MFG501041

What a State We're In: Inventor Model State Tips & Tricks

Pete Strycharske
TeamD3

Learning Objectives

- See why naming Model States is important and be cautious of relying on the Primary / Master State.
- Learn how to create a setup row to make working in Excel MUCH easier.
- Learn about using Excel to rearrange Model State rows and columns.
- Learn about building simple iLogic rules to animate and "test out" part forming.

Description

Model States is now more than a year old since coming out in Inventor 2022 software, and the speaker has worked with many clients, exploring functionality, answering questions, and teaching classes on Model States. In this class, you'll learn tips that can that help Model States work more efficiently and effectively, streamlining the creation and maintenance of Model States, and you'll get an introduction on how to use iLogic with Model States. We'll cover things as straightforward as naming Model States—all the way to animating Model States to show the process of forming a part. No matter your familiarity with this new functionality, there should be a little something for everyone.

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 4

What's in a Name? 5
- Caution with Using the Master / Primary Model State 5
- Consistent Naming is Key 8

Let's Get Organized 9
- Starting a "Setup" State to Build All the Features 9
- Excel Benefits 11

Everyone Loves a Good Remodel 13
- Rearranging Property Columns in Excel 13
- Rearranging Rows 14

Bend it Like Beckham 16
- Accessing Models States via the API 16
- Setting an iