CAD designer could easily compile multiple components into one compiled model, any CAD designer could populate BIM connectors for final insertion into Revit with a click of a button? Doesn't this sound amazing! The workflow in this class will utilize everyday Inventor techniques to set the stage for powerful iLogic capabilities to allow virtually any designer to configure Inventor models into Revit-ready content.

Speaker(s)

I am an implementation consultant with D3 Technologies, a Platinum Autodesk Partner and Authorized Training Center, based out of our Minneapolis office. I focus primarily on the following areas engineering design and manufacturability, design automation and configuration, process efficiency and manufacturing layouts. Typically, I will partner with clients to perform an assessment of a design or process, determine some improvements, propose a path forward and develop content / mentor users to implement the project. I'm also an Autodesk Certified Instructor and professionally certified in AutoCAD, Inventor Professional and Fusion 360. I frequent the Inventor and Factory Design Forums / Idea Stations, so if you ever have a question, please just ask! Privileged to have attended and taught at Autodesk University; love sharing the crazy stuff I work on and always looking to learn more from all the excellent sessions!

Table of Contents

The Goal

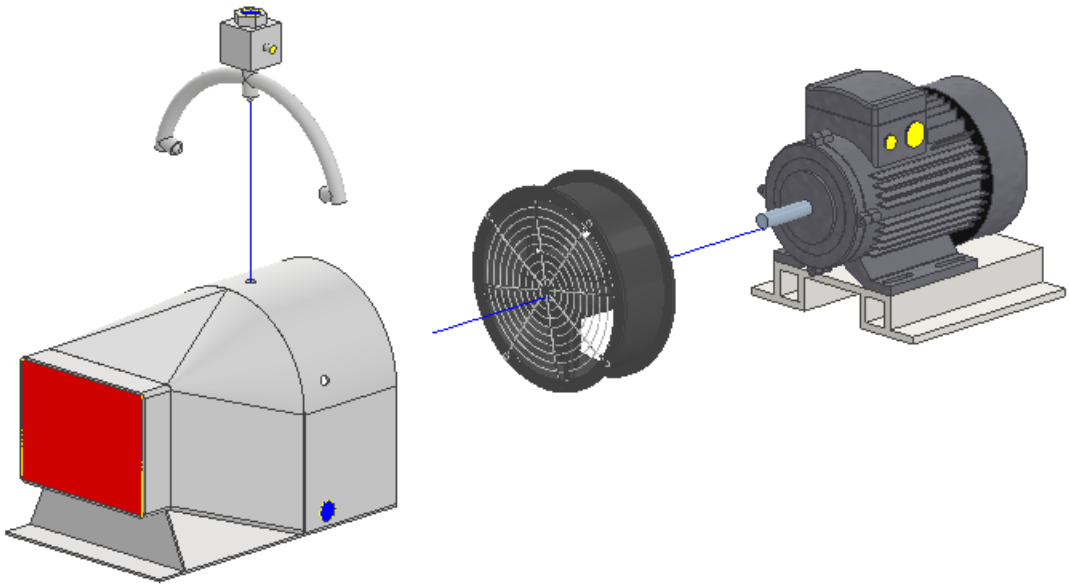
Why is this topic even important?

As design, manufacturing and construction industries increasingly become more sophisticated and intertwined, the way we do things must adapt to meet these new realities. With modular and prefabricated designs gaining huge traction in the construction industry, the time is right to take the same approach in the way we design products for building construction. If we can utilize this approach, we can drastically improve our efficiencies and produce models that are more ready for end user consumption, with the added advantage of making this process available to as many users as possible.

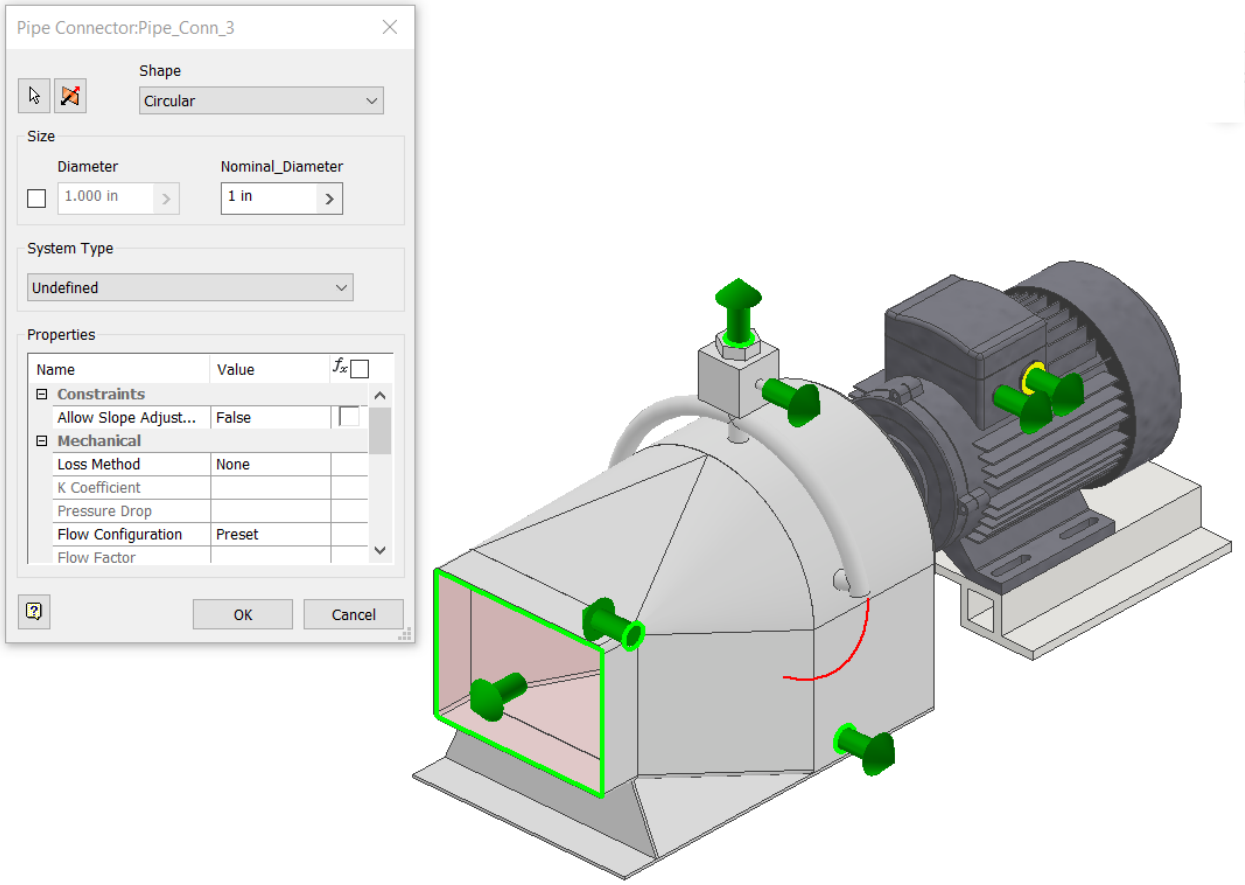
How do we accomplish this (in a nutshell)?

The approach that I'm proposing has both simple and more complex aspects. You may of course take on as much of this as you wish, as the process could be automated all the way to point of pre-configuring all the BIM connectors and publishing the model in Revit Family format. The general steps for this solution are as follows:

- (Optional, but HIGHLY recommended) Simplify the designs as much as possible (Shrinkwrap will help with this as well, more on that later…)
- Prepare the individual component models with the appropriate BIM specific surfaces
- Assemble the components and utilize the Shrinkwrap workflow to create the final configuration model
- Utilizing iLogic rules to configure the BIM connectors to the desired level and even export the model as a Revit family (optional)



SAMPLE ASSEMBLY WITH SIMPLIFIED MODULAR DESIGN



ILOGIC RULE DEPLOYED TO AUTOMATE THE CREATION OF BIM CONNECTORS

Setting up the Models

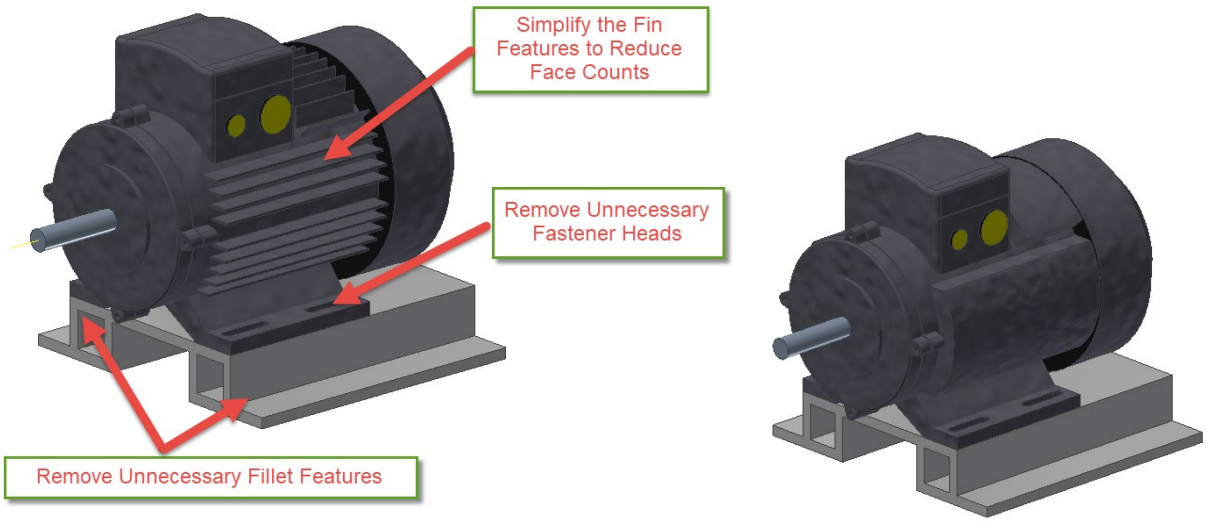
There are a handful of items that must be done to facilitate the process and improve the final performance for the Revit user.

- Simplify the model as much as possible
- Create and store the custom BIM specific appearances
- Apply the custom BIM appearances to Solid Faces
- Apply the custom BIM appearances to Surface Bodies

Simplify the model as much as possible

When we're designing components that are required to complete an actual engineering design, we must include an adequate amount of detail to complete the design. Often this may include displaying every single fillet edge for a part or every fastener used in an assembly. However, when we are producing models for downstream consumption in Revit, Plant 3D, Factory Design, etc. we must NOT put the full engineering details into this model, as this will decrease the performance in Revit and leave the end user frustrated with the model.

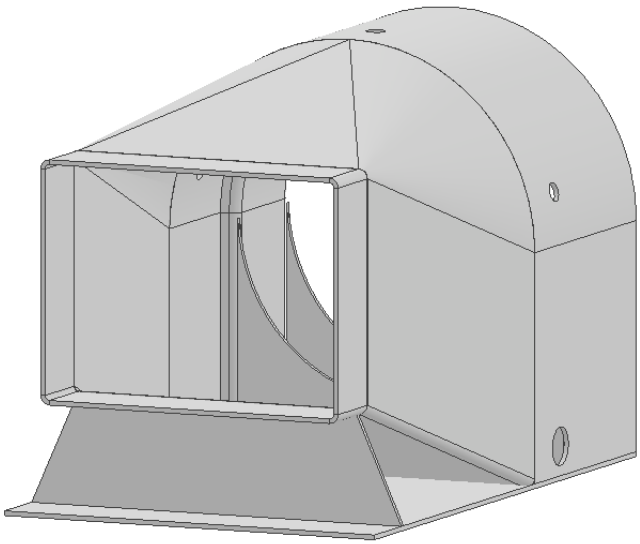
When considering the simplification of a model, there are a couple of different approaches. First, if possible, remove any unnecessary features. This could involve removing features that you yourself have modeled, like fillets. If you've imported a model, this could involve removing unwanted surfaces, like fastener heads, etc. Whatever we can do to "lighten" up the load on the model is advisable, as long as we don't compromise the overall design intent. (We still want to understand what the item is or have the information required to install the item.) See the image below as an example.



SAMPLE MOTOR MODEL SHOWING FEATURES THAT COULD BE REMOVED

If you are not sure whether or not to keep a feature, this step could also be accomplished with an iPart or a derived part approach. That way the original engineering model is preserved and a more Revit friendly model can be utilized.

If you have the luxury of standard equipment or treating this more like a product configurator or sales tool, but need to provide a 3D model for the customer, then you can build the model from scratch with only the level of detail required. This approach is also desirable because the model can be designed parametrically to cover a wide range of design scenarios and options. In this sheet metal hood example below, I utilized a multi-solid body approach so I could create more detailed parts later, if we land the job. You could even make such approach a part of your Copy Design process, if you use Vault or some other data management tool.



SHEET METAL HOOD DESIGN UTILIZING A MULTI-BODY MODELING APPROACH

Parameters

Parameter Name	Consumed by	Unit/Type	Equation	N
Model Parameters				
Reference Parameters				
User Parameters				
Fan_OD	d2	in	10 in	1
Fan_Inner_ID	d1	in	9.03125 in	9.
Fan_Inner_Width	Fan_Chamber_Length, d0	in	3.1 in	3.
Fan_Height_Position	d23, d13, d3	in	6 in	6.
Base_Plate_Length	d29	in	16 in	1
Fan_Chamber_Length	d40	in	Fan_Inner_Width * 2 ul	6.
Water_Inlet_Angle	d71, d58	deg	75 deg	7
Outlet_Duct_Width	d82, d74	in	8 in	8.
Outlet_Duct_Height	d75, d83	in	6 in	6.

Add Numeric | Update | Purge Unused

Link | Immediate Update

PARAMETRICALLY DRIVE THE MODEL MORE EASILY WITH A NEW MODEL

Create and store the custom BIM specific appearances

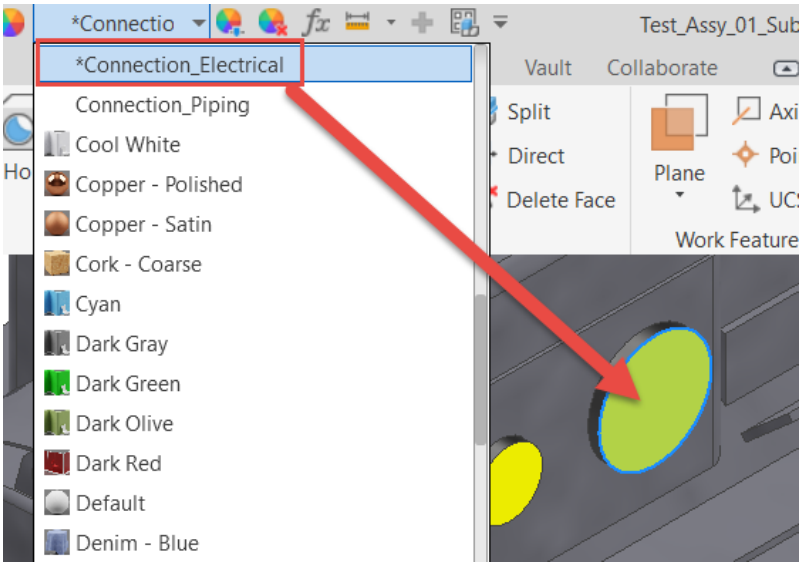
Once the model has been simplified with only the required geometry and features, we turn our attention to the most critical piece of the modeling process, defining the custom appearances that are going to be used to drive the creation of the BIM connectors. A natural and good question at this point, is why have I chosen to base the solution on custom appearances? I had three reasons for going this route:

- Custom appearance names can convey valuable information for driving BIM connector configurations.
- Appearance color itself can serve as a visual cue that a surface is unique and help users better understand the component being utilized.
- When using the Shrinkwrap process, custom appearances will pass through to the final component configuration.

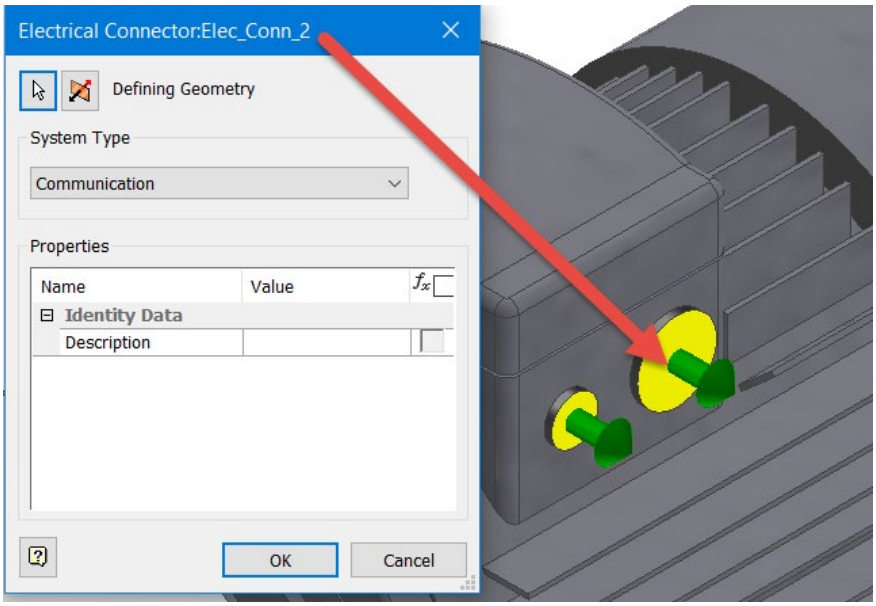
The last point is one of the most important, as usually an assembly model is FAR too detailed to pass along to a customer and the Shrinkwrap process helps us to reduce a models complexity and aids in the protection of intellectual property.

Custom Appearance Naming Considerations

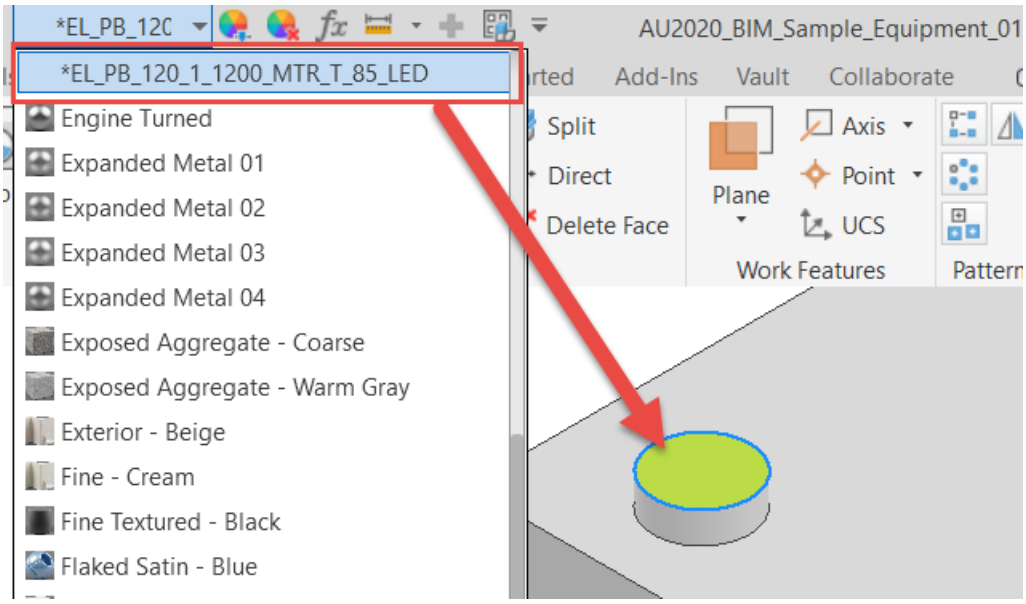
There are a couple different approaches that someone can take to defining the appearance name, generic or specific. If we take the generic path, then the BIM connectors that are generated will only be classified at a high level; “Electrical” or “Duct” for example. However, if we use a more specific type of naming convention, then we can classify the connector with more detail; for example, “Electrical – Balanced Power – 240 Volts, etc.”



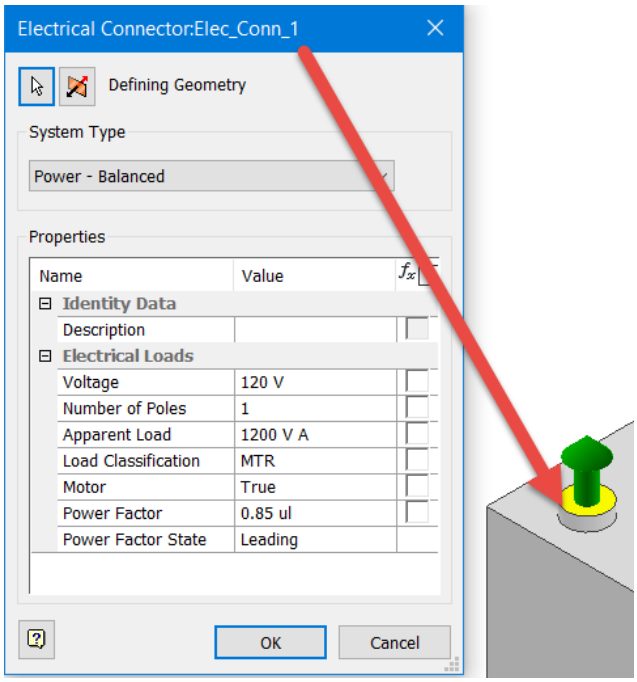
GENERIC APPEARANCE NAME AND ASSIGNMENT



GENERIC APPEARANCE NAME RESULTS IN GENERIC BIM CONNECTOR CREATION



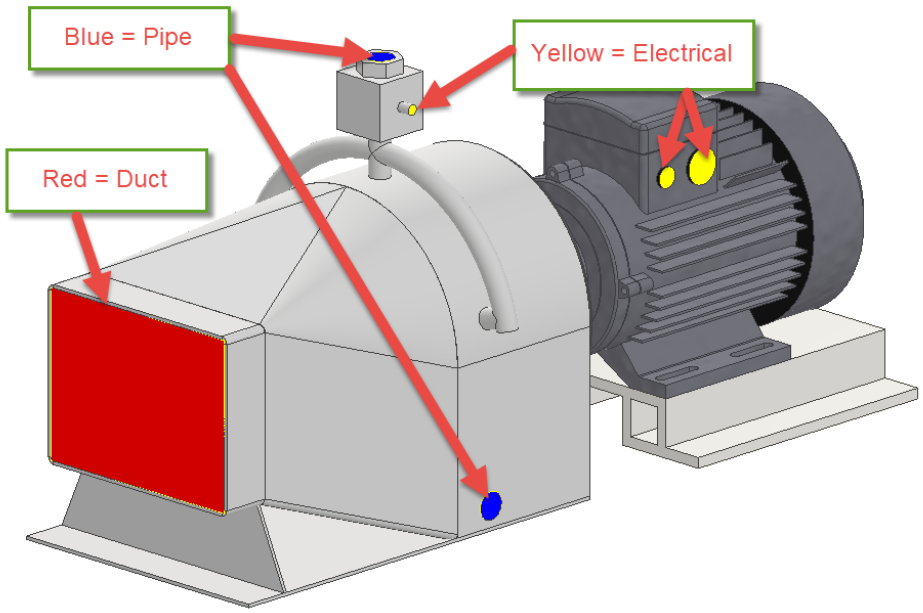
SPECIFIC APPEARANCE NAME AND ASSIGNMENT



SPECIFIC APPEARANCE NAME RESULTS IN DETAILED BIM CONNECTOR CREATION

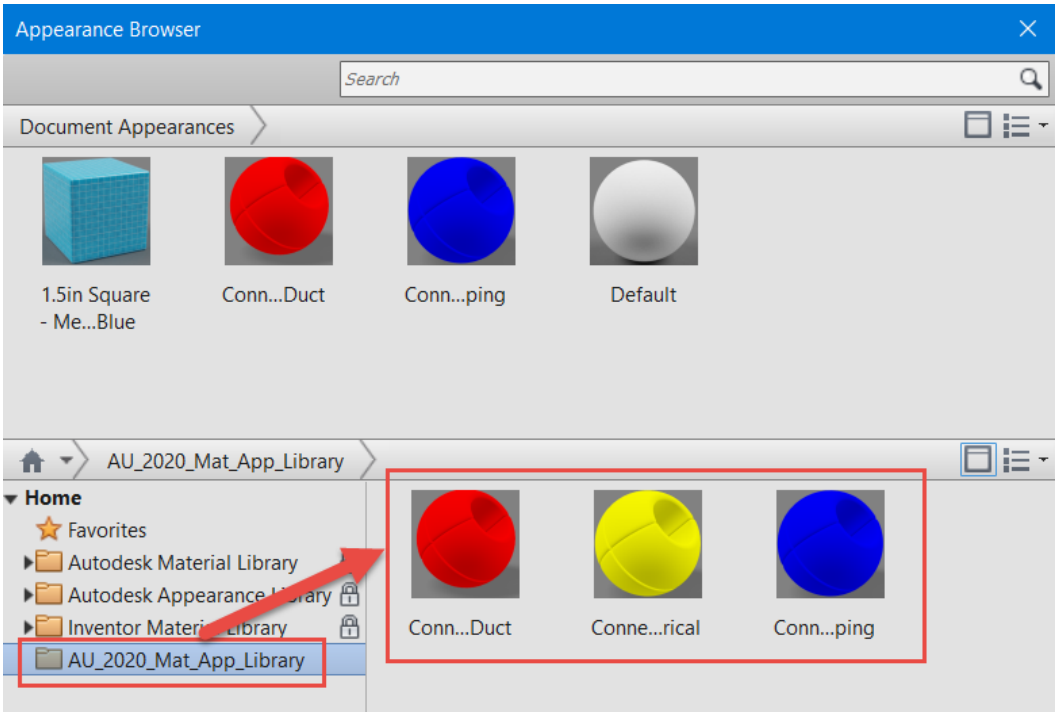
Custom Appearance Color Considerations

While applying a specific color to the custom appearance technically isn't necessary, there are some strong benefits. For example, a downstream user of the component will immediately understand that a particular surface is unique and, with time and standardization, what type of connection is required. Additionally, a user would be able to ascertain what type of positioning or tool clearance may be required for a given component configuration. We would want to ensure the designer leaves ample room for critical maintenance on a unit's power supply connection. The color scheme I have used only serves as an example but illustrates the point well.

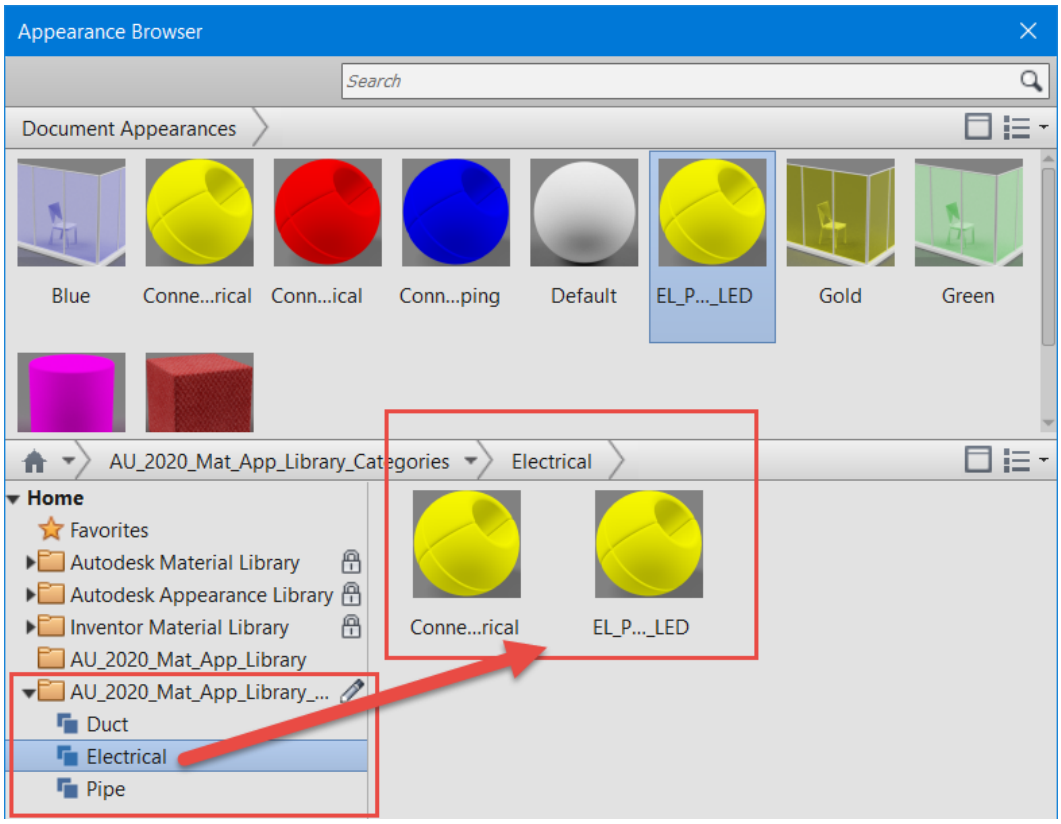


CUSTOM APPEARANCE COLORS CAN COMMUNICATE SURFACE FUNCTION

Because these custom appearances play a crucial role in the process and we may have dozens (or more) components to configure, we must manage and reuse these appearances efficiently. The easiest and best way to accomplish this is to create a Material / Appearance library or add these appearances to your existing library. If you choose, these surfaces can further be organized into library categories, as shown below.



CUSTOM MATERIAL / APPEARANCE LIBRARY WITHOUT CATEGORIES

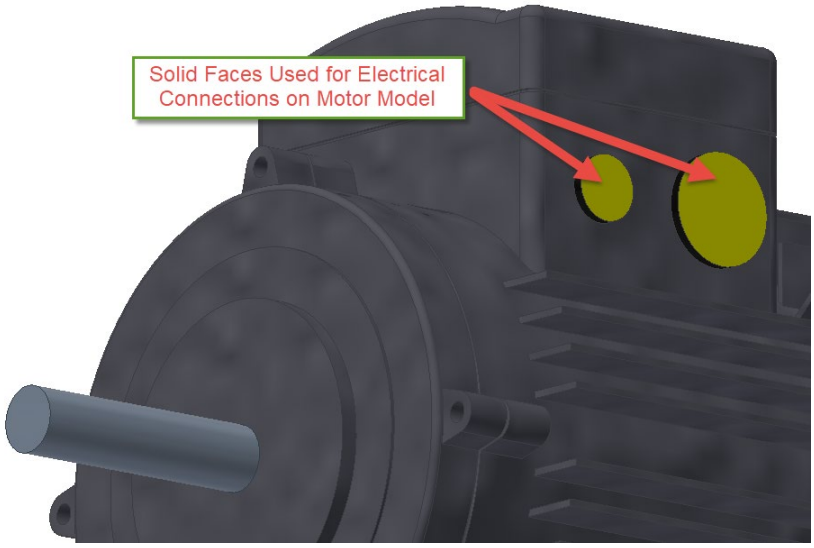


CUSTOM MATERIAL / APPEARANCE LIBRARY WITH SPECIFIC CATEGORIES

Once the desired custom appearances have been created the components can be reconfigured to utilize these appearances to drive the automation of BIM Connector generation. The custom appearances can be utilized in two different ways.

Apply the custom BIM appearances to Solid Faces

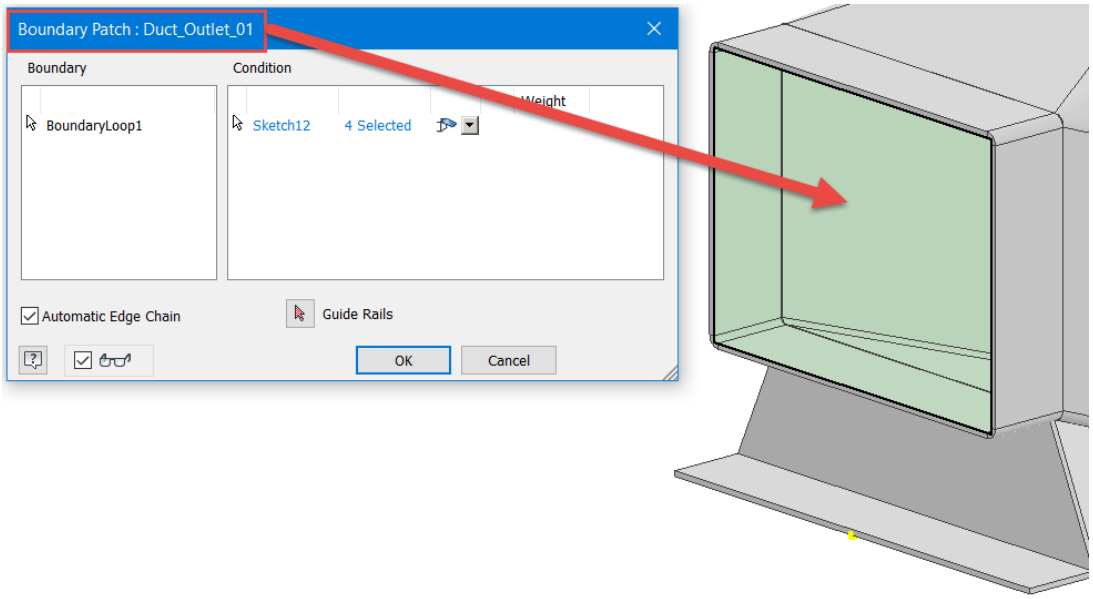
Depending on the models being used, solid faces are excellent candidates for receiving the custom appearances. This is also the most straightforward approach as no additional modeling is required, as is typically necessary to utilize surface bodies. Solid body faces work great for sales engineering models or for connections that don't have a real "orifice" (electrical connections vs. duct connections, for example).



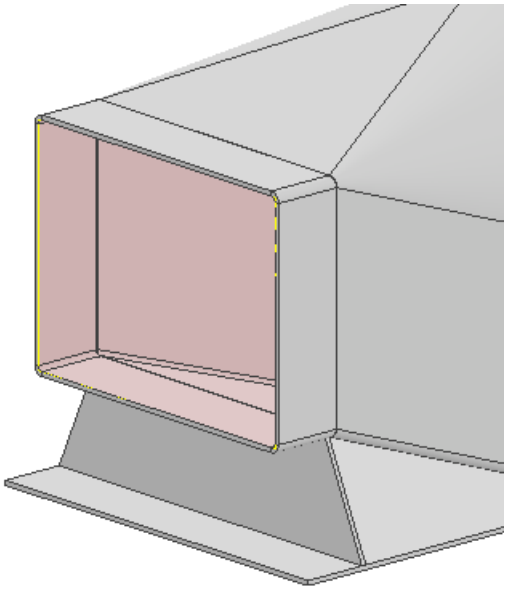
SAMPLE MOTOR MODEL USING SOLID FACES FOR ELECTRICAL CONNECTIONS

Apply the custom BIM appearances to Surface Bodies

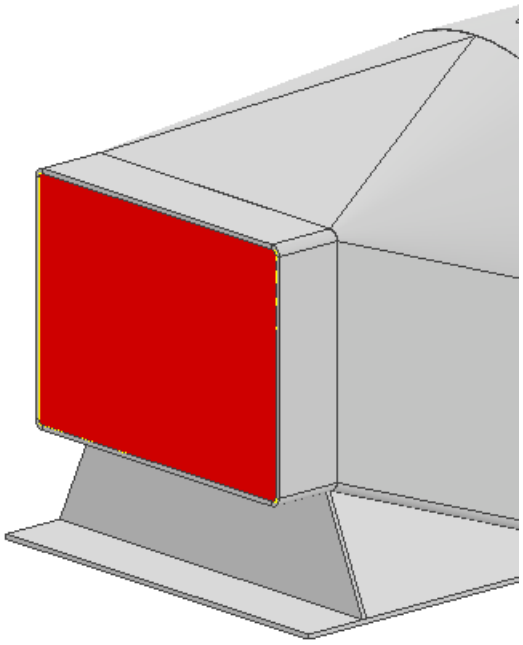
For some component types, solid faces simply aren't appropriate. For example, if a component has fluid flow and may be used for a flow analysis, solid bodies will definitely not work. Additionally, if the design is derived from an existing engineering component, which contains orifices, we may not desire to create multiple versions that include solid faces. Thankfully, the custom appearance applied to surface bodies is as equally effective as using solid faces. While this will involve the extra work of creating the surface bodies, this approach offers the flexibility of utilizing existing components largely as-is. I won't go into detail in creating surfaces, but please see the screenshots below as a general reference for the steps required and the results.



CREATE A DUCT OUTLET BOUNDARY PATCH SURFACE USING THE SHEET METAL LOFTED FLANGE SKETCH



CUSTOM APPEARANCE APPLIED TO THE TRANSLUCENT BOUNDARY PATCH



CUSTOM APPEARANCE APPLIED TO THE OPAQUE BOUNDARY PATCH

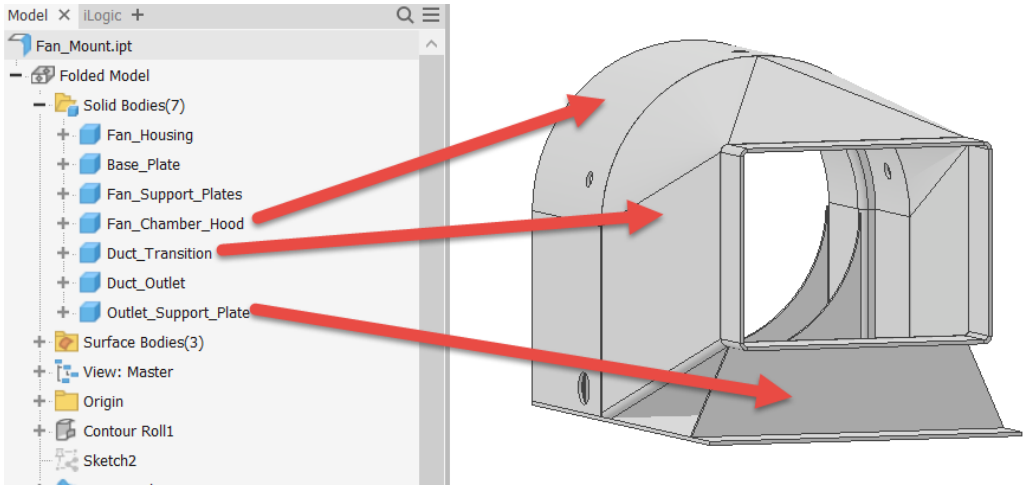
I wanted to show that the surfaces can be utilized in either the native translucent format or in opaque format. You should use whatever will be most consistent and easiest for your design group to work with. I will be using the opaque format for the rest of this process.

The Strategy – Using a Component Approach

To make the process as easy to implement as possible, I'm recommending a configured component approach. The idea being there is a roster of preconfigured, simplified, components containing the information necessary to efficiently complete BIM / Revit ready designs. The more information that we can place into these models upfront, the less work required for downstream designers to configure these designs.

Utilize simplified versions

We discussed some simplification ideas in the previous session and now we'll utilize these models to create the individual components. For this portion of the presentation, I'm going to use a multi-body model to represent a fan shroud and duct transition assembly.



FAN MOUNT ASSEMBLY MODELED AS A MULTI-BODY PART FOR GREATER CONTROL AND FLEXIBILITY

The multi-body approach is effective because it is easy to control the overall conceptual design, without having to alter multiple part files all at once. See the images below for a parametric change that impacts several solid bodies simultaneously.

Parameters

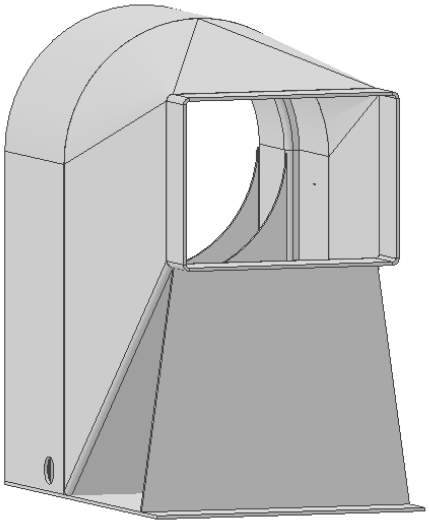
Parameter Name	Consumed by	Unit/Type	Equation
Model Parameters			
Reference Parameters			
User Parameters			
Fan_OD	d2	in	10 in
Fan_Inner_ID	d1	in	9.03125 in
Fan_Inner_Width	Fan_Chamber_Length, d0	in	3.1 in
Fan_Height_Position	d23, d13, d3	in	6 in
Base_Plate_Length	d29	in	16 in
Fan_Chamber_Length	d40	in	Fan_Inner_Width * 2 ul
Water_Inlet_Angle	d71, d58	deg	75 deg
Outlet_Duct_Width	d82, d74	in	8 in
Outlet_Duct_Height	d75, d83	in	6 in

INITIAL "FAN_HEIGHT_POSITION" PARAMETER VALUE

Parameters

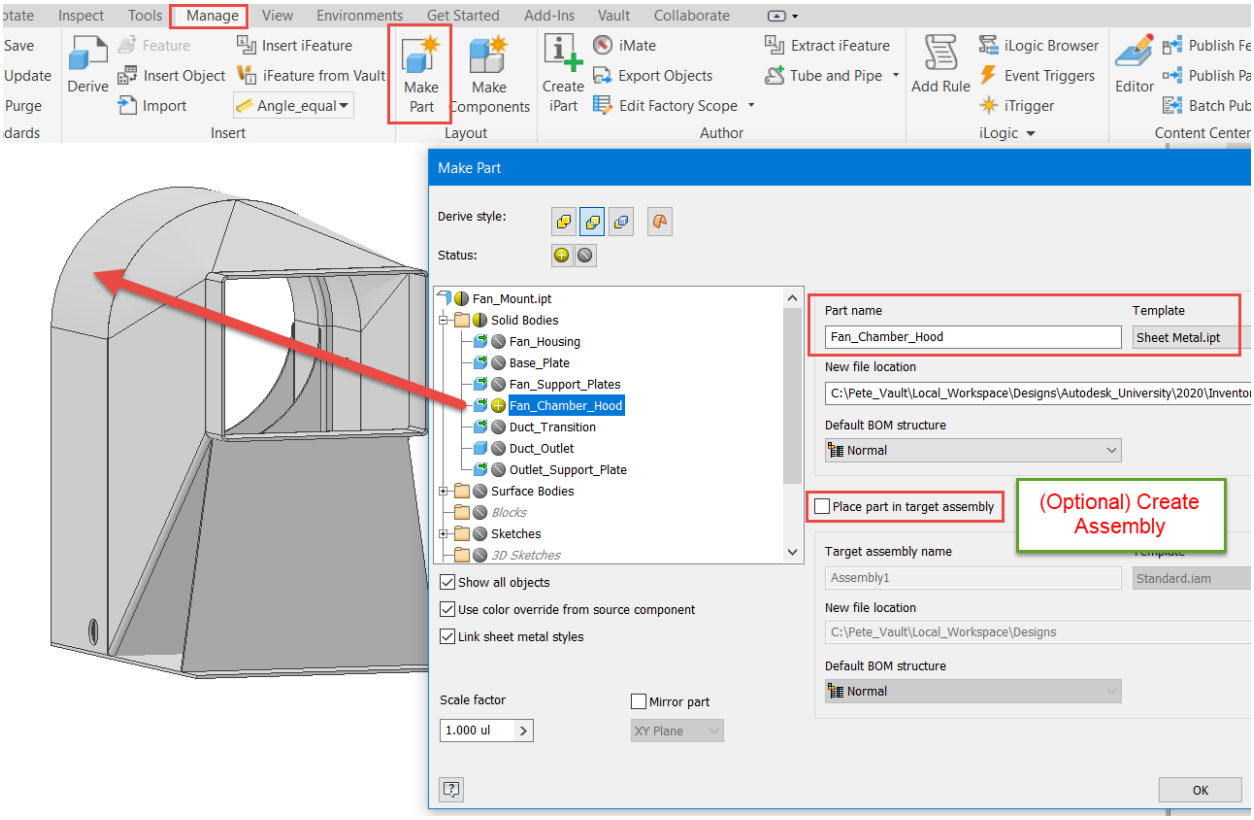
Parameter Name	Consumed by	Unit/Type	Equation	Nomi
Model Parameters				
Reference Parameters				
User Parameters				
Fan_OD	d2	in	10 in	10.00
Fan_Inner_ID	d1	in	9.03125 in	9.031
Fan_Inner_Width	Fan_Chamber_Length, d0	in	3.1 in	3.100
Fan_Height_Position	d23, d13, d3	in	12 in	12.00
Base_Plate_Length	d29	in	16 in	16.00
Fan_Chamber_Length	d40	in	Fan_Inner_Width * 2 ul	6.200
Water_Inlet_Angle	d71, d58	deg	75 deg	75.00
Outlet_Duct_Width	d82, d74	in	8 in	8.000
Outlet_Duct_Height	d75, d83	in	6 in	6.000

ALTERED "FAN_HEIGHT_POSITION" PARAMETER VALUE

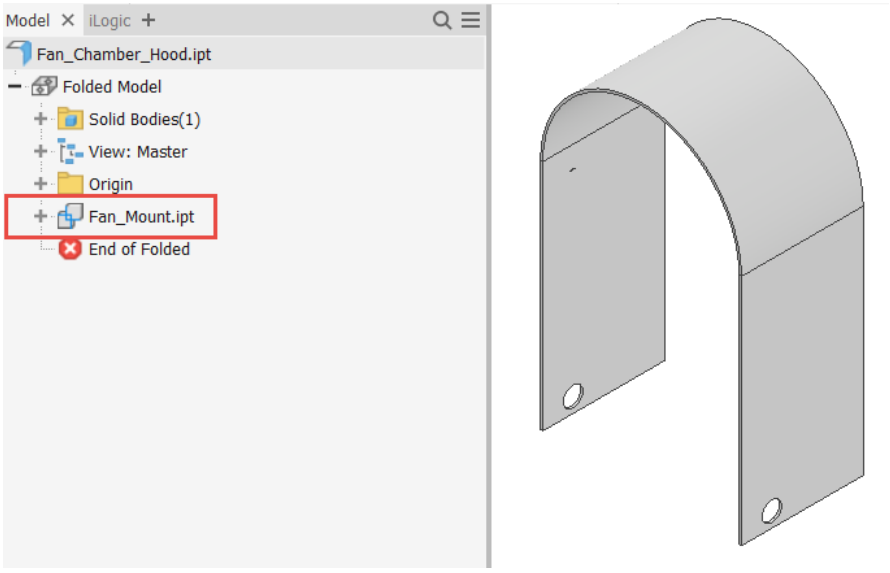


ALTERED "FAN_HEIGHT_POSITION" PARAMETER MODEL RESULTS

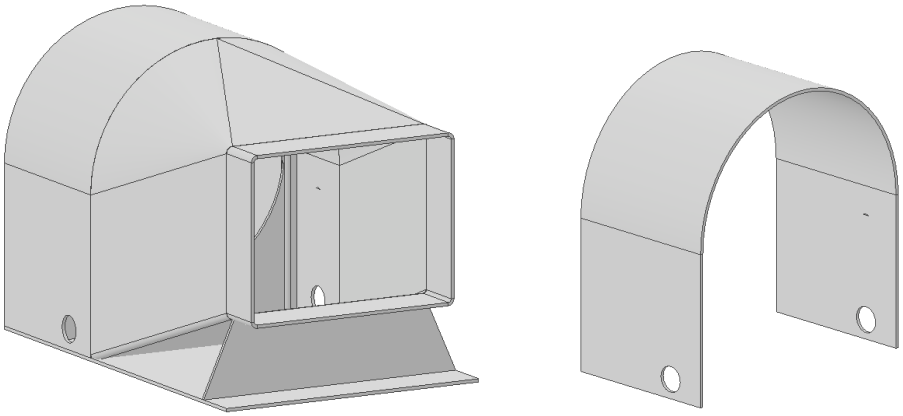
Another advantage of starting with a simple version of the design is a more detailed version can be extracted later, if the initial model becomes a production job requiring more complete engineering. Below is an example of the Make Part command being utilized to create the actual sheet metal component for the "Fan_Chamber_Hood" solid body.



THE MAKE PART COMMAND CREATES A LINKED UNIQUE PART FILE



MAKE PART RESULTS IN A DERIVED DESIGN, LINKED BACK TO THE "FAN_MOUNT" DESIGN FILE



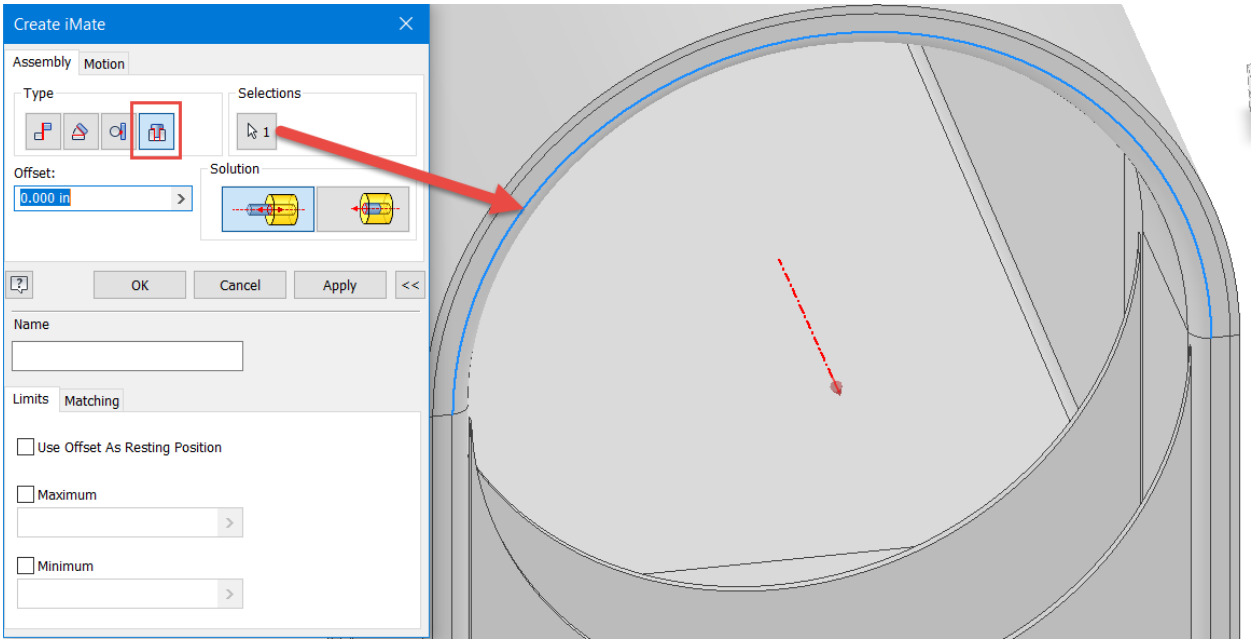
DERIVED DESIGNS ALLOW CHANGES TO THE DESIGN FILE TO FLOW TO UNIQUE PART FILES

I utilize this technique ALL THE TIME to create simplified versions of conceptual or more complex assemblies, knowing that I can transition to more complex engineering work when appropriate.

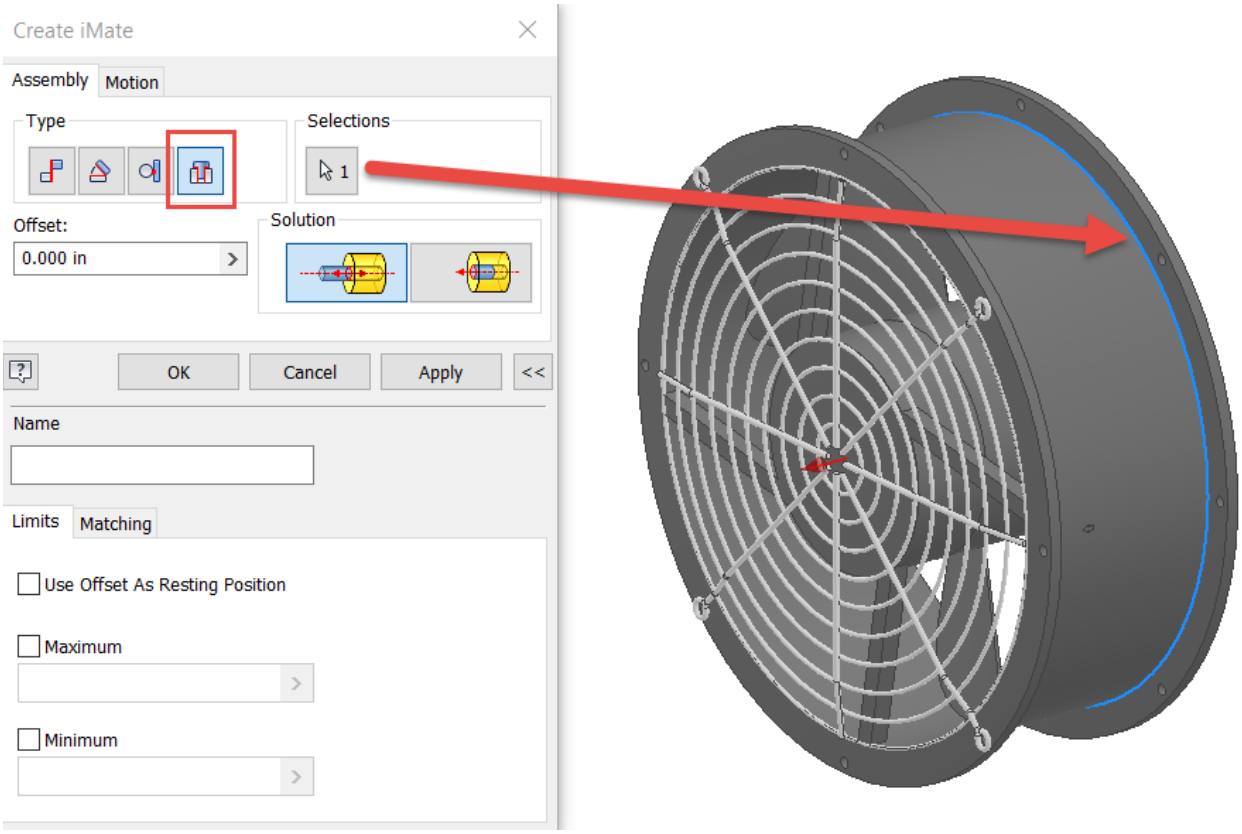
Pre-configuring component characteristics

The most important part of configuring each component, for this process, is to assign the custom BIM appearances to the desired surfaces. Since, we've already discussed those considerations in the section on "Setting up the Models", I want to cover some additional considerations. Besides giving designers the ability to create BIM connectors more easily, we can also add features like iMates, or utilize a Factory Asset approach to improve the efficiency of building the assemblies.

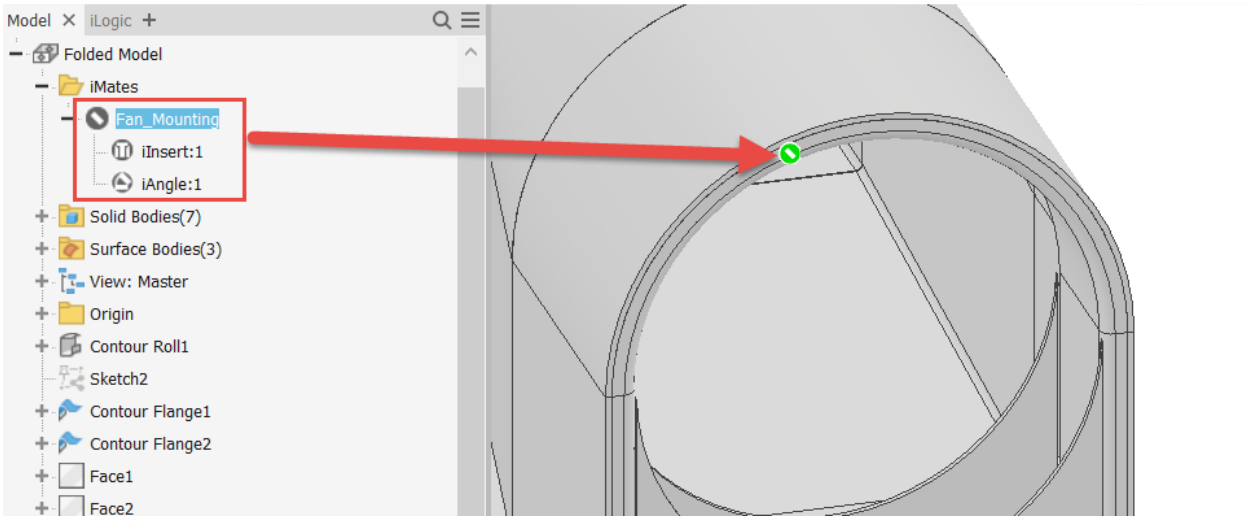
For those who aren't familiar, iMates are preconfigured constraint relationships that can be added to the individual components and activated in the assembly process. iMates can either be added individually or grouped together as a composite iMate, depending on how many assembly steps should be eliminated. Please see the images below for some examples of where iMates could be applied.



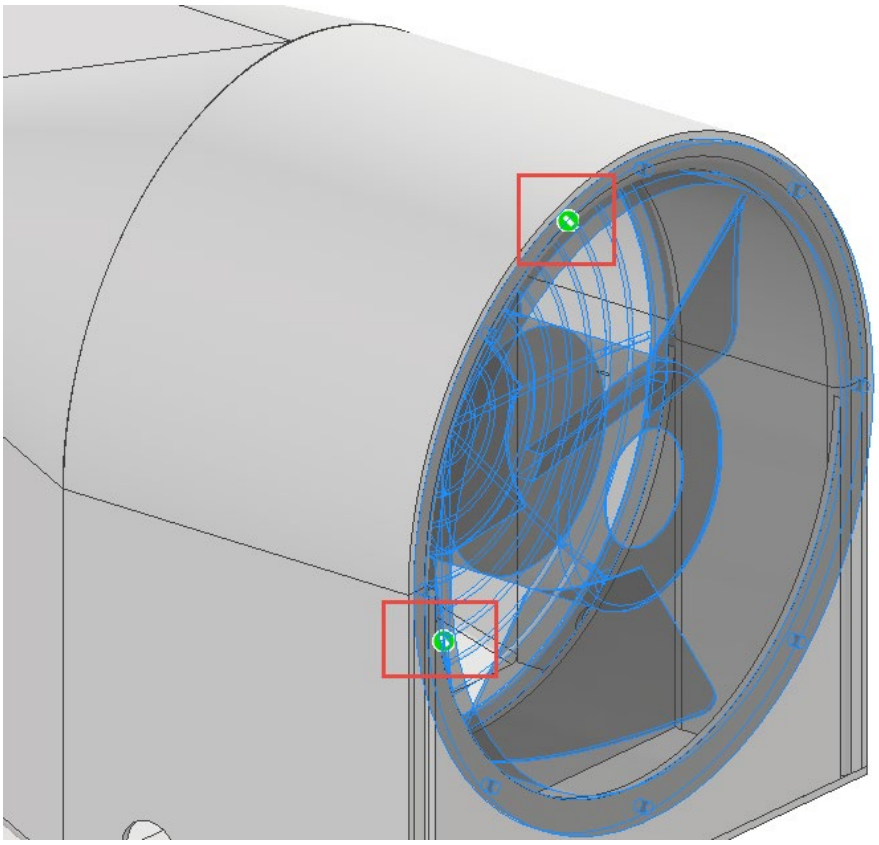
EXAMPLE IMATE USING THE INTERIOR OF THE "FAN_MOUNT" COMPONENT



EXAMPLE INSERT IMATE USING THE INTERIOR OF THE FAN



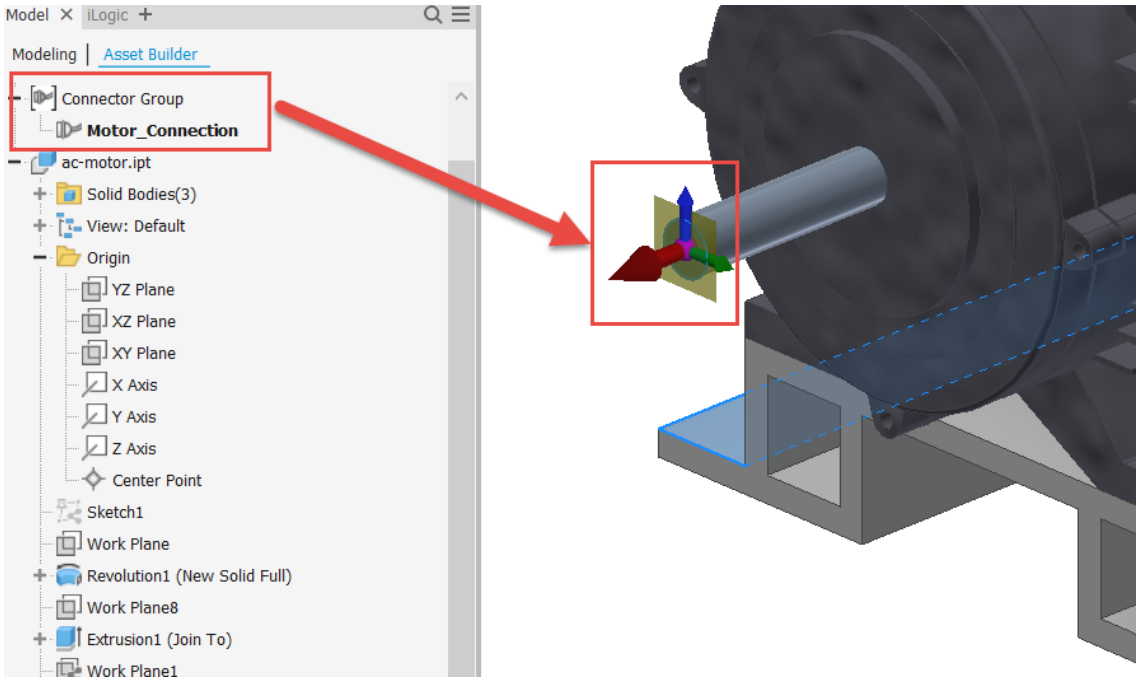
EXAMPLE OF A MORE COMPLEX COMPOSITE IMATE



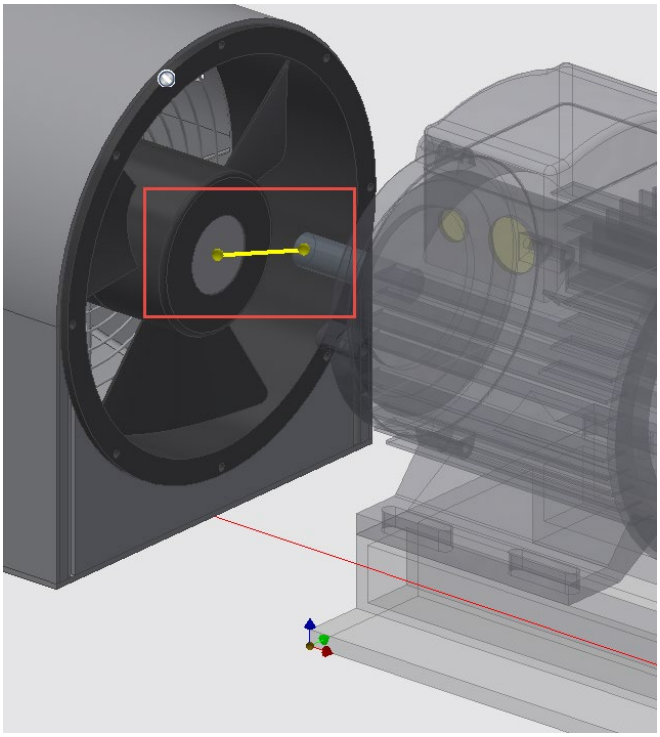
ASSEMBLY RESULT OF RELATIONSHIPS FORMED FROM IMATES

Besides the use of iMates, there are other options available to us. If you've ever heard of the Factory Design Utilities (FDU), you've probably discovered those tools are very useful for the configuration of spaces (factory floors, retail, offices, etc.). However, I have some clients that have ventured out to use the FDU for configuring products because the Factory connector

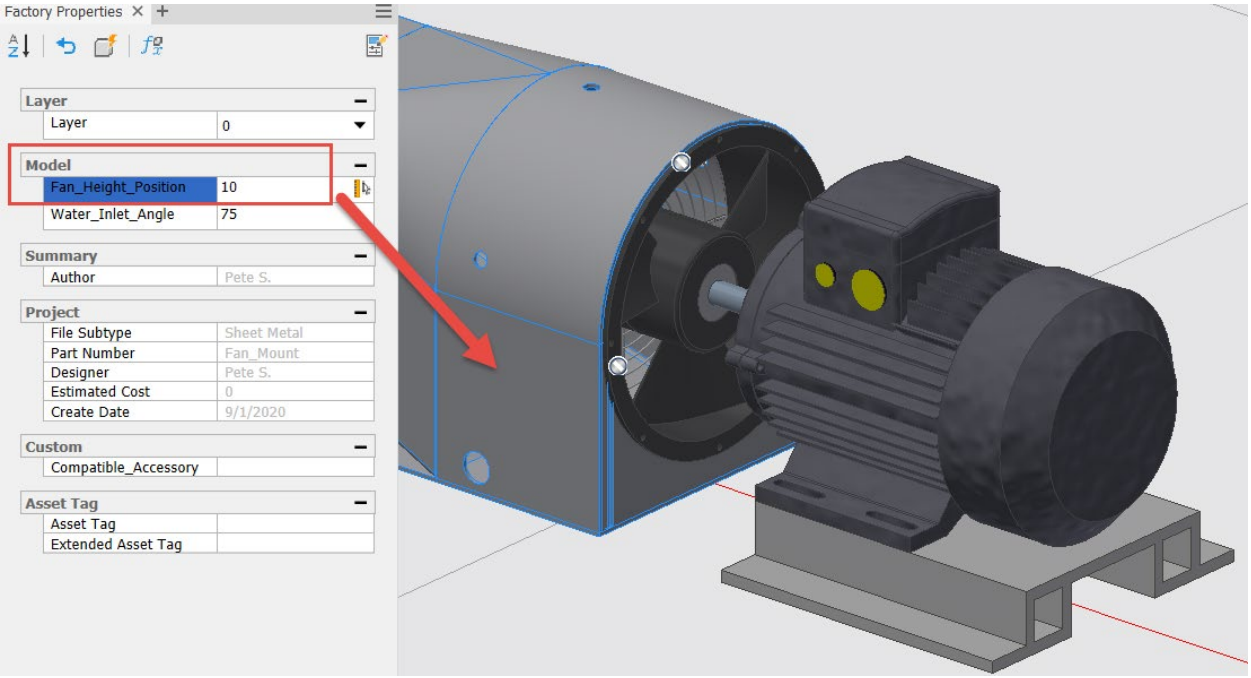
technology allows us to “snap build” models together and to pass along parameter information between components. There are several good AU classes on FDU, so I won’t go into detail here, but please see an example of these FDU components.



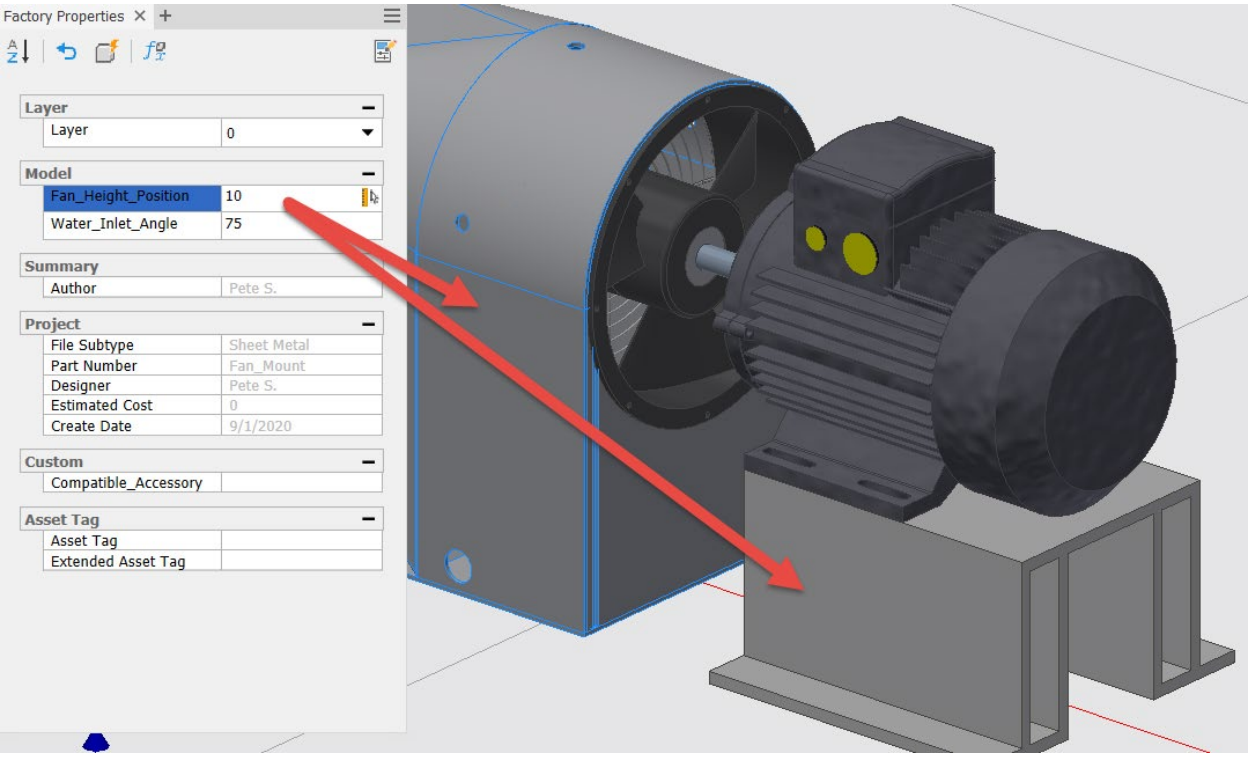
DEFINE THE FDU CONNECTOR, SELECTING THE ATTACHMENT POINT AND ORIENTATION



PLACING FDU ASSETS IN A LAYOUT ALLOWS THE FDU CONNECTORS TO HELP “SNAP” BUILD THE MODELS



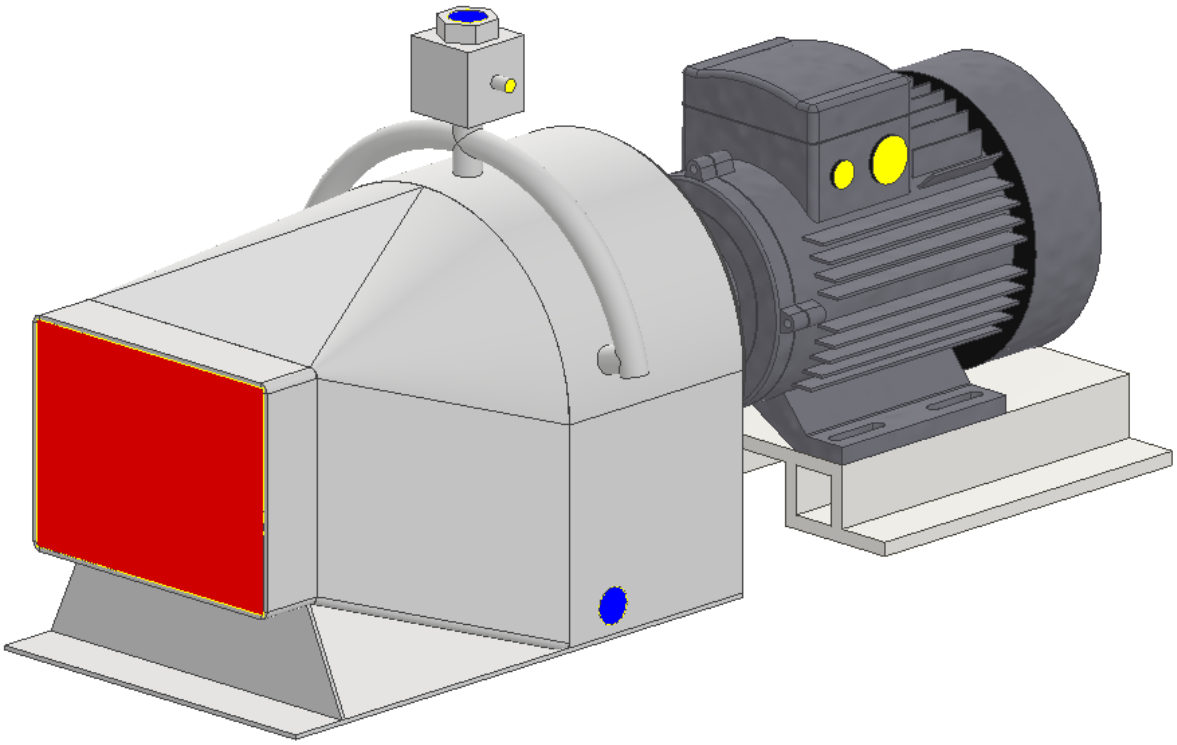
CHANGING FDU ASSET PARAMETERS CAN IMPACT OTHER CONNECTED ASSETS, LIKE A HEIGHT SWITCH FROM 6 TO



10 INCHES CHANGES THE MOTOR BASE HEIGHT, AS WELL

There is an additional potential benefit to utilizing Factory Assets, as methods exist to convert the final configured simplified models into engineered models later. This report is focused more on the creation of the BIM ready models, however in Appendix C, I've provided a link to my AU2018 course on converting Factory models into final engineering models. Please peruse that course if you're interested and let me know if you have any questions.

No matter which techniques you choose, the idea is to find ways to help the end users / designers create the final assemblies as quickly as possible, with the just the right amount of flexibility to meet customer needs.



FINAL ASSEMBLY OF CONNECTED COMPONENTS

Create Shrinkwrap Models for Final Configurations

Creating the components and building the assembly is the most difficult part of the process, so great job on getting this far! The next step in the process allows us to further simplify the model by combining the individual components into a single part file and helps to remove unwanted geometry to streamline the design and protect intellectual property. The Shrinkwrap tool is perfectly suited to help us accomplish all those goals with a few simple selections.

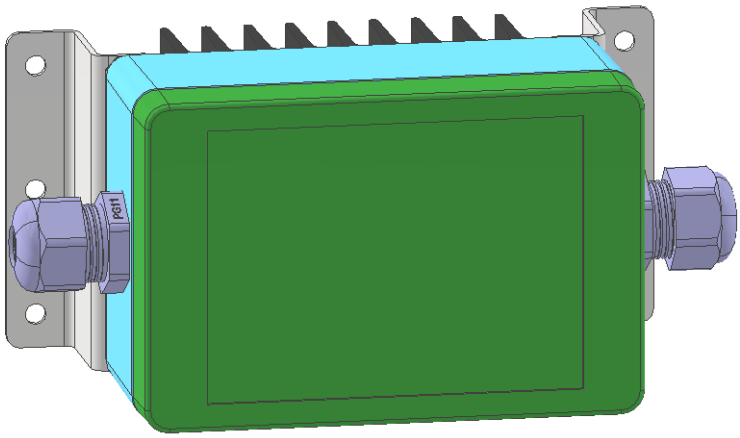
Why use Shrinkwrapping?

There are a handful of reasons why I feel that the Shrinkwrap tool is perfectly suited and I've listed them below. We'll explore each of these in more detail shortly:

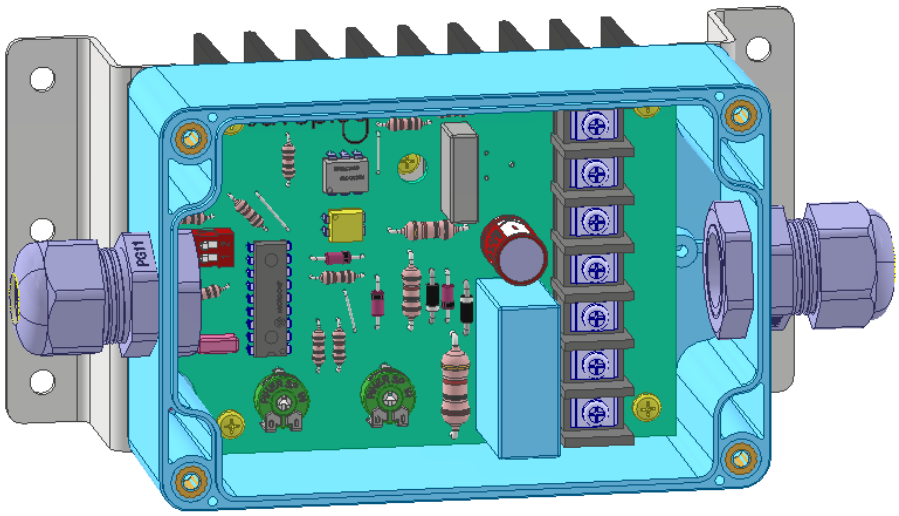
- Selecting the composition of the final simplified part is easy after the changes made to Shrinkwrap with Inventor 2018.
- Shrinkwrapping allows us another opportunity to remove features that are unnecessary for the final part.
- Hollow geometry can be solid filled, so that end customers do not see precious intellectual property, if we don't want them to.
- Even internal designs can benefit from the Shrinkwrap Substitute tool, as we can replace a complex subassembly with the simplified part, while still preserving the BOM characteristics of the subassembly.
- The specialized BIM surfaces that we created and utilized in the component setup will successfully pass through the Shrinkwrap process.

Shrinkwrapping allows for easy selection for final part composition

Despite our best efforts to simplify models as much as possible, there will be some assemblies that end up containing lots of components. To ensure the best possible performance for the Revit users, we need to utilize the Shrinkwrap command's unique functionality to reduce the number of these components and eliminate unnecessary features. To illustrate this, let's look at a component that I downloaded from an outside source.

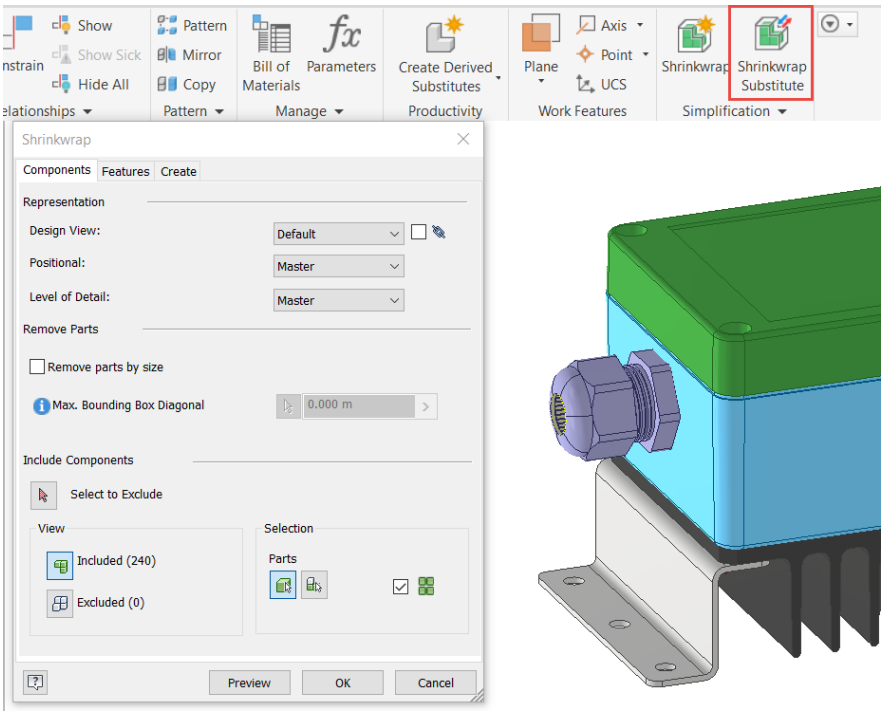


DOWNLOADED CONTROLLER MODEL

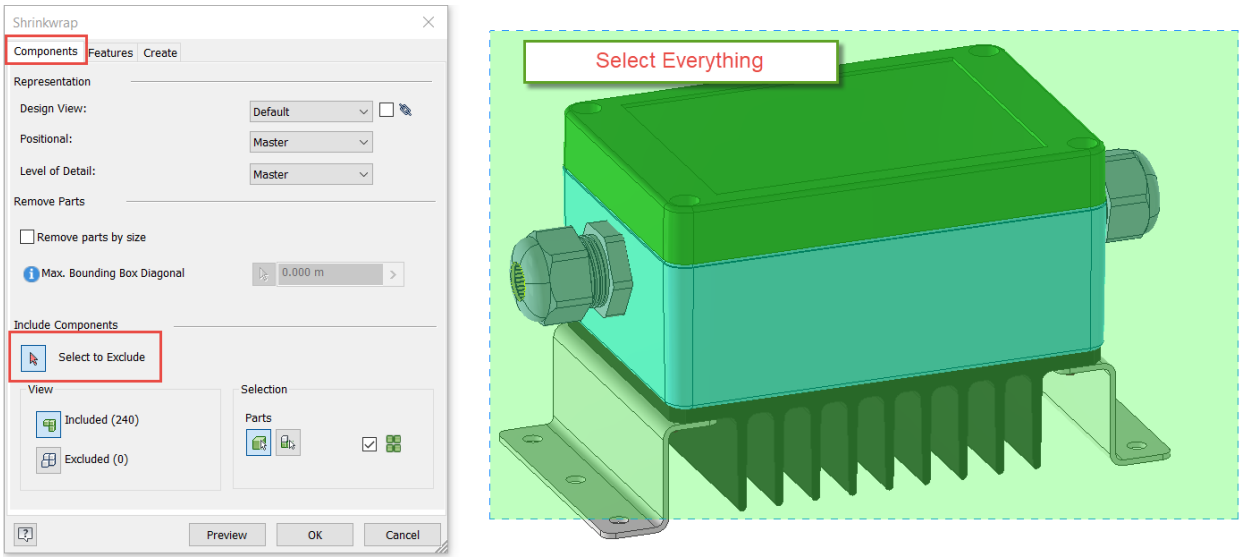


A CLOSER LOOK WITH THE COVER REMOVED REVEALS A PLETHORA OF ELECTRICAL COMPONENTS

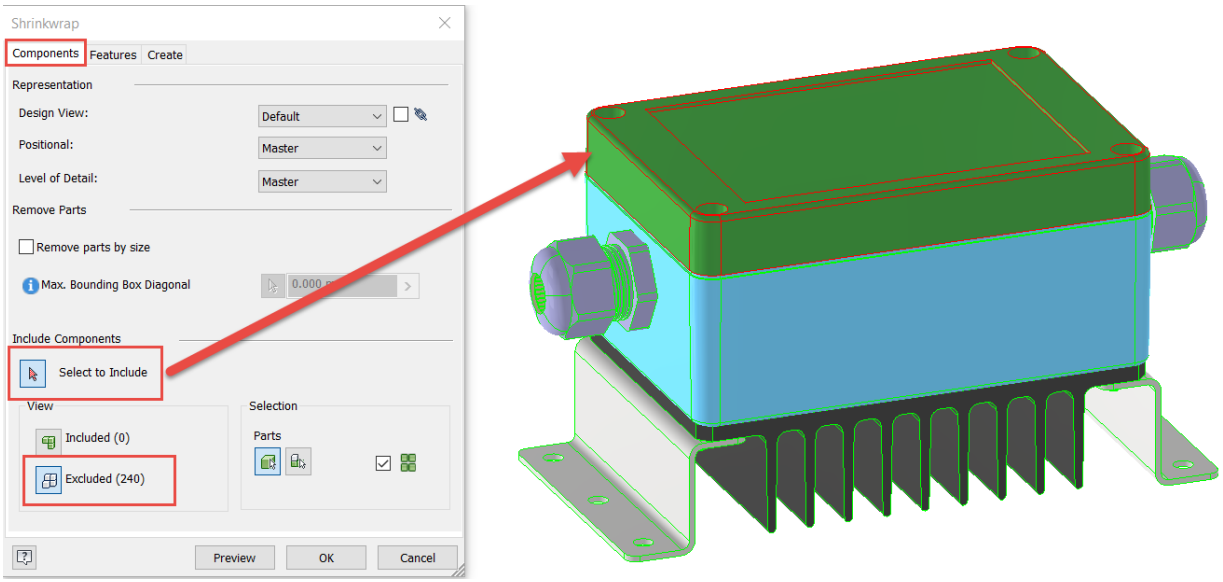
The electrical components that comprise this design are essential to the controller itself, but items that we do not wish to pass along to the customer, so we'll utilize the Shrinkwrap selection filters to remove them from the final design. To maximize the benefits to the team, we'll utilize the Shrinkwrap Substitute command to create a useful level detail as well as a simplified part model.



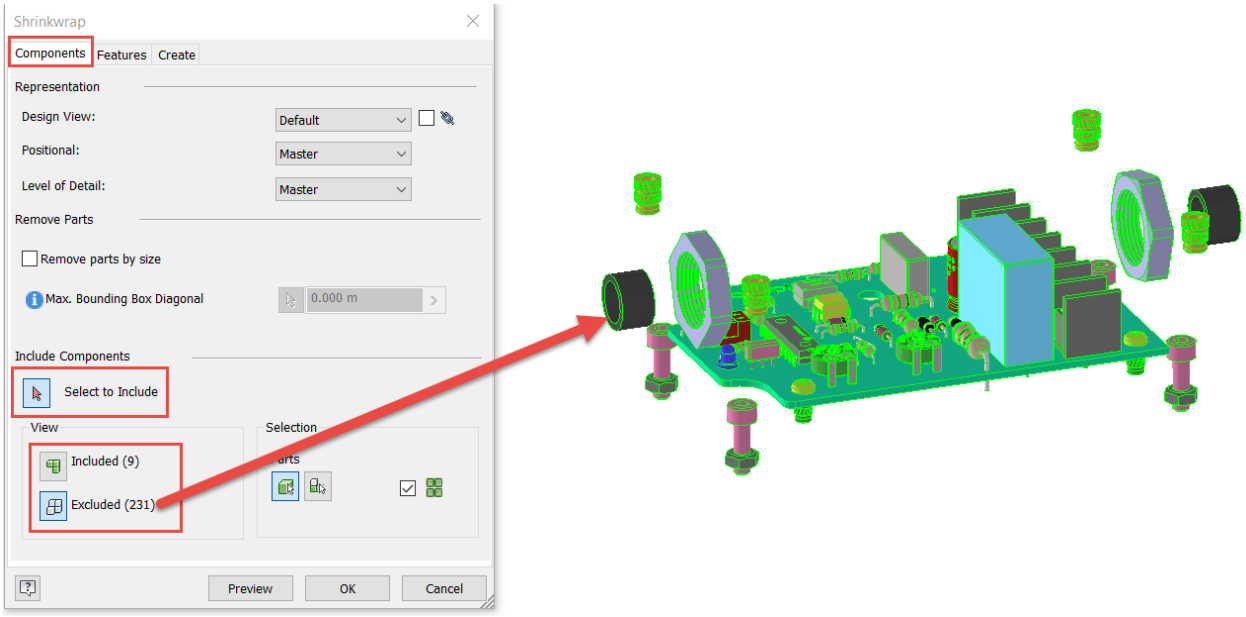
UTILIZE THE SHRINKWRAP SUBSTITUTE COMMAND TO CREATE A SIMPLIFIED PART AND LEVEL OF DETAIL



WHEN MANY COMPONENTS NEED REMOVAL, SAVE TIME BY EXCLUDING EVERYTHING

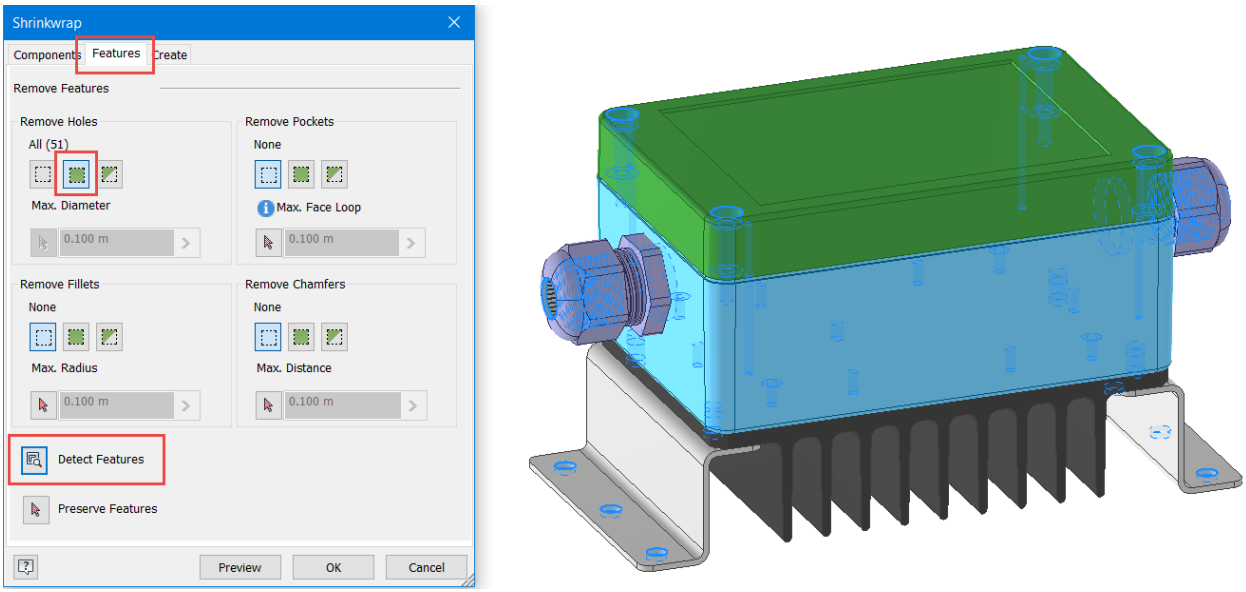


TOGGLE TO SELECT ONLY THE COMPONENTS THAT YOU WISH TO INCLUDE

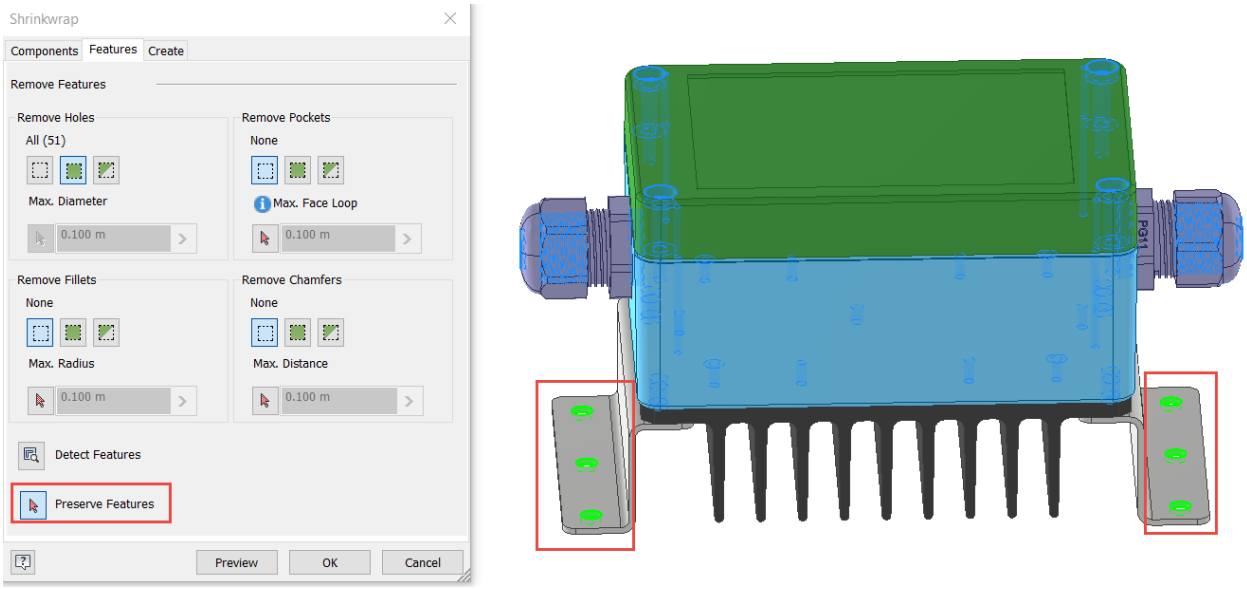


*AFTER SELECTING A HANDFUL OF COMPONENTS **231** WILL BE REMOVED*

After the component list has been pared down to the required final geometry, we can turn our attention to removing any unnecessary features.



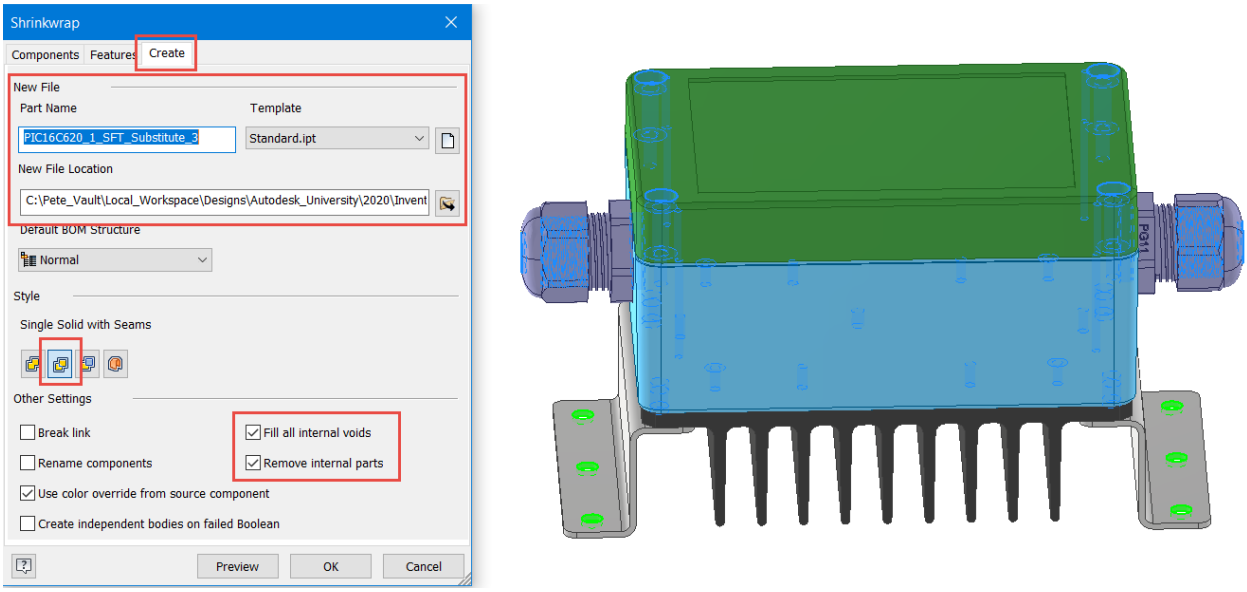
SWITCHING TO THE FEATURES TAB, WE CAN REMOVE ALL HOLES



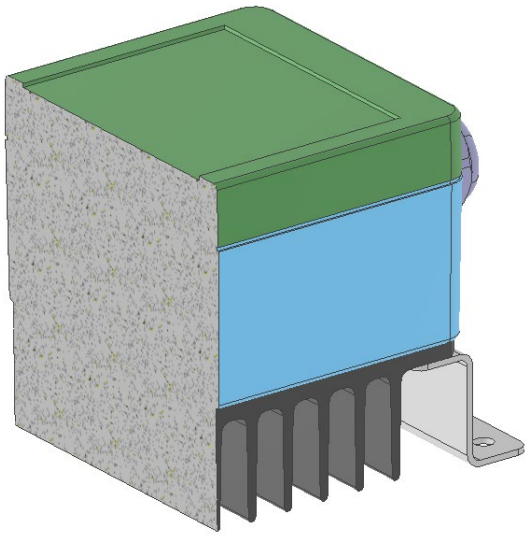
PRESERVE AND FEATURES, LIKE THESE MOUNTING HOLES, FOR THE FINAL COMPONENT

Shrinkwrapping removes internal components and fills hollow cavities

After selecting only the required components and removing unnecessary features, further Shrinkwrapping options can help remove any components we missed and eliminate hollow cavities. This will further simplify the model and help protect intellectual property.



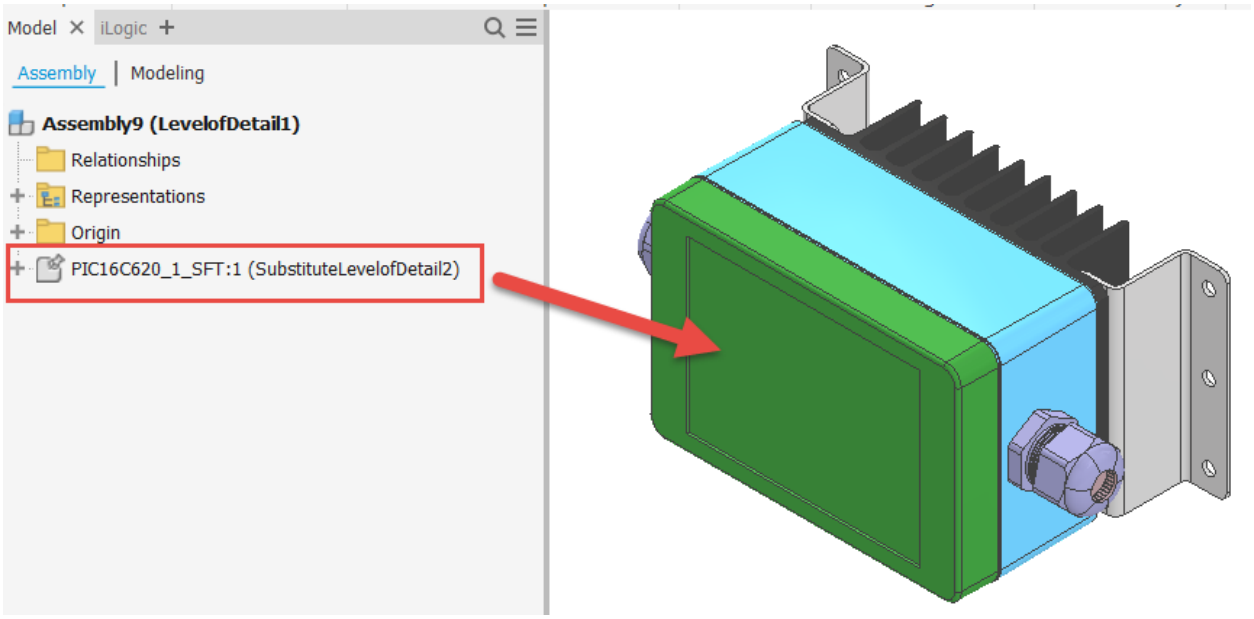
ON THE CREATE TAB NAME THE FINAL COMPONENT, REMOVE ANY LAST PARTS AND FILL ALL INTERNAL VOIDS



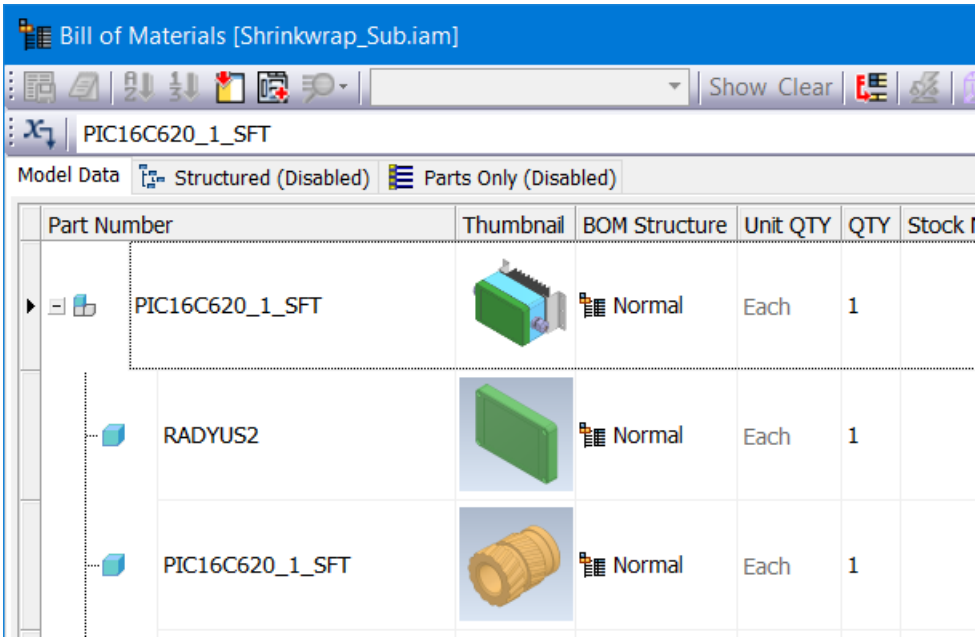
SECTION CUT OF THE FINAL COMPONENT SHOWS THAT THE INTERNAL VOIDS HAVE BEEN FILLED SOLID

Shrinkwrap Substitutes improve downstream subassembly performance

The Shrinkwrap part is an efficient model that will be used to create the Revit equipment. However, before we move on to the Revit conversion, I'd like to point out that the Shrinkwrap command can also benefit the internal assembly process. The Shrinkwrap Substitute level of detail uses the Shrinkwrap part as the assembly representation but does bring along the full subassembly BOM. This allows us to build larger, more efficient assemblies and we could even build larger equipment models that can be converted into a Revit ready format.



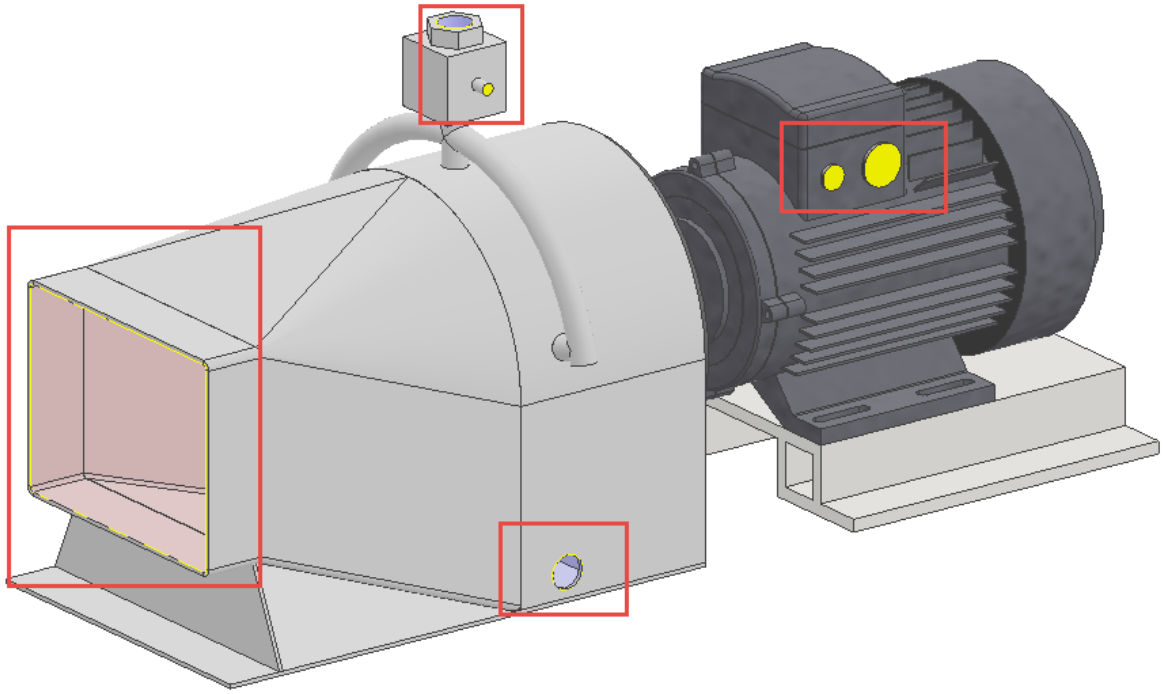
PLACING A SUBASSEMBLY WITH A SUBSTITUTE LEVEL OF DETAIL UTILIZES THE SHRINKWRAP PART



EVEN THOUGH THE SHRINKWRAP PART IS USED THE FULL SUBASSEMBLY BOM IS AVAILABLE

Custom BIM appearances pass through the Shrinkwrap

While I've demonstrated there are several benefits to utilizing the Shrinkwrap command, the primary reason that it benefits this process of converting Inventor equipment to Revit models is that the custom BIM appearances that were applied, carry through to the final part.



SHRINKWRAP PART WITH BIM SURFACES PRESERVED

iLogic – the Magic that Brings it All Together

All the work up to this point has been to lay the groundwork for iLogic to come onto the scene and dazzle us with the automation component. For those unfamiliar with iLogic, it is a VB.NET based coding module that resides completely within Inventor. Many Inventor designers use iLogic for what's called "Rules-Based" design, where specific conditions are set that impact the geometry of a component. At an even more advanced level, the Inventor API (Application Programming Interface, the "guts" of Inventor) can be utilized to generate new parts and geometry. In this process, I'll use iLogic to do the following:

- Navigate through the model browser using the Inventor API to launch and select specific items in the BIM environment.
- Create BIM Connectors on the fly
- Utilize a control rule to run other rules in an exact sequence
- (Optional) Export the final BIM ready Revit model

Navigating the Model Browser and working with nodes

To create the BIM Connectors, we must first switch to the BIM Environment and expand the BIM browser nodes to access each class of connector that we wish to create. Please note that only some of the most relevant snippets will be listed in this section, but the full iLogic rules will be posted in Appendix B.

```
'Enter the BIM Content environment
ThisApplication.CommandManager.ControlDefinitions.Item("AEC_Exchange:Environm
ent").Execute()

'Access the Browser and Activate the "Bim Content" Browser Pane
Dim oPartDoc As PartDocument = ThisDoc.Document
Dim bps = oPartDoc.BrowserPanes

Dim bimContentPane As BrowserPane =
oPartDoc.BrowserPanes.Item("AEC_Exchange:Browser")
```

THE CODE ABOVE ACCESSES THE BIM ENVIRONMENT VIA THE COMMAND MANAGER AND THE BIM BROWSER

```
'Set the Connection Counters based on the results from the Solid_Faces and
the current counts from the BIM Browser Nodes
Dim systemNode As BrowserNode = bimContentPane.TopNode.BrowserNodes.Item("MEP
System Connections")

Dim electicalNode As BrowserNode = systemNode.BrowserNodes.Item("Electrical")
                electicalNode.Expanded = True

Dim oElecConnCount As Integer = electicalNode.BrowserNodes.Count
```

THIS CODE SNIPPET ACCESSES THE TOP NODE OF THE BROWSER, EXPANDS THE ELECTRICAL NODE AND EXTRACTS THE COUNT OF EXISTING ELECTRICAL CONNECTORS

Creating BIM connectors on the fly

To build the connectors, we must first identify which surfaces are being utilized by the connectors and see if they line up with the custom BIM appearances we created earlier.

```
'Define the Appearance variables
Dim oElecAppearance As Asset = oPartDoc.Assets.Item("Connection_Electrical")
Dim oDuctAppearance As Asset = oPartDoc.Assets.Item("Connection_Duct")
Dim oPipeAppearance As Asset = oPartDoc.Assets.Item("Connection_Piping")
```

DEFINE INTERNAL VARIABLES AND ASSIGN THEM TO THE CUSTOM BIM APPEARANCE NAMES

```
'Set the Connection Counters
Elec_Count = oElecConnCount
Duct_Count = oDuctConnCount
Pipe_Count = oPipeConnCount
```

DEFINE THE COUNTER VARIABLES, WHICH ARE SET TO ZERO BY DEFAULT IN ANOTHER PORTION OF THE RULE

The next portion of the code will cycle through each boundary patch surface and see that surface's appearance matches up with one of the custom BIM appearances. If the appearances match, then that patch is identified and used as the basis for creating the BIM connector. Based on the current connector count, a new connector is added and named "Elec_Conn_#".

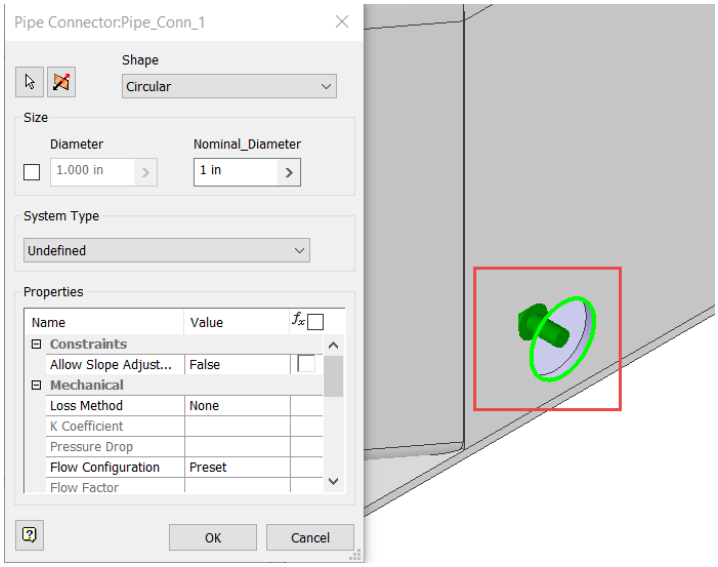
```
'Check through each surface in the part and see if it uses the special
appearances
For Each oWrkSurf As WorkSurface In oPartDoc.ComponentDefinition.WorkSurfaces
        For Each oSrfBody As SurfaceBody In oWrkSurf.SurfaceBodies
                For Each oFace As Face In oSrfBody.Faces

                        If oFace.Appearance.DisplayName =
oElecAppearance.DisplayName Then

                                Dim Elec_Coll As ObjectCollection
                                Elec_Coll =
ThisApplication.TransientObjects.CreateObjectCollection

                                Elec_Coll.Add(oFace)

                                'Create Electrical Connector Definition
                                Dim oElecConn As
BIMElectricalConnectorDefinition

                                oElecConn =
oPartDocBIM.Connectors.CreateElectricalConnectorDefinition(Elec_Coll)

                                'Add the Electrical BIM Connection
                                Elec_Count = Elec_Count + 1
                                Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oElecConn)
```

```
                                    con.Name = "Elec_Conn_" & Elec_Count
'oPartDocBIM.Connectors.Count
```

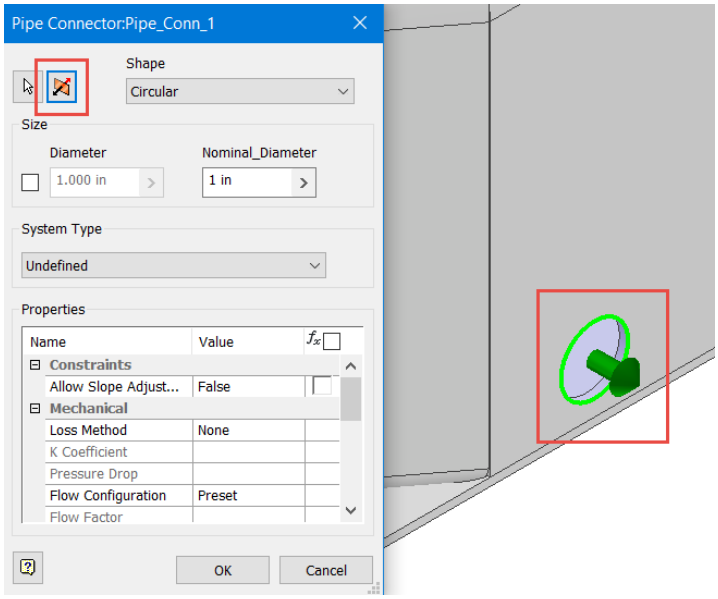
CYCLE THROUGH EACH BOUNDARY PATCH SURFACE AND CHECK TO SEE IF IT MATCHES THE ELECTRICAL CONNECTOR APPEARANCE AND THEN CREATE A NEW CONNECTOR

```
'Since using a surface versus a solid face, must flip the direction
con.Definition.ReverseDirection
```

IF THE FACE IS A BOUNDARY PATCH THEN THE CONNECTOR DIRECTION NEEDS TO BE FLIPPED



BIM CONNECTOR ARROWS FOR BOUNDARY PATCHES AND OTHER PURE SURFACE POINT "INSIDE" THE MODEL



BIM CONNECTOR ARROWS MUST BE REVERSED TO POINT "OUTSIDE" THE MODEL

The BIM connectors can be made automatically without any human interaction, which will require more work upfront, but will reduce the amount of time spent by the end user. However, I've decided to launch the Edit Connector command for each connector to allow users to verify or modify any information or the direction of the connector. To initialize the Edit Connector command, we must access the Inventor API Control Definitions

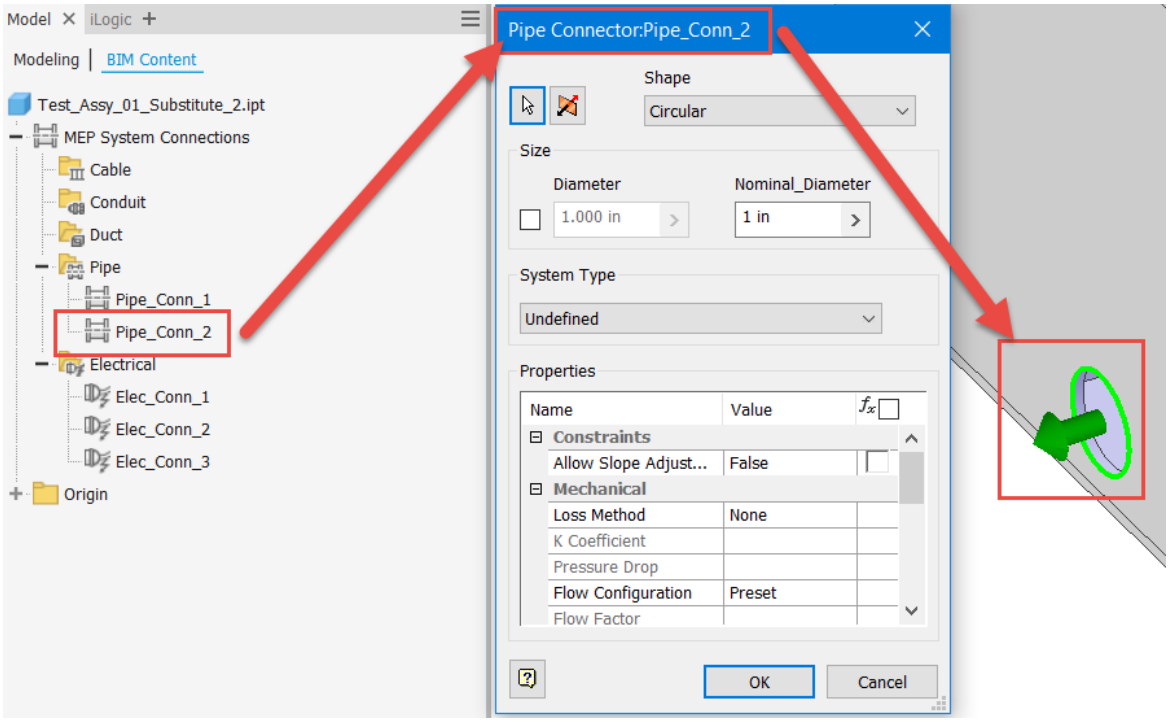
```
' Get the collection of control definitions.
Dim oControlDefs As ControlDefinitions =
ThisApplication.CommandManager.ControlDefinitions

Dim oControlDef As ControlDefinition =
oControlDefs.Item("AEC_Exchange:Command:Connector:Edit")
```

ACCESS THE INVENTOR API CONTROL DEFINITIONS AND CREATE A SPECIFIC CONTROL DEFINITION FOR THE BIM CONNECTOR EDIT COMMAND

Once the connector has been created, the specific browser node is selected and the Edit Connector command control definition is activated.

```
'Activate the current electrical node so characteristics can be modified, if
desired.
Dim oConnNode As BrowserNode = electicalNode.BrowserNodes.Item(con.Name)
oConnNode.DoSelect()
oControlDef.Execute()
```

FIND AND SELECT THE CURRENT BROWSER NODE THEN ACTIVATE THE EDIT COMMAND



EACH BIM CONNECTOR NODE IS SELECTED AND THE EDIT COMMAND LAUNCHED

Exporting a Revit family

After configuring the settings for the BIM model and building all the connectors, the last step is to export the model as a Revit family. For this presentation, I've decided to name the Revit family the same name as the Inventor model and export it to the same location. To accomplish this, the original filename and file path must be extracted from the Inventor model.

```
'Extract the current filename and parse Out the .ipt file extension
Dim Comp_Name As String = oPartDoc.DisplayName
Dim Comp_Name_Count As Integer = Comp_Name.Length
Dim Final_Comp_Name As String = Left(Comp_Name, Comp_Name_Count - 4
```

USE TEXT STRING FUNCTIONS TO PARSE THE FILENAME AND REMOVE THE INVENTOR FILE EXTENSION

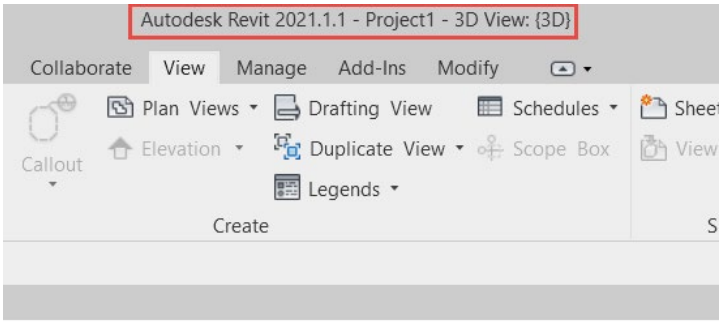
```
'Set up the file path from the original host file
Dim File_Path_Length As Integer = oPartDoc.FullFileName.Length
Dim File_Path As String = Left(oPartDoc.FullFileName, File_Path_Length -
Comp_Name_Count)
```

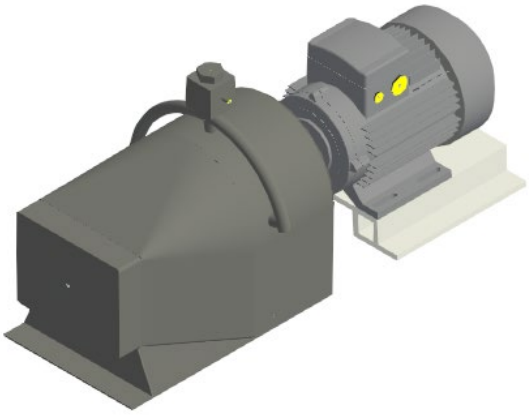
EXTRACT THE FILE PATH FROM THE INVENTOR FILE AND REMOVE THE FILENAME AND EXTENSION FROM THE STRING

Define variables to access the Inventor API and execute the Revit export

```
'Define the Part Document to prepare for BIM Revit Export
Dim oPartDoc As PartDocument = ThisDoc.Document

'Define the BIM Component
Dim oPartDocBIM As BIMComponent = oPartDoc.ComponentDefinition.BIMComponent

'Create the Revit Family using the extracted file path and formed Revit file
name
oPartDocBIM.ExportBuildingComponent(File_Path & Final_Comp_Name & ".rfa")
```

ACCESS THE BIM COMPONENT DEFINITION AND UTILIZE THE EXPORTBUILDINGCOMPONENT METHOD





RESULT OF THE REVIT EXPORT

INSERT IMAGE OF REVIT CONNECTIONS BEING MADE

REVIT CONNECTIONS BEING MADE TO THE EXPORTED REVIT MODEL

Using a control rule to sequence operations

To cycle through all the solid faces and boundary patch surfaces that have the custom BIM appearances assigned, I wrote two rules; one to handle the solid faces and the other the boundary patch surfaces. I also built a rule that would automatically export the model as a Revit family. I could have written one rule that would have attempted to perform all those actions, but that rule would have been quite lengthy and complicated.

A great way to stay organized and allow each rule to perform a specific function is to create a central rule that calls the other rules in the sequence that you desire. Below is a sample rule that I've written to first check all the solid faces for the custom BIM appearances, then check the boundary patch surfaces and finally to publish the Revit family.

```
'Check all the Solid Faces for the BIM Custom Appearances and create
connectors
iLogicVb.RunRule("Jelte_Rule_Solid_Faces")

'Check all the Surface Boundary Patches for the BIM Custom Appearances and
create connectors
iLogicVb.RunRule("Jelte_Rule_Surfaces")
```

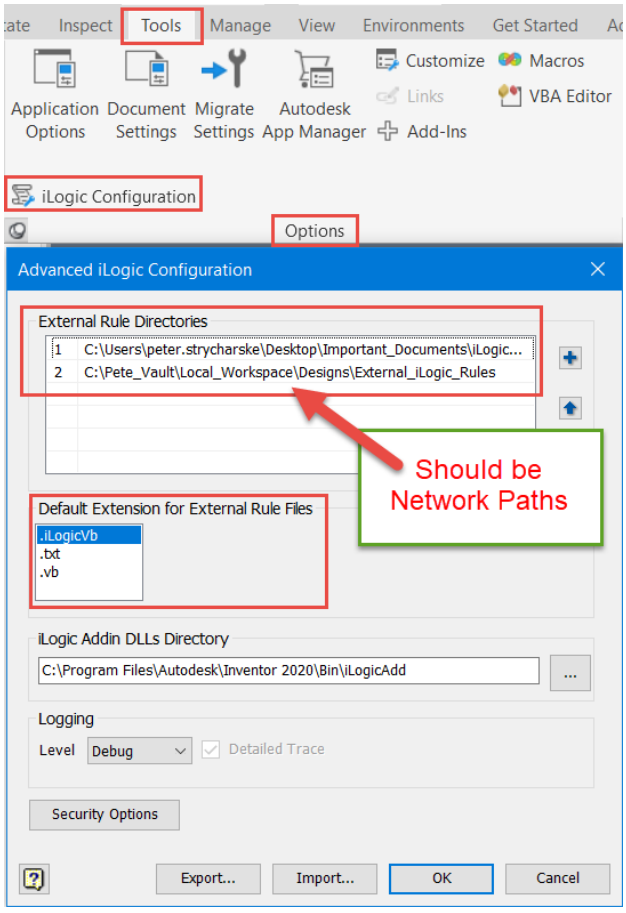
```
'Export the model as a Revit family with the same root filename in the same
file location
iLogicVb.RunRule("Create_Revit_Family")

iLogicVb.UpdateWhenDone = True
```

SIMPLE INTERNAL ILOGIC CENTRAL CONTROL RULE USED TO LAUNCH OTHER RULES IN SEQUENCE

These rules don't even have to be local to the component and can be launched as external iLogic rules, which can be an even more efficient way to utilize automation. To utilize external iLogic rules, the following steps must be taken:

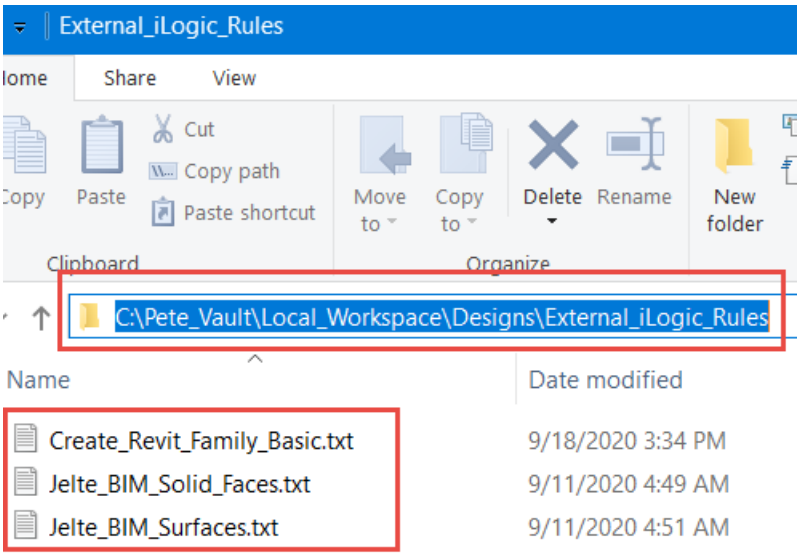
- Configure the external iLogic locations.
- Create / copy the rules in the external location.
- Modify the iLogic central rule to utilize the other external iLogic rules.

To properly configure the external iLogic rule locations, you should use a network path, as I note in the image below. Please note, since I'm working on my own for this project, I am using "local" paths.



ENSURE THE ILOGIC CONFIGURATION SETTINGS ARE UTILIZING NETWORK PATHS

Next, we must place the rules in the proper locations we specified as External Rule Directories.



EXTERNAL ILOGIC RULES SAVED AS TEXT FILES IN ONE OF THE EXTERNAL RULE DIRECTORIES

Finally, the code must be modified slightly to take advantage of the external rules.

```
'Check all the Solid Faces for the BIM Custom Appearances and create
connectors
iLogicVb.RunExternalRule("Jelte_BIM_Solid_Faces")

'Check all the Surface Boundary Patches for the BIM Custom Appearances and
create connectors
iLogicVb.RunExternalRule("Jelte_BIM_Surfaces")

'Export the model as a Revit family with the same root filename in the same
file location
iLogicVb.RunExternalRule("Create_Revit_Family_Basic")

iLogicVb.UpdateWhenDone = True
```

MODIFIED CENTRAL CONTROL RULE TO USE OTHER EXTERNAL ILOGIC RULES

IMPORTANT NOTE: Complete versions of the iLogic code can be found in Appendix B.

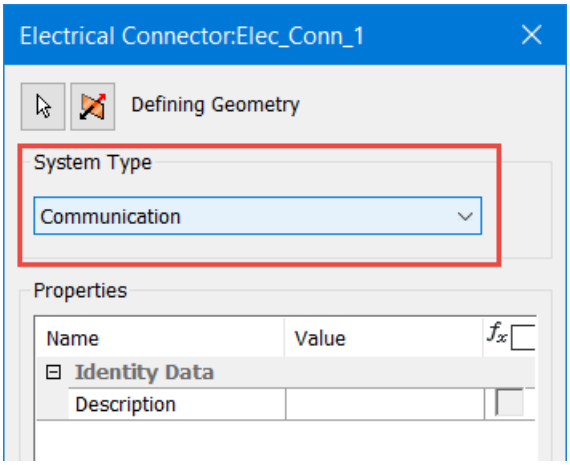
Advanced topic – Information-rich appearance names

Up to this point, I have been demonstrating the general approach to automating the placement of BIM connectors based on specialized appearances applied to specific surfaces. While this will indeed greatly increase the efficiency for the end designers who are configuring these models, there is another level of efficiency that can be achieved.

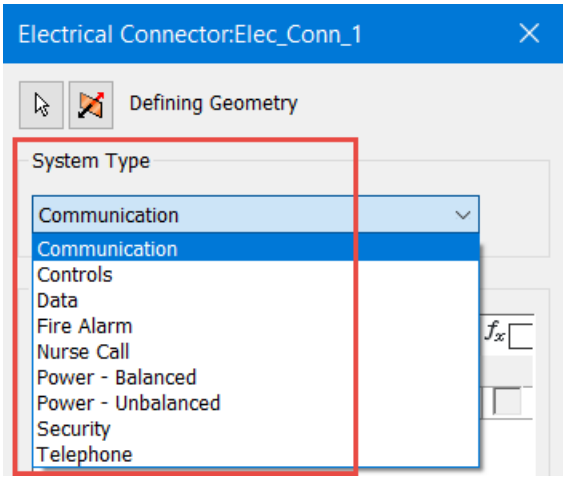
If the custom appearances have even more information embedded in the name, we can utilize that information to prepopulate the BIM connector information. Additionally, the iLogic rules must be adapted for these new characters.

Utilize character coding to populate BIM Connector data

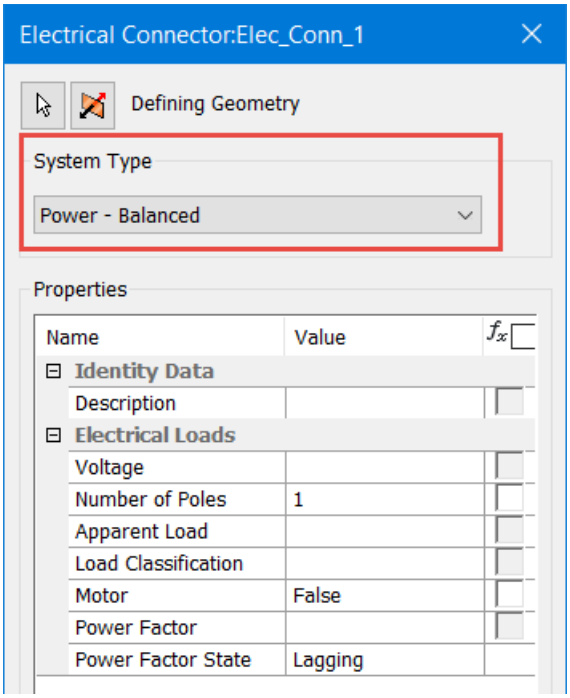
When building the BIM connectors, presetting the appearance and surface data is only a portion of the intelligence that can be placed inside of a connector. If we look at an Electrical BIM connector, for example, we have several different System Types that we can specify, and those Types then require specific subgroups of information.



THE COMMUNICATION ELECTRICAL CONNECTOR ONLY HAS A DESCRIPTION FIELD



THE LIST OF ALL ELECTRICAL CONNECTOR SYSTEM TYPES



A BALANCED POWER CONNECTOR HAS SEVERAL FIELDS OF DATA TO UTILIZE

This information may be intimidating for an end designer, like me, if they are not familiar with the specifications for a component or the code requirements for a given project. To simplify this process and allow a greater number of designers to utilize these components, we can implement a character-based appearance naming scheme, which can help with these types of operations. For an example, let's examine the Electrical connector possibilities.

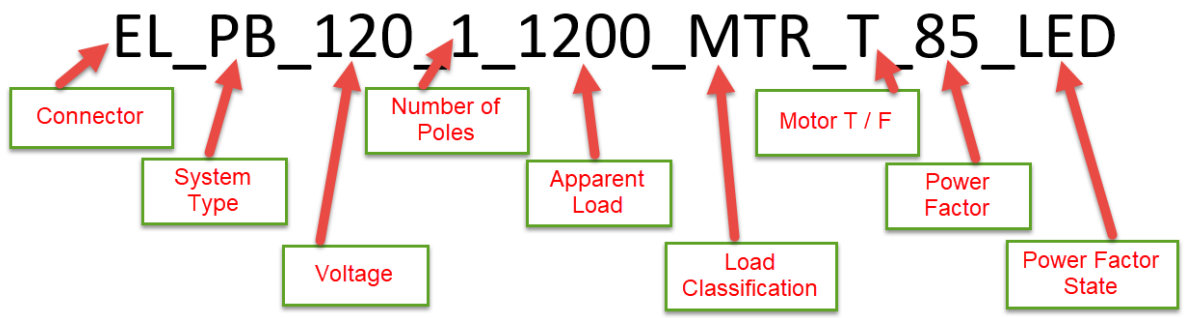
For Electrical Connectors we see there are 9 different System Types. Most of these only list a description, so this won't be as intimidating as it first seems. Only the "Power – Balanced" and "Power – Unbalanced" System Types are information rich. If we can break down the name slightly into different character groupings, then iLogic can be used to populate the BIM connector information. Please see the following example for distinguishing Electrical connectors.

Connector Types:
EL - Electric
PI - Pipe
DU - Duct
CO - Conduit
CA - Cable Tray

Electrical System Types:
CM - Communication
CN - Controls
DA - Data
FI - Fire Alarm
NU - Nurse Call

PB - Power Balanced
PU - Power Unbalanced
SE - Security
TE - Telephone

If the Power Balanced is selected, then a character structure can be used to populate the rest of the connector data. Below is a sample string:



SAMPLE CHARACTER STRING TO POPULATE THE POWER – BALANCED ELECTRICAL CONNECTOR

Another example would be a “Controls” System Type, which is a less substantial string example

SAMPLE CHARACTER STRING TO POPULATE THE CONTROLS ELECTRICAL CONNECTOR

iLogic rule adaption

If the surfaces utilize more information rich appearance names, the iLogic rules must be adapted to accommodate and parse through this data. The results will be worth it, so let’s get to work.

The surfaces will be verified just as before, but we must first check the first three characters to verify which BIM connector type is being utilized. I’m using three characters because I don’t want the special BIM character code to get mixed up with an actual finish, like “Elf Green” (it’s a real color, I looked it up). If we utilize a standard naming convention for the other appearance names, there should never be an issue. I will highlight the iLogic rule adaptions for this approach, but the balance of the code will remain the same. A full version of this sample code is listed in Appendix B.

```
'Define a variable to determine which type of Connector Surface is present
Dim BIM_Conn_Type As String = Left(oFace.Appearance.DisplayName, 3)
'Determine if this is an Electrical Connector
If BIM_Conn_Type = "EL_" Then
```

UTILIZE THE "LEFT" FUNCTION TO ACCESS THE 3 LEFTMOST CHARACTERS AND DETERMINE IF THIS IS AN ELECTRICAL CONNECTOR

If this is indeed an Electrical connector, then we'll have to verify the System Type. To do this, we'll have to grab a specific portion of the appearance name. The Mid function allows us to start at specific character and then analyze a section of the string. In this sample code, I'll also introduce the Select Case technique.

```
'Create a variable to determine what kind System Type is being utilized
Dim Elec_Sys_Type As String = Mid(oFace.Appearance.DisplayName, 4, 2)
'Determine which Electrical System Type is being used and adjust accordingly
Select Case Elec_Sys_Type
```

UTILIZE THE "MID" FUNCTION TO ANALYZE THE TEST STRING, STARTING WITH THE 4TH CHARACTER FROM THE LEFT AND LOOKING AT THE NEXT 2 CHARACTERS. THIS WOULD RETURN THE "PB" FROM THE NAME "EL_PB…"

The Select Case technique is similar to an If / Then approach but only checks to see if the specified condition is met. For example, when the System Type is "PB" (Power – Balanced), then different operations are fired, versus when the System Type is "CN" (Controls).

```
Case "PB"
        oElecConn.SystemType =
BIMElectricalSystemTypeEnum.kPowerBalancedElectricalSystemType

        oElecConn.Voltage = Mid(oFace.Appearance.DisplayName, 7, 3)

        oElecConn.NumberOfPoles = Mid(oFace.Appearance.DisplayName, 11, 1)

        oElecConn.ApparentLoad = Mid(oFace.Appearance.DisplayName, 13, 4)

        oElecConn.LoadClassification = Mid(oFace.Appearance.DisplayName, 18,
3)

        If Mid(oFace.Appearance.DisplayName, 22, 1) = "T" Then
                oElecConn.HasMotor = "True"
        Else
                oElecConn.HasMotor = "False"
        End If

        oElecConn.PowerFactor = "." & Mid(oFace.Appearance.DisplayName, 24, 2)

        If Right(oFace.Appearance.DisplayName, 3) = "LED" Then
                oElecConn.PowerFactorState = kLeadingPowerFactorStateType
        Else
```

```
                oElecConn.PowerFactorState = kLaggingPowerFactorStateType
        End If
```

THE OPERATIONS LAUNCHED WHEN THE CASE IS "PB" POWER – BALANCED

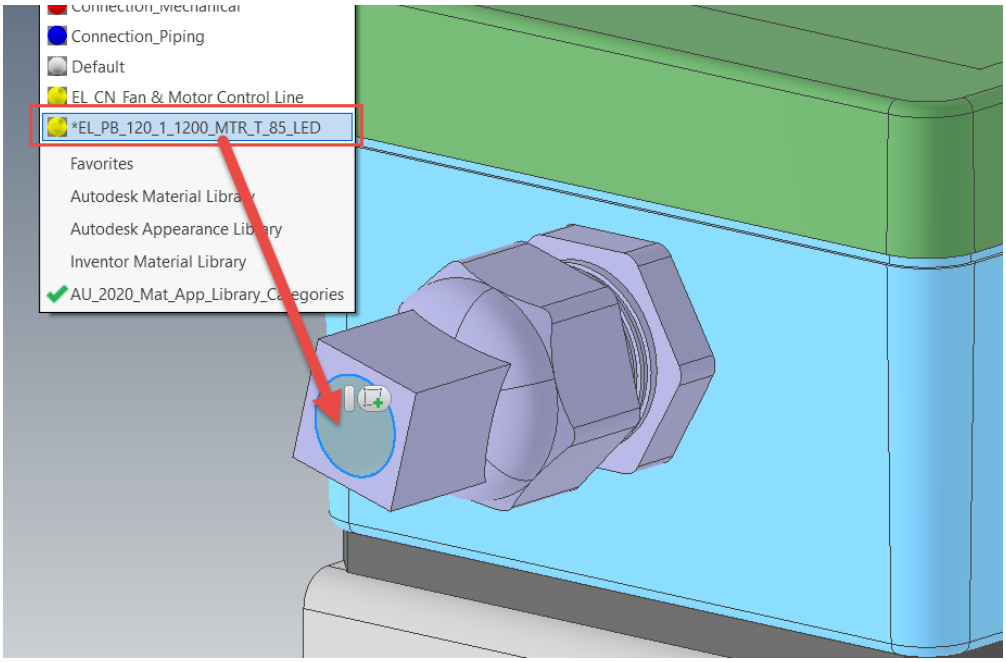
```
Case "CN"
        oElecConn.SystemType =
BIMElectricalSystemTypeEnum.kControlsElectricalSystemType

'Determine the length of the overall text string and then we can remove
'the first six characters "EL_CN_" and use the rest in the Description
Dim oStringLength As Integer = Len(oFace.Appearance.DisplayName)

'Use the overall string length minus the first 6 characters for the
Description
oElecConn.Description = Mid(oFace.Appearance.DisplayName, 7, oStringLength -
6)

End Select
```

THE OPERATIONS LAUNCHED WHEN THE CASE IS "CN" CONTROLS

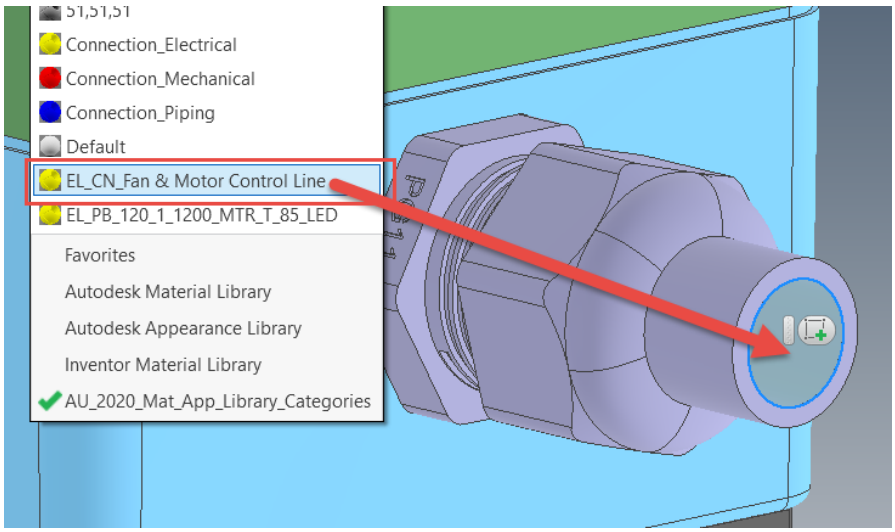
The general format is as follows:

```
Select Case (Condition)
      Case (Value)
End Select
```

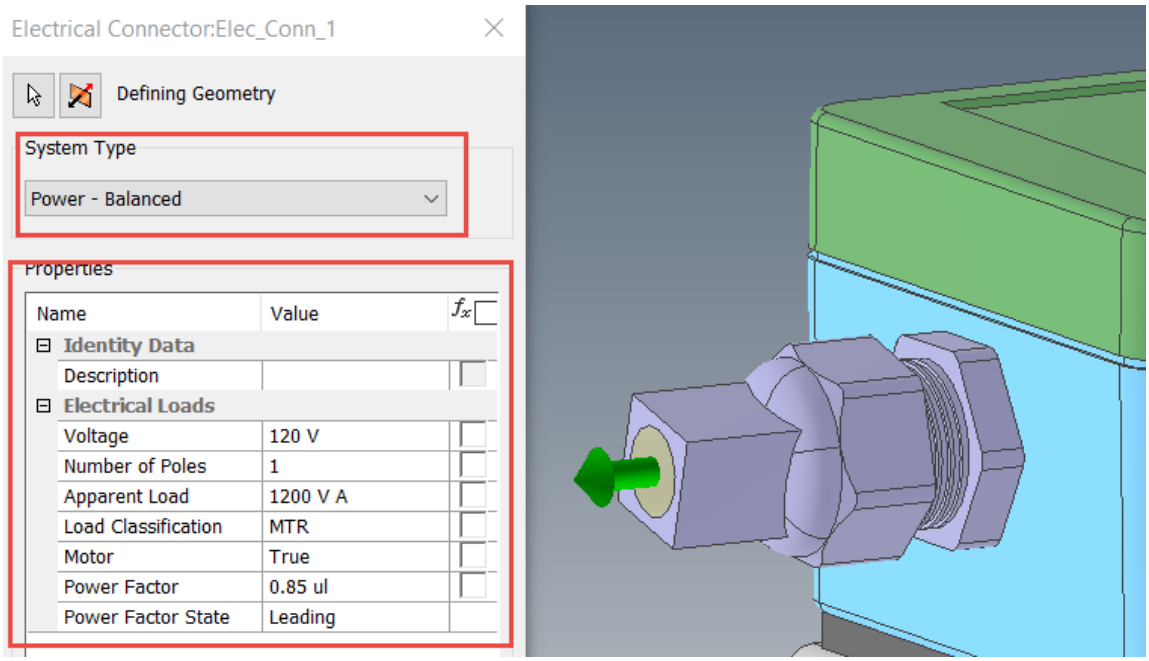
This is great way to check a large range of conditions without the extra coding of the If / Then conditions. Since there are nine System Types for the Electrical, the Select Case technique is a great fit. Running the rule on the Shrinkwrapped Controller yields the results below:



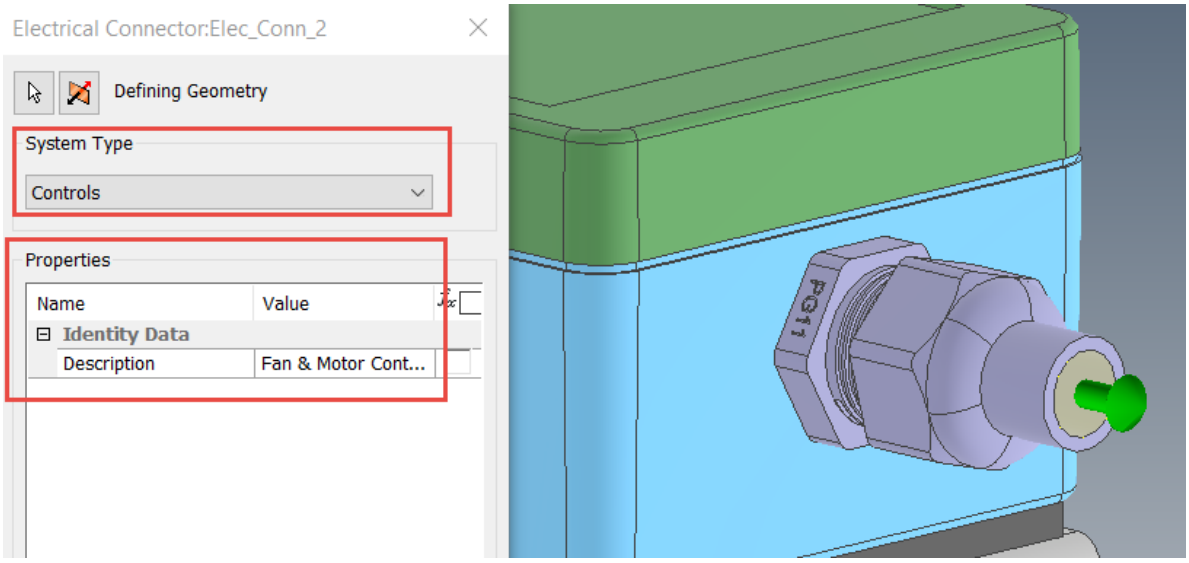
POWER CONNECTION SURFACE ON THE SHRINKWRAPPED CONTROLLER



CONTROLS CONNECTION SURFACE ON THE SHRINKWRAPPED CONTROLLER



THE BIM CONNECTOR RESULTS FOR THE POWER CONNECTION



THE BIM CONNECTOR RESULTS FOR THE CONTROLS CONNECTION

In summary, if the surface appearance naming is specific enough, those appearances can be utilized to create information-rich BIM connectors and saving the end designers even more configuration time and expanding the number of users who can create configurations.

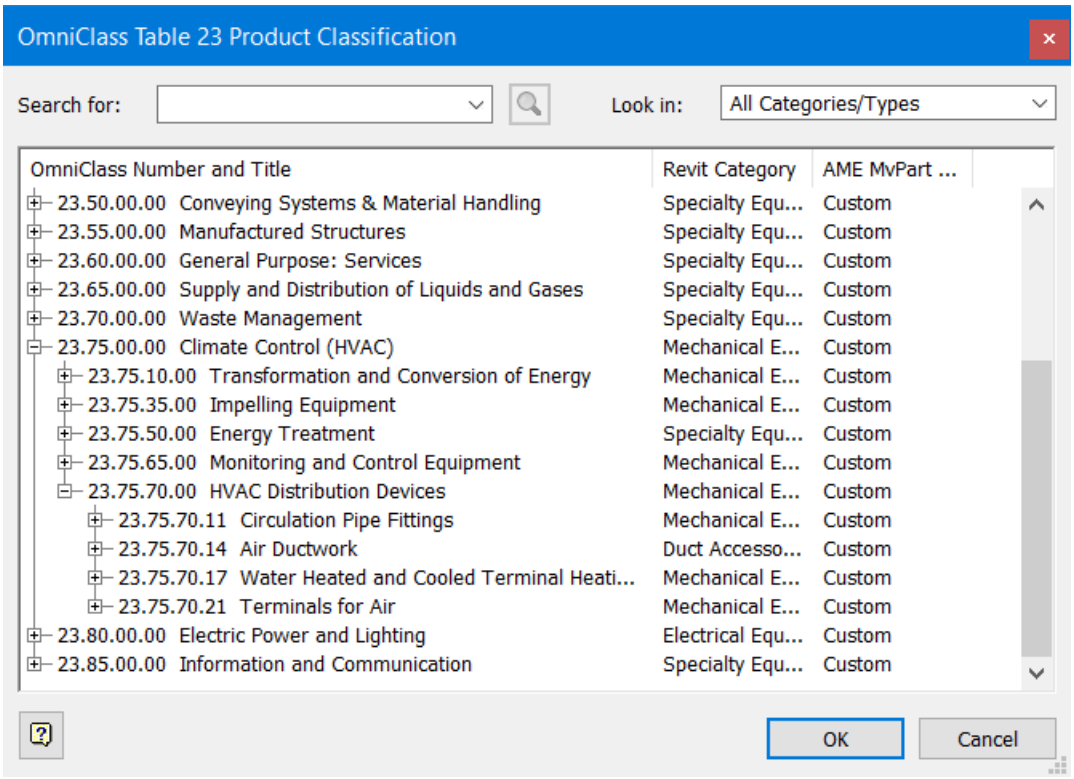
Future research (and things to watch out for)

No process is ever perfect, and this one is no exception, so there are some items that can be improved in future iterations of this workflow. Here is a list of the items that I'd like to address in the future:

- Efficient methodology for implementing the Omni Class
- Surface inconsistencies for non-native Inventor files
- Overcoming inconsistent connector direction issues
- Diagnosing and remedying Revit duct issues

Efficient methodology for implementing the Omni Class

The final piece of information involves the Author Building Components command and the information that can be placed within those fields, most notably the OmniClass. Omni Classifications help to organize information specific to the construction industry and as you can see below, these are quite expansive.



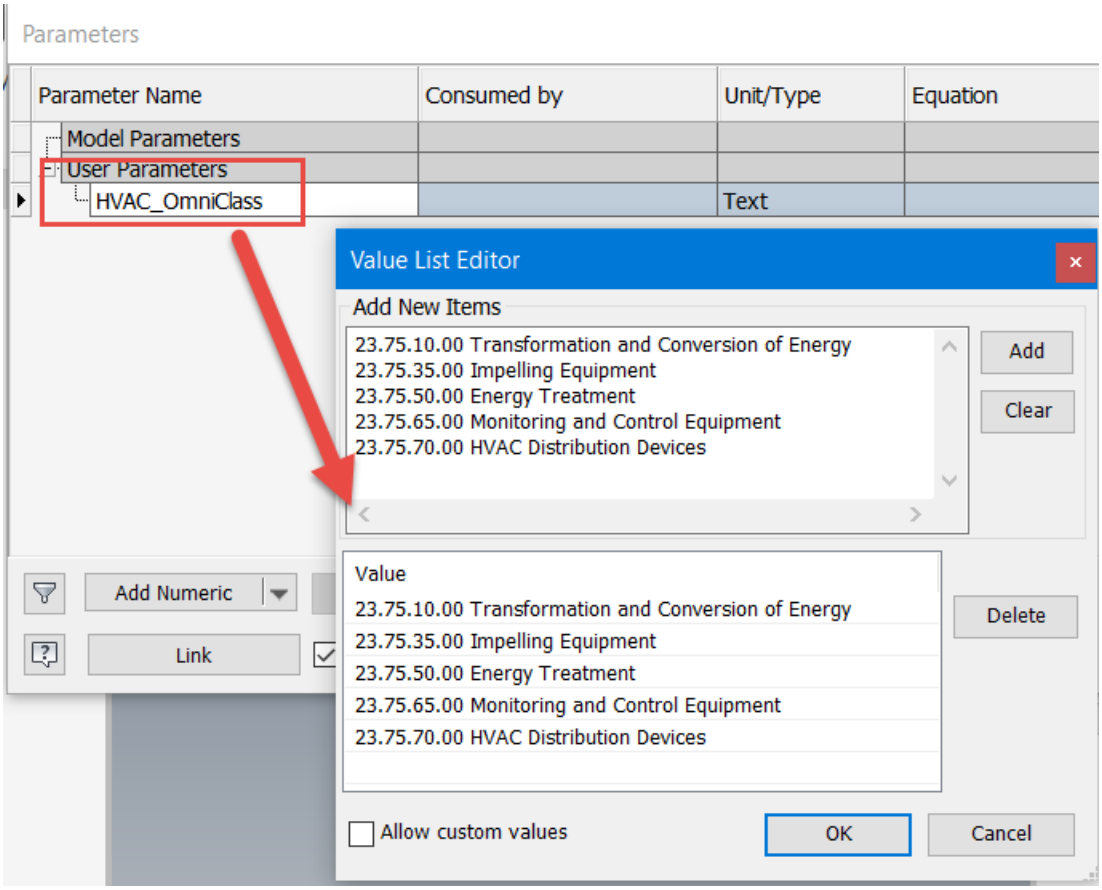
SAMPLE TABLE OF OMNICLASS CLASSIFICATION IN INVENTOR

Since there are so many potential OmniClass classifications to choose from, it is a difficult challenge to determine the best way to accomplish this goal. Currently, I'm leaning towards the following solution:

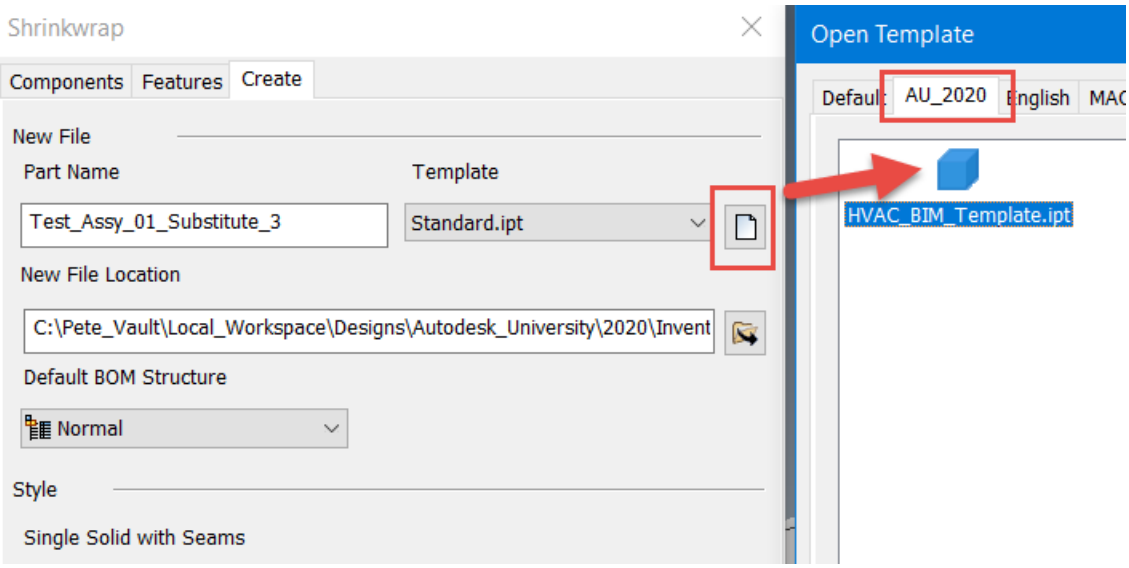
- Creating specific templates that have the relevant OmniClass classifications already built in (this could work for other properties besides OmniClass).
- Providing a parametric list for end users to choose from, depending on the design configuration.
- Utilize iLogic to properly populate the BIM component properties.

Create specific templates containing the relevant OmniClass classifications

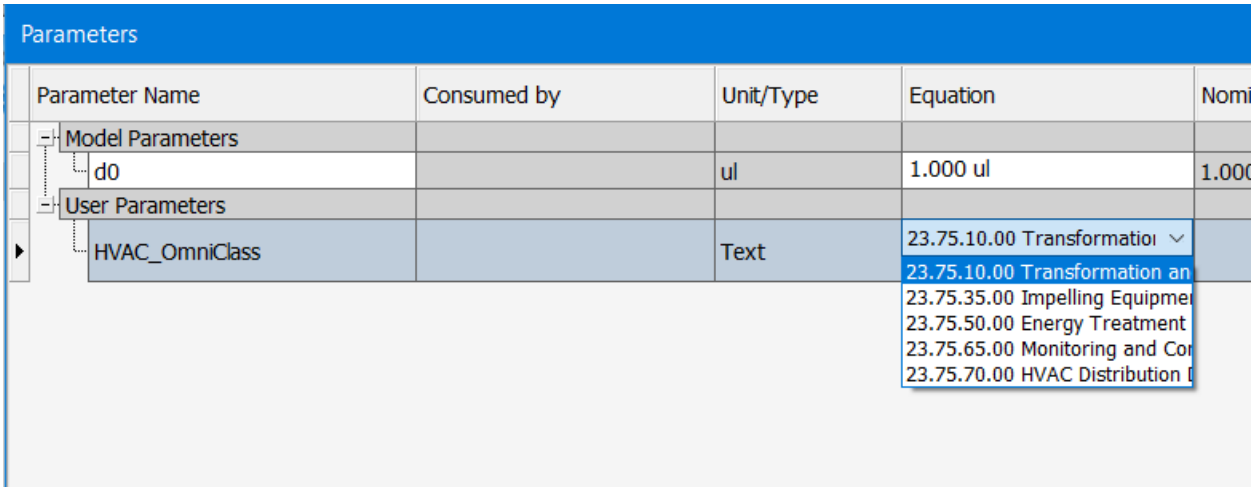
For certain types of equipment, the OmniClass list can be pared down dramatically and if placed within a specific template. This can make life much easier for the end users to properly populate critical component data. The most challenging part of the process would be choosing the proper template when creating the Shrinkwrap / Shrinkwrap Substitute.



INSIDE A SPECIFIC TEMPLATE FILE, CREATE A MULTI-VALUE PARAMETER WITH THE OMNICLASS POSSIBILITIES LISTED



UTILIZE THE EXACT TEMPLATE FOR THIS EQUIPMENT CLASS



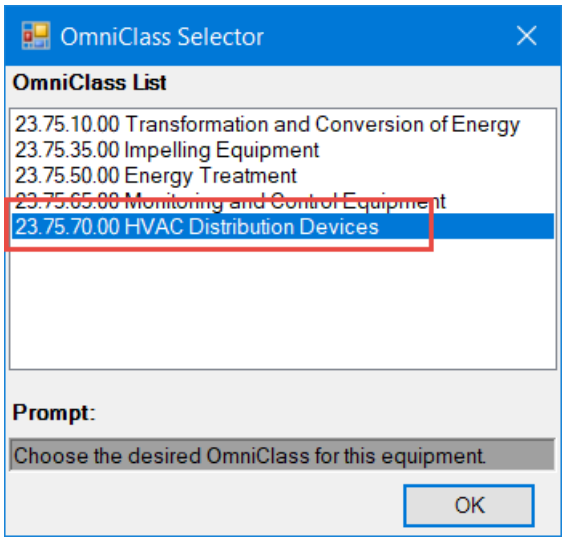
THE GENERATED SHRINKWRAP PART CONTAINS THE DESIRED OMNICLASS LIST

Provide a parametric list to for end users to choose from and apply properties

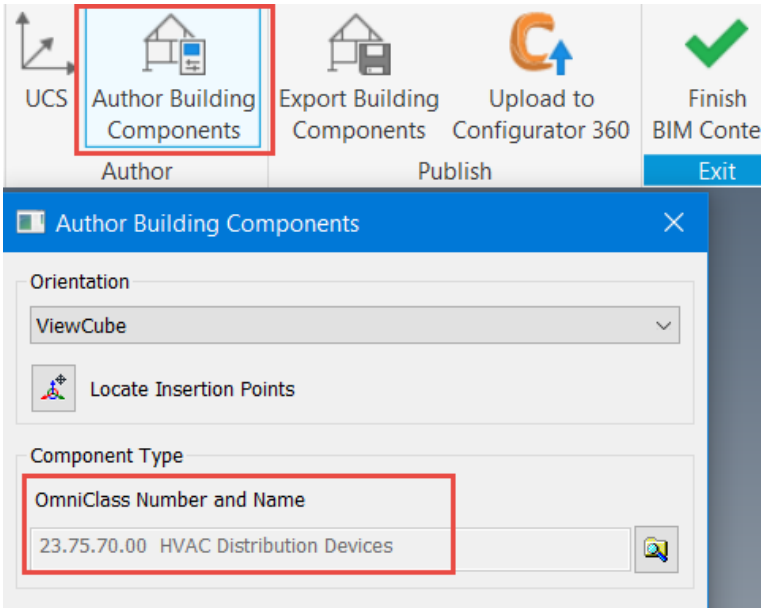
To utilize the existing iLogic methodology, the best approach would be to expose the list to the users as the other iLogic rules are building the BIM connectors. In the following example, I let the user choose from the multi-value parameter list and assign the selection to the OmniClass value.

```
'Pare down and automate the setting of the OmniClass
'Allow the user to access the HVAC OmniClass list and select the desired
classification
HVAC_OmniClass = InputListBox("Choose the desired OmniClass for this
equipment.", MultiValue.List("HVAC_OmniClass"), HVAC_OmniClass, Title :=
"OmniClass Selector", ListName := "OmniClass List")

'Push the OmniClass value to the BIM Component Properties
oPartDocBIM.ComponentDescription.ComponentType = Left(HVAC_OmniClass,11)
```

ACCESS THE HVAC_OMNICLASS PARAMETER FOR USER SELECTION AND ASSIGN THE NUMERIC CODE TO THE OMNICLASS FIELD



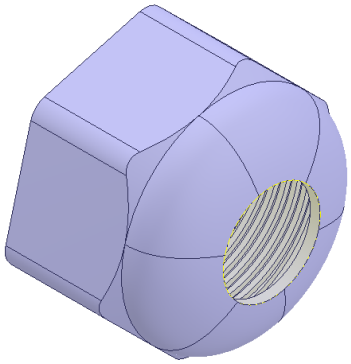
OMNICLASS SELECTION THE USER SEES BASED ON HVAC_OMNICLASS PARAMETER



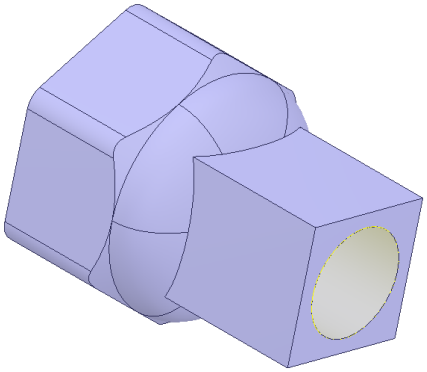
RESULTING OMNICLASS VALUE

Surface inconsistencies for non-native Inventor files

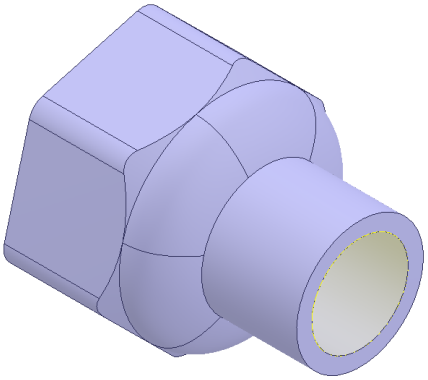
When working with imported STEP files, I came across scenarios where the iLogic rules wouldn't run on the native geometry. Thankfully, when I added Inventor features to the STEP files and then the iLogic rules ran without issue, so the solution isn't onerous. This limitation didn't appear with any of the Inventor models I worked with, so I'm confident this issue is limited to non-native files. See the following example, from the Shrinkwrapped model.



ORIGINAL IMPORTED AND CONVERTED STEP FILE COMPONENT (ILOGIC IS NOT FUNCTIONING)



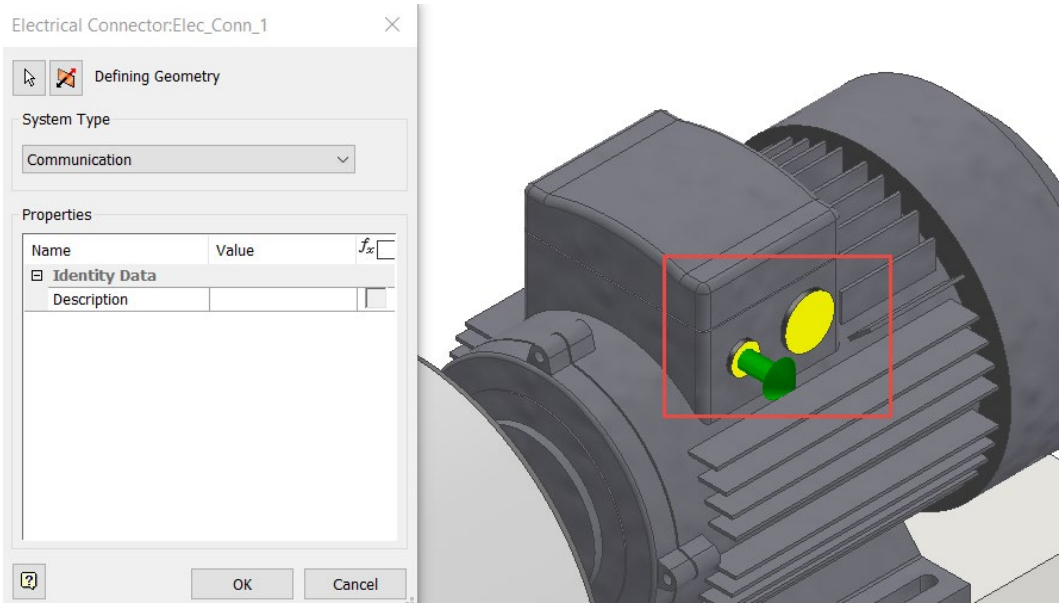
IMPORTED COMPONENT WITH INVENTOR ADDITION FOR POWER SUPPLY (ILOGIC IS FUNCTIONING)



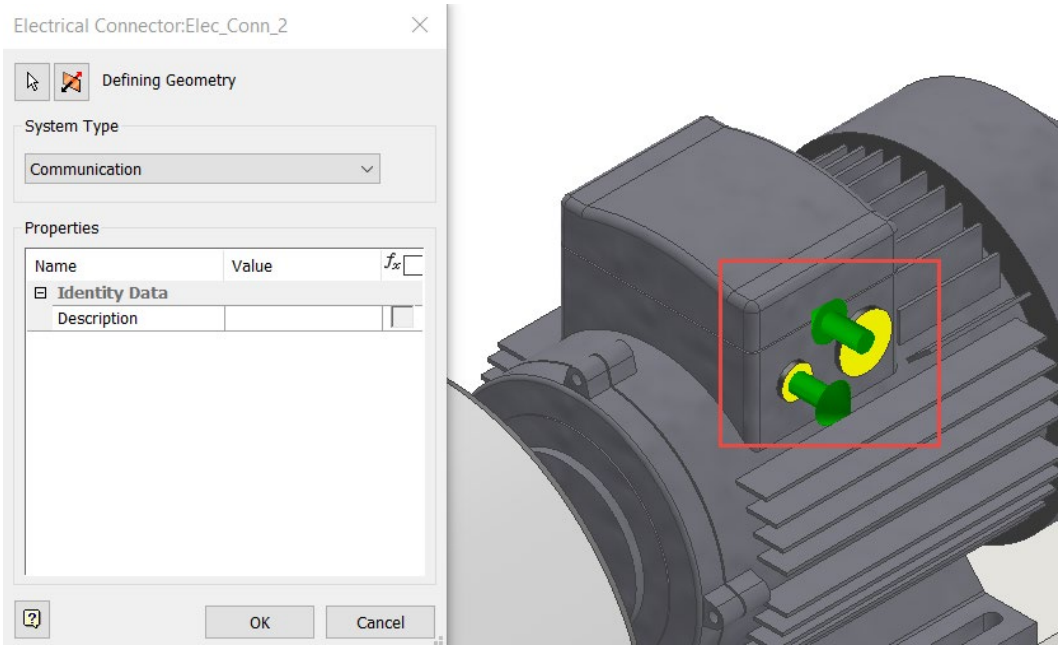
IMPORTED COMPONENT WITH INVENTOR ADDITION FOR CONTROLS (ILOGIC IS FUNCTIONING)

Inconsistent connector direction issues

I noticed on an Inventor model that I didn't create, some of directional arrows for the connectors, generated on solid faces, were pointed into the model, not away from the model as normal. I've not had this happen often, so I'm uncertain why this particular model had that issue. Everything else worked just fine, so I'm just careful to double check the final arrow directions. This is one of the reasons why I decided to launch the Edit Connector command for each connector, so the data AND the arrow direction can be verified.



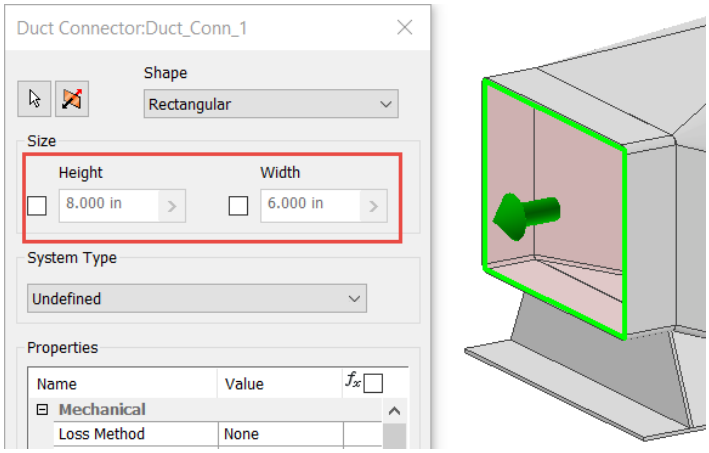
DIRECTION FOR THE FIRST MOTOR CONNECTOR ON A SOLID FACE



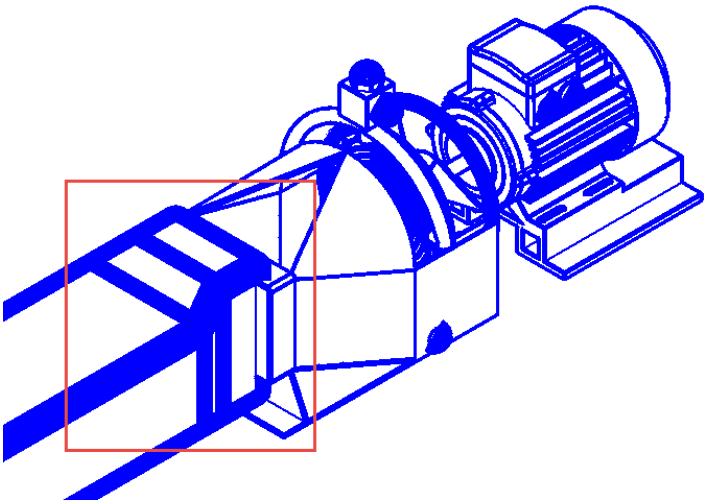
DIRECTION FOR THE SECOND MOTOR CONNECTION ON A SOLID FACE

Rectangular duct issues in Revit

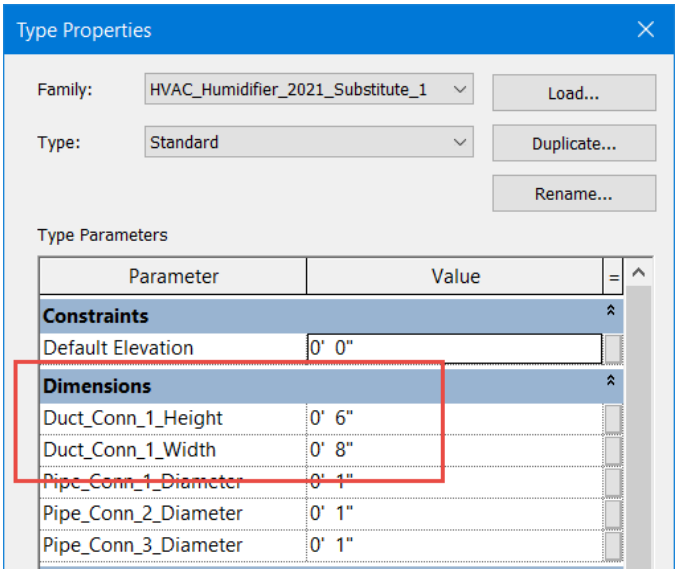
As I was building some of my sample models, I was noticing an odd behavior in Revit. When creating rectangular duct BIM Connectors in Inventor, the exported model was causing an odd transition piece to be created when connecting to duct work in Revit.



RECTANGULAR DUCT CONNECTION WITH REVERSED HEIGHT AND WIDTH



DUCT TRANSITION PIECE AUTOMATICALLY PLACED WHEN BUILDING A REVIT CONNECTION

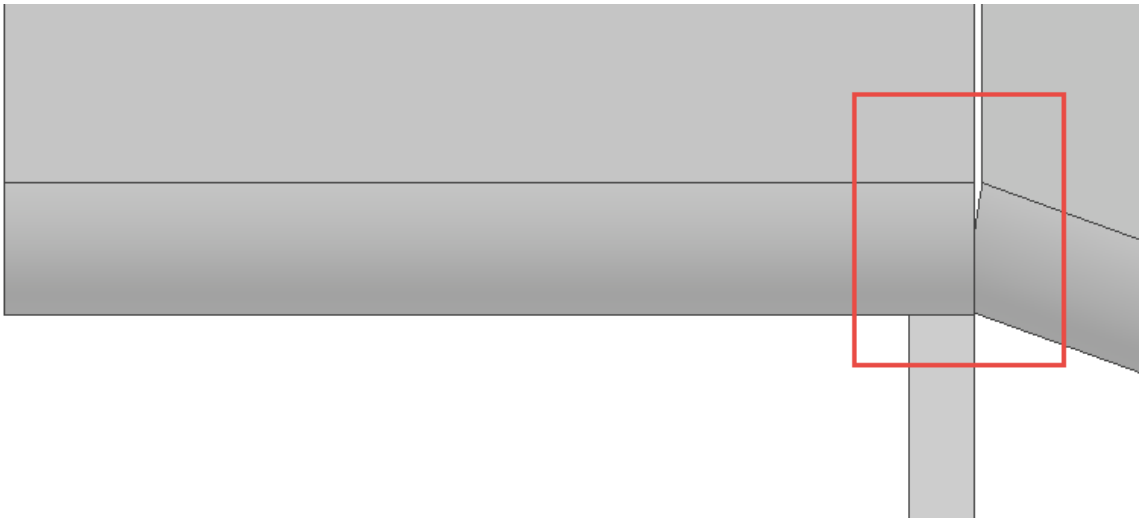


CONFIRMATION OF DUCT CONNECTION IN REVIT PROPERTIES

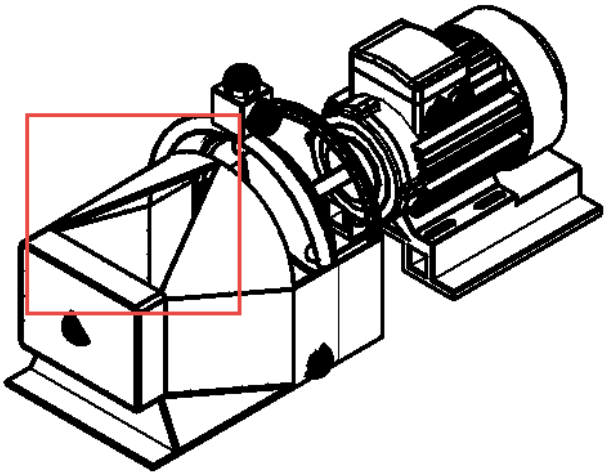
I brought this up to Autodesk and they've confirmed that something odd is happening and have opened a development team request to investigate the root cause and a possible resolution. For now, the only resolution is to edit the family in Revit, which will allow the rectangular duct to be corrected.

Complex Inventor faces missing in Revit

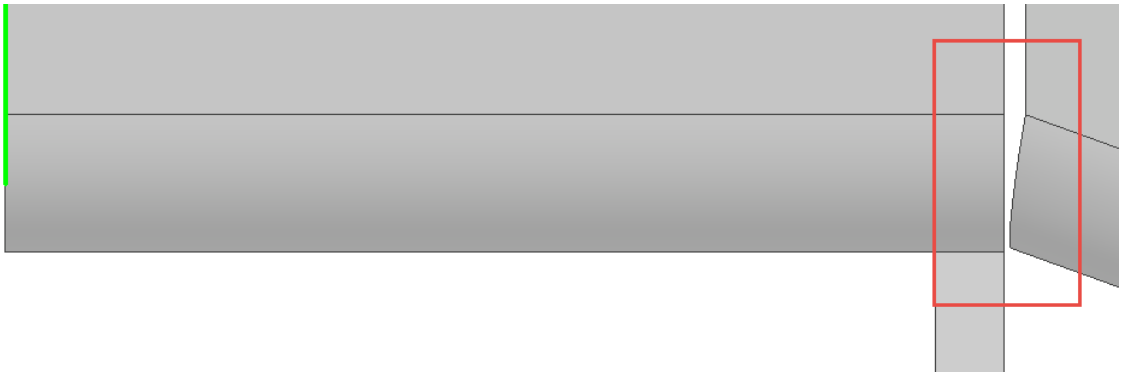
I have noticed that for some of the more complex surfaces that I've created, the model that I converted to Revit was missing some of the faces. After doing an extensive amount of research, I've discovered that "bad" geometric transitions (overlapping geometry, small gaps around a tangent surface, etc.) will cause a face to disappear when converting the model to Revit. If great care is taken in the component building stage, then these types of situations can be avoided. Please see below for some examples of "bad" geometry and how to remedy them.



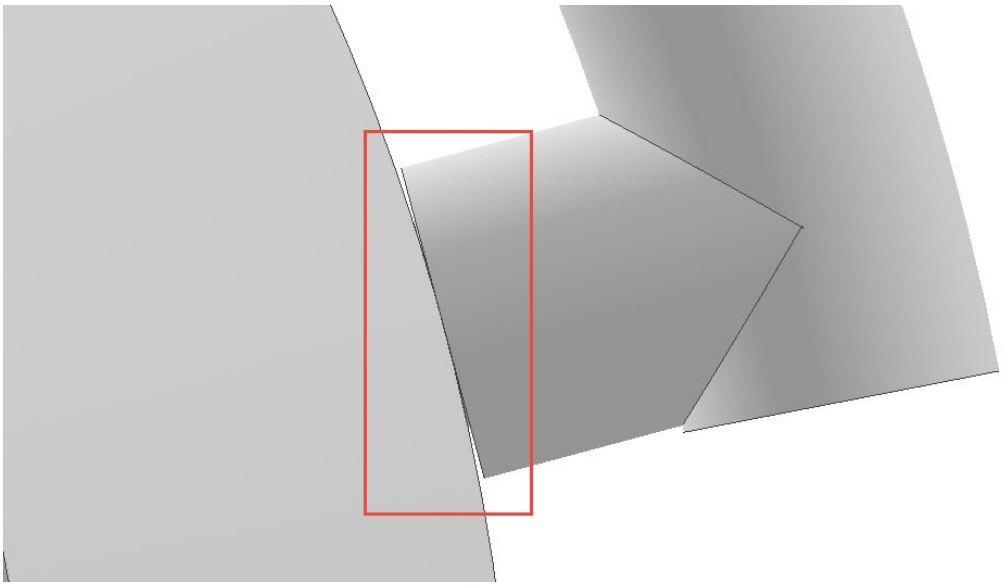
OVERLAPPING GEOMETRY CAN CAUSE PROBLEMS, AS SHOWN ON THE FAN MOUNT



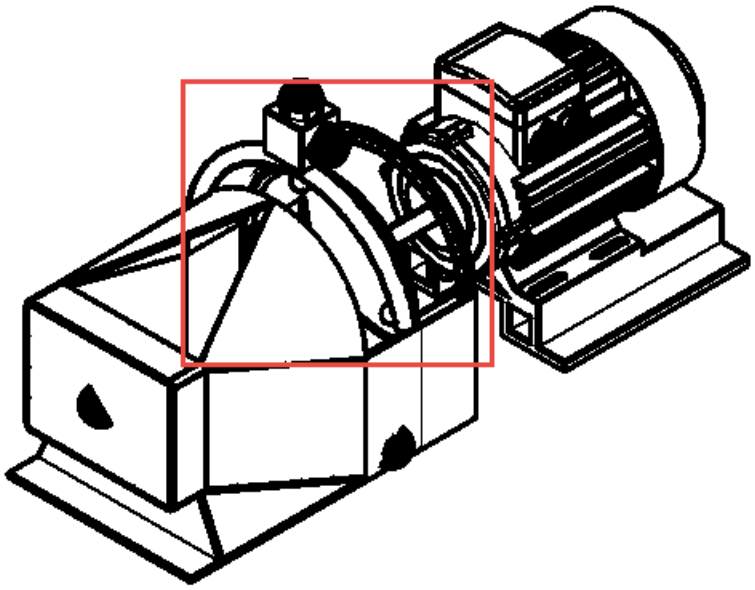
OVERLAPPING GEOMETRY CAUSES TRANSITIONAL FACE "TRIANGLE" TO DISAPPEAR IN REVIT



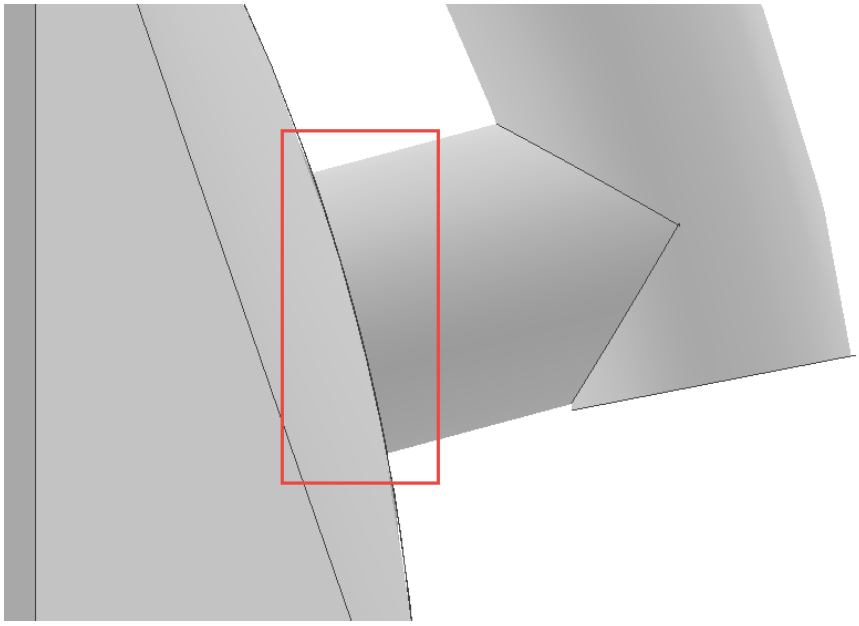
REMOVING THE OVERLAP WITH A SMALL GAP HELPS ELIMINATE THE ISSUE



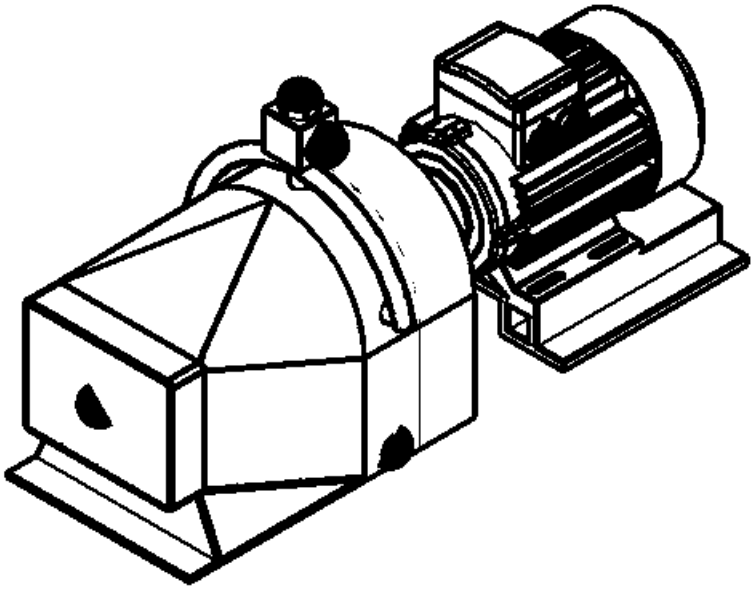
SMALL GAPS WITH A TANGENCY ALSO CAUSED SOME PROBLEMS



SMALL GAPS WITH TANGENCY CAUSED ROUND HOOD TO GO MISSING



REPLACE THE SMALL GAPS AROUND THE TANGENCY WITH A SADDLE-CUT FEATURE



REPAIRING THE OVERLAP AND SMALL GAPS WITH TANGENCY SITUATIONS FIXES ALL VISUAL ISSUES IN REVIT

Acknowledgements

I cannot accomplish all of this on my own and I am blessed and thankful to recognize the following people who helped to make this course possible.

- God Himself for this wonderful opportunity to serve others and literally every breath that take!
- Scott Dibben, my boss, for allowing me the time and space to explore these zany ideas
- Jelte de Jong for his assistance with constructing portions of the iLogic code. Please see some of his other blog posts (https://clintbrown.co.uk/tag/jelte/)
- Hakan Yildirm who gave permission for me to use the controller model. Please see some of his other models on GrabCAD (https://grabcad.com/hakan.yildirim-1)

Appendix A: Weblinks to video demonstrations

Video Link for Overview of Solution:
https://www.youtube.com/watch?v=tLeMrownUGQ

Video Link for Custom BIM Appearance Creation and Application (Solid Faces and Surfaces):
https://knowledge.autodesk.com/community/screencast/bd6330e0-8042-4af6-90ad-42a45f2adaab

Video Link to Component Approach Using iMates:
https://knowledge.autodesk.com/community/screencast/eff45e0b-c624-4f17-acfb-94388c875cf2

Video Link for Shrinkwrap Substitute Creation:
https://www.youtube.com/watch?v=npQqKn1XJHY

Video Link for Shrinkwrap Surface Recover After Editing:
https://youtu.be/gnPrzyjzsJ0

Video Link Demonstrating iLogic Rule Automation:
https://knowledge.autodesk.com/community/screencast/68661ce5-19a6-4ae8-b223-96b4300e1257

Video Link for Advanced BIM Connector Creation:
https://knowledge.autodesk.com/community/screencast/25a8feb9-29bd-49b0-9549-2ce5131c9ede

Video Link Demonstrating OmniClass Automation:
https://knowledge.autodesk.com/community/screencast/43432d13-23dc-4171-a1cd-5f21692c38da

Video Link for Fixing Small Gap Tangency Issues:
https://youtu.be/viZfc6FOMy4

Video Link for Fixing Overlapping Geometry Issues:
https://youtu.be/jif-feF-Yls

Appendix B: iLogic code samples

iLogic Rule for Applying BIM Connectors to Solid Faces

```
Public Class ThisRule

        Private busy As Boolean = False

        Sub Main()
                ' Get the collection of control definitions.
                Dim oControlDefs As ControlDefinitions =
ThisApplication.CommandManager.ControlDefinitions
                Dim oControlDef As ControlDefinition =
oControlDefs.Item("AEC_Exchange:Command:Connector:Edit")

                ' check if this document is a part document
                If ThisDoc.Document.DocumentType <>
DocumentTypeEnum.kPartDocumentObject Then
                        MessageBox.Show("The current document must be a part
file", "Incorrect Document Type")
                        Return
                End If

                'Enter the BIM Content environment

        ThisApplication.CommandManager.ControlDefinitions.Item("AEC_Exchange:E
nvironment").Execute()

                'Access the Browser and Activate the "Bim Content" Browser
Pane
                Dim oPartDoc As PartDocument = ThisDoc.Document
                Dim bps = oPartDoc.BrowserPanes
                Dim bimContentPane As BrowserPane =
oPartDoc.BrowserPanes.Item("AEC_Exchange:Browser")

                'Define the Appearance variables
                Dim oElecAppearance As Asset =
oPartDoc.Assets.Item("Connection_Electrical")
                Dim oDuctAppearance As Asset =
oPartDoc.Assets.Item("Connection_Duct")
                Dim oPipeAppearance As Asset =
oPartDoc.Assets.Item("Connection_Piping")

                'Set the Connection Counters
                Elec_Count = 0
                Duct_Count = 0
                Pipe_Count = 0

                'Define the BIM Component Information
                Dim oPartDocBIM As BIMComponent =
oPartDoc.ComponentDefinition.BIMComponent

'               'Check through each Face on the body and see if uses the
special appearance.
```

```
                For Each oSrfBody As SurfaceBody In
oPartDoc.ComponentDefinition.SurfaceBodies
                    For Each oFace As Face In oSrfBody.Faces
                        If oFace.Appearance.DisplayName =
oElecAppearance.DisplayName Then

                            Dim Elec_Coll As ObjectCollection
                            Elec_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                            Elec_Coll.Add(oFace)

                            'Create Electrical Connector Definition
                            Dim oElecConn As
BIMElectricalConnectorDefinition
                            oElecConn =
oPartDocBIM.Connectors.CreateElectricalConnectorDefinition(Elec_Coll)
                            'oElecConn.SystemType =
BIMElectricalSystemTypeEnum.kPowerBalancedElectricalSystemType

                            'Add the Electrical BIM Connection
                            Elec_Count = Elec_Count + 1
                            Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oElecConn)
                            con.Name = "Elec_Conn_" & Elec_Count
'oPartDocBIM.Connectors.Count

                            Dim systemNode As BrowserNode =
bimContentPane.TopNode.BrowserNodes.Item("MEP System Connections")
                            Dim electicalNode As BrowserNode =
systemNode.BrowserNodes.Item("Electrical")
                            electicalNode.Expanded = True
                            Dim oConnNode As BrowserNode =
electicalNode.BrowserNodes.Item(con.Name)
                            oConnNode.DoSelect()

                            oControlDef.Execute()
                            Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                            AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                            busy = True
                            While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                            End While
                            RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()
                            'electicalNode.Expanded = False
```

```
                                Else If oFace.Appearance.DisplayName =
oDuctAppearance.DisplayName Then

                                    Dim Duct_Coll As ObjectCollection
                                    Duct_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                                    Duct_Coll.Add(oFace)

                                    'Create Duct Connector Definition
                                    Dim oDuctConn As
BIMDuctConnectorDefinition
                                    oDuctConn =
oPartDocBIM.Connectors.CreateDuctConnectorDefinition(Duct_Coll,kRectangularSh
apeConnector)

                                    'Add the Duct BIM Connection
                                    Duct_Count = Duct_Count + 1
                                    Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oDuctConn)
                                    con.Name = "Duct_Conn_" & Duct_Count
'oPartDocBIM.Connectors.Count

                                    Dim systemNode As BrowserNode =
bimContentPane.TopNode.BrowserNodes.Item("MEP System Connections")
                                    Dim DuctNode As BrowserNode =
systemNode.BrowserNodes.Item("Duct")
                                    DuctNode.Expanded = True
                                    Dim oConnNode As BrowserNode =
DuctNode.BrowserNodes.Item(con.Name)
                                    oConnNode.DoSelect()

                                    oControlDef.Execute()
                                    Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                                    AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                    busy = True
                                    While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                    End While
                                    RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()

                                Else If oFace.Appearance.DisplayName =
oPipeAppearance.DisplayName Then

                                    Dim Pipe_Coll As ObjectCollection
                                    Pipe_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
```

```
                                        Pipe_Coll.Add(oFace)

                                        'Create Duct Connector Definition
                                        Dim oPipeConn As
BIMPipeConnectorDefinition
                                        oPipeConn =
oPartDocBIM.Connectors.CreatePipeConnectorDefinition(Pipe_Coll,kCircularShape
Connector)

                                        'Add the Duct BIM Connection
                                        Pipe_Count = Pipe_Count + 1
                                        Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oPipeConn)
                                        con.Name = "Pipe_Conn_" & Pipe_Count
'oPartDocBIM.Connectors.Count

                                        'Since using a surface versus a solid
face, must flip the direction
                                        con.Definition.ReverseDirection

                                        Dim systemNode As BrowserNode =
bimContentPane.TopNode.BrowserNodes.Item("MEP System Connections")
                                        Dim pipeNode As BrowserNode =
systemNode.BrowserNodes.Item("Pipe")
                                        pipeNode.Expanded = True
                                        Dim oConnNode As BrowserNode =
pipeNode.BrowserNodes.Item(con.Name)
                                        oConnNode.DoSelect()

                                        oControlDef.Execute()
                                        Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                                        AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                        busy = True
                                        While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                        End While
                                        RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()
                                End If
                                Next
                        Next
'               Next
        End Sub
```

```
        Private Sub docEvents_OnChange(ReasonsForChange As CommandTypesEnum,
BeforeOrAfter As EventTimingEnum, Context As NameValueMap, ByRef HandlingCode
As HandlingCodeEnum)

                If (BeforeOrAfter = EventTimingEnum.kAfter And Context(1) =
"API Change") Then
                        busy = False
                End If

        End Sub

End Class
```

iLogic Rule for Applying BIM Connectors to Boundary Patch Surfaces

```
Public Class ThisRule

	Private busy As Boolean = False

	Sub Main()
		' Get the collection of control definitions.
		Dim oControlDefs As ControlDefinitions =
ThisApplication.CommandManager.ControlDefinitions
		Dim oControlDef As ControlDefinition =
oControlDefs.Item("AEC_Exchange:Command:Connector:Edit")

		' check if this document is a part document
		If ThisDoc.Document.DocumentType <>
DocumentTypeEnum.kPartDocumentObject Then
			MessageBox.Show("The current document must be a part
file", "Incorrect Document Type")
			Return
		End If

		'Enter the BIM Content environment

	ThisApplication.CommandManager.ControlDefinitions.Item("AEC_Exchange:E
nvironment").Execute()

		'Access the Browser and Activate the "Bim Content" Browser
Pane
		Dim oPartDoc As PartDocument = ThisDoc.Document
		Dim bps = oPartDoc.BrowserPanes
		Dim bimContentPane As BrowserPane =
oPartDoc.BrowserPanes.Item("AEC_Exchange:Browser")

		'Define the Appearance variables
		Dim oElecAppearance As Asset =
oPartDoc.Assets.Item("Connection_Electrical")
		Dim oDuctAppearance As Asset =
oPartDoc.Assets.Item("Connection_Duct")
		Dim oPipeAppearance As Asset =
oPartDoc.Assets.Item("Connection_Piping")

		'Set the Connection Counters based on the results from the
Solid_Faces and the current counts from the BIM Browser Nodes
		Dim systemNode As BrowserNode =
bimContentPane.TopNode.BrowserNodes.Item("MEP System Connections")
		Dim electicalNode As BrowserNode =
systemNode.BrowserNodes.Item("Electrical")
		electicalNode.Expanded = True
		Dim oElecConnCount As Integer =
electicalNode.BrowserNodes.Count
		Dim DuctNode As BrowserNode =
systemNode.BrowserNodes.Item("Duct")
		DuctNode.Expanded = True
		Dim oDuctConnCount As Integer = DuctNode.BrowserNodes.Count
```

```
                Dim pipeNode As BrowserNode =
systemNode.BrowserNodes.Item("Pipe")
                pipeNode.Expanded = True
                Dim oPipeConnCount As Integer = pipeNode.BrowserNodes.Count

'               'Set the Connection Counters
                Elec_Count = oElecConnCount
                Duct_Count = oDuctConnCount
                Pipe_Count = oPipeConnCount

                'Define the BIM Component Information
                Dim oPartDocBIM As BIMComponent =
oPartDoc.ComponentDefinition.BIMComponent


                'Check through each surface in the part and see if it uses the
special appearances
                For Each oWrkSurf As WorkSurface In
oPartDoc.ComponentDefinition.WorkSurfaces
                        For Each oSrfBody As SurfaceBody In
oWrkSurf.SurfaceBodies
                                For Each oFace As Face In oSrfBody.Faces
                                If oFace.Appearance.DisplayName =
oElecAppearance.DisplayName Then

                                        Dim Elec_Coll As ObjectCollection
                                        Elec_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                                        Elec_Coll.Add(oFace)

                                        'Create Electrical Connector Definition
                                        Dim oElecConn As
BIMElectricalConnectorDefinition
                                        oElecConn =
oPartDocBIM.Connectors.CreateElectricalConnectorDefinition(Elec_Coll)

                                        'Add the Electrical BIM Connection
                                        Elec_Count = Elec_Count + 1
                                        Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oElecConn)
                                        con.Name = "Elec_Conn_" & Elec_Count
'oPartDocBIM.Connectors.Count

                                        'Since using a surface versus a solid
face, must flip the direction
                                        con.Definition.ReverseDirection

                                        'Activate the current electrical node
so characteristics can be modified, if desired.
                                        Dim oConnNode As BrowserNode =
electicalNode.BrowserNodes.Item(con.Name)
                                        oConnNode.DoSelect()

                                        oControlDef.Execute()
```

```
                                    Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                                    AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                    busy = True
                                    While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                    End While
                                    RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()
                                    'electicalNode.Expanded = False

                                Else If oFace.Appearance.DisplayName =
oDuctAppearance.DisplayName Then

                                    Dim Duct_Coll As ObjectCollection
                                    Duct_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                                    Duct_Coll.Add(oFace)

                                    'Create Duct Connector Definition
                                    Dim oDuctConn As
BIMDuctConnectorDefinition
                                    oDuctConn =
oPartDocBIM.Connectors.CreateDuctConnectorDefinition(Duct_Coll,kRectangularSh
apeConnector)

                                    'Add the Duct BIM Connection
                                    Duct_Count = Duct_Count + 1
                                    Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oDuctConn)
                                    con.Name = "Duct_Conn_" & Duct_Count
'oPartDocBIM.Connectors.Count

                                    'Since using a surface versus a solid
face, must flip the direction
                                    con.Definition.ReverseDirection

                                    'Activate the current duct node so
characteristics can be modified, if desired.
                                    DuctNode.Expanded = True
                                    Dim oConnNode As BrowserNode =
DuctNode.BrowserNodes.Item(con.Name)
                                    oConnNode.DoSelect()

                                    oControlDef.Execute()
                                    Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
```

```
                                    AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                    busy = True
                                    While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                    End While
                                    RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()

                                Else If oFace.Appearance.DisplayName =
oPipeAppearance.DisplayName Then

                                    Dim Pipe_Coll As ObjectCollection
                                    Pipe_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                                    Pipe_Coll.Add(oFace)

                                    'Create Duct Connector Definition
                                    Dim oPipeConn As
BIMPipeConnectorDefinition
                                    oPipeConn =
oPartDocBIM.Connectors.CreatePipeConnectorDefinition(Pipe_Coll,kCircularShape
Connector)

                                    'Add the Duct BIM Connection
                                    Pipe_Count = Pipe_Count + 1
                                    Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oPipeConn)
                                    con.Name = "Pipe_Conn_" & Pipe_Count
'oPartDocBIM.Connectors.Count

                                    'Since using a surface versus a solid
face, must flip the direction
                                    con.Definition.ReverseDirection

                                    'Activate the current pipe node so
characteristics can be modified, if desired.
                                    pipeNode.Expanded = True
                                    Dim oConnNode As BrowserNode =
pipeNode.BrowserNodes.Item(con.Name)
                                    oConnNode.DoSelect()

                                    oControlDef.Execute()
                                    Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                                    AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                    busy = True
                                    While (busy = True)
```

```
        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                        End While
                                        RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()
                                End If
                                Next
                        Next
                Next
        End Sub


        Private Sub docEvents_OnChange(ReasonsForChange As CommandTypesEnum,
BeforeOrAfter As EventTimingEnum, Context As NameValueMap, ByRef HandlingCode
As HandlingCodeEnum)

                If (BeforeOrAfter = EventTimingEnum.kAfter And Context(1) =
"API Change") Then
                        busy = False
                End If

        End Sub

End Class
```

iLogic Rule to Create a Basic Revit Family

```
'Define the Part Document to prepare for BIM Revit Export
Dim oPartDoc As PartDocument = ThisDoc.Document

'Define the BIM Component
Dim oPartDocBIM As BIMComponent = oPartDoc.ComponentDefinition.BIMComponent

'Extract the current filename And parse Out the .ipt file extension
Dim Comp_Name As String = oPartDoc.DisplayName
Dim Comp_Name_Count As Integer = Comp_Name.Length
Dim Final_Comp_Name As String = Left(Comp_Name, Comp_Name_Count - 4)

'Set up the file path from the original host file
Dim File_Path_Length As Integer = oPartDoc.FullFileName.Length
Dim File_Path As String = Left(oPartDoc.FullFileName, File_Path_Length -
Comp_Name_Count)

'Create the Revit Family using the extracted file path and formed Revit file
name
oPartDocBIM.ExportBuildingComponent(File_Path & Final_Comp_Name & ".rfa")
```

iLogic Internal Central Control Rule (Simple)

```
'Check all the Solid Faces for the BIM Custom Appearances and create
connectors
iLogicVb.RunRule("Jelte_Rule_Solid_Faces")

'Check all the Surface Boundary Patches for the BIM Custom Appearances and
create connectors
iLogicVb.RunRule("Jelte_Rule_Surfaces")

'Export the model as a Revit family with the same root filename in the same
file location
iLogicVb.RunRule("Create_Revit_Family")

iLogicVb.UpdateWhenDone = True
```

iLogic External Central Control Rule (Simple)

```
'Check all the Solid Faces for the BIM Custom Appearances and create
connectors
iLogicVb.RunExternalRule("Jelte_BIM_Solid_Faces")

'Check all the Surface Boundary Patches for the BIM Custom Appearances and
create connectors
iLogicVb.RunExternalRule("Jelte_BIM_Surfaces")

'Export the model as a Revit family with the same root filename in the same
file location
iLogicVb.RunExternalRule("Create_Revit_Family_Basic")

iLogicVb.UpdateWhenDone = True
```

iLogic Rule for Information Rich BIM Connectors

```
Public Class ThisRule

        Private busy As Boolean = False

        Sub Main()

                ' Get the collection of control definitions.
                Dim oControlDefs As ControlDefinitions =
ThisApplication.CommandManager.ControlDefinitions
                Dim oControlDef As ControlDefinition =
oControlDefs.Item("AEC_Exchange:Command:Connector:Edit")

                ' check if this document is a part document
                If ThisDoc.Document.DocumentType <>
DocumentTypeEnum.kPartDocumentObject Then
                        MessageBox.Show("The current document must be a part
file", "Incorrect Document Type")
                        Return
                End If

                'Enter the BIM Content environment

        ThisApplication.CommandManager.ControlDefinitions.Item("AEC_Exchange:E
nvironment").Execute()

                'Access the Browser and Activate the "Bim Content" Browser
Pane
                Dim oPartDoc As PartDocument = ThisDoc.Document
                Dim bps = oPartDoc.BrowserPanes
                Dim bimContentPane As BrowserPane =
oPartDoc.BrowserPanes.Item("AEC_Exchange:Browser")

                'Set the Connection Counters
                Elec_Count = 0
                Duct_Count = 0
                Pipe_Count = 0

                'Define the BIM Component Information
                Dim oPartDocBIM As BIMComponent =
oPartDoc.ComponentDefinition.BIMComponent


                'Check through each Face on the body and see if uses the
special appearance.
                For Each oSrfBody As SurfaceBody In
oPartDoc.ComponentDefinition.SurfaceBodies
                        For Each oFace As Face In oSrfBody.Faces
                                'Define a variable to determine which type of
Connector Surface is present
                                Dim BIM_Conn_Type As String =
Left(oFace.Appearance.DisplayName, 3)
                                'Determine if this is an Electrical Connector
```

```
                        If BIM_Conn_Type = "EL_" Then

                                Dim Elec_Coll As ObjectCollection
                                Elec_Coll =
ThisApplication.TransientObjects.CreateObjectCollection
                                Elec_Coll.Add(oFace)

                                'Create Electrical Connector Definition
                                Dim oElecConn As
BIMElectricalConnectorDefinition
                                oElecConn =
oPartDocBIM.Connectors.CreateElectricalConnectorDefinition(Elec_Coll)

                                'Create a variable to determine what
kind System Type is being utilized
                                Dim Elec_Sys_Type As String =
Mid(oFace.Appearance.DisplayName, 4, 2)
                                'Determine which Electrical System Type
is being used and adjust accordingly
                                Select Case Elec_Sys_Type

                                Case "PB"
                                        oElecConn.SystemType =
BIMElectricalSystemTypeEnum.kPowerBalancedElectricalSystemType

                                        oElecConn.Voltage =
Mid(oFace.Appearance.DisplayName, 7, 3)

                                        oElecConn.NumberOfPoles =
Mid(oFace.Appearance.DisplayName, 11, 1)

                                        oElecConn.ApparentLoad =
Mid(oFace.Appearance.DisplayName, 13, 4)

                                        oElecConn.LoadClassification =
Mid(oFace.Appearance.DisplayName, 18, 3)

                                        If
Mid(oFace.Appearance.DisplayName, 22, 1) = "T" Then
                                                oElecConn.HasMotor =
"True"
                                        Else
                                                oElecConn.HasMotor =
"False"
                                        End If

                                        oElecConn.PowerFactor = "." &
Mid(oFace.Appearance.DisplayName, 24, 2)

                                        If
Right(oFace.Appearance.DisplayName, 3) = "LED" Then

        oElecConn.PowerFactorState = kLeadingPowerFactorStateType
                                        Else
```

```
        oElecConn.PowerFactorState = kLaggingPowerFactorStateType
                                            End If

                                        Case "CN"
                                            oElecConn.SystemType =
BIMElectricalSystemTypeEnum.kControlsElectricalSystemType

                                            'Determine the length of the
overall text string and then we can remove
                                            'the first six characters
"EL_CN_" and use the rest in the Description
                                            Dim oStringLength As Integer =
Len(oFace.Appearance.DisplayName)

                                            'Use the overall string length
minus the first 6 characters for the Description
                                            oElecConn.Description =
Mid(oFace.Appearance.DisplayName, 7, oStringLength - 6)

                                        End Select

                                        'Add the Electrical BIM Connection
                                        Elec_Count = Elec_Count + 1
                                        Dim con As BIMConnector =
oPartDocBIM.Connectors.Add(oElecConn)
                                        con.Name = "Elec_Conn_" & Elec_Count
'oPartDocBIM.Connectors.Count

                                        Dim systemNode As BrowserNode =
bimContentPane.TopNode.BrowserNodes.Item("MEP System Connections")
                                        Dim electicalNode As BrowserNode =
systemNode.BrowserNodes.Item("Electrical")
                                        electicalNode.Expanded = True
                                        Dim oConnNode As BrowserNode =
electicalNode.BrowserNodes.Item(con.Name)
                                        oConnNode.DoSelect()

                                        oControlDef.Execute()
                                        Dim docEvents As DocumentEvents =
ThisApplication.ActiveDocument.DocumentEvents
                                        AddHandler docEvents.OnChange,
AddressOf docEvents_OnChange
                                        busy = True
                                        While (busy = True)

        System.Threading.Thread.Sleep(50)

        ThisApplication.UserInterfaceManager.DoEvents()
                                        End While
                                        RemoveHandler docEvents.OnChange,
AddressOf docEvents_OnChange

        ThisApplication.CommandManager.StopActiveCommand()
```

```
                                        'electicalNode.Expanded = False

                                End If
                        Next
                Next
        End Sub


        Private Sub docEvents_OnChange(ReasonsForChange As CommandTypesEnum,
BeforeOrAfter As EventTimingEnum, Context As NameValueMap, ByRef HandlingCode
As HandlingCodeEnum)

                If (BeforeOrAfter = EventTimingEnum.kAfter And Context(1) =
"API Change") Then
                        busy = False
                End If

        End Sub

End Class
```

iLogic Rule Advanced Create Revit Family (OmniClass Selector)

```
'Define the Part Document to prepare for BIM Revit Export
Dim oPartDoc As PartDocument = ThisDoc.Document

'Define the BIM Component
Dim oPartDocBIM As BIMComponent = oPartDoc.ComponentDefinition.BIMComponent

'Extract the current filename And parse Out the .ipt file extension
Dim Comp_Name As String = oPartDoc.DisplayName
Dim Comp_Name_Count As Integer = Comp_Name.Length
Dim Final_Comp_Name As String = Left(Comp_Name, Comp_Name_Count - 4)

'Set up the file path from the original host file
Dim File_Path_Length As Integer = oPartDoc.FullFileName.Length
Dim File_Path As String = Left(oPartDoc.FullFileName, File_Path_Length -
Comp_Name_Count)

'Pare down and automate the setting of the OmniClass
'Allow the user to access the HVAC OmniClass list and select the desired
classification
HVAC_OmniClass = InputListBox("Choose the desired OmniClass for this
equipment.", MultiValue.List("HVAC_OmniClass"), HVAC_OmniClass, Title :=
"OmniClass Selector", ListName := "OmniClass List")
'Push the OmniClass value to the BIM Component Properties
oPartDocBIM.ComponentDescription.ComponentType = Left(HVAC_OmniClass,11)

'Create the Revit Family using the extracted file path and formed Revit file
name
oPartDocBIM.ExportBuildingComponent(File_Path & Final_Comp_Name & ".rfa")
```

Appendix C: Web Articles

Minneapolis – St. Paul Business Journal Article on Rise Modular
https://www.bizjournals.com/twincities/news/2020/09/20/rise-modular-begins-first-project-in-minneapolis.html#g/471805/9

Link to Factory Design Utilities for the Product Designer AU2018 Course
https://www.autodesk.com/autodesk-university/class/Factory-Design-Utilities-Product-Designer-2018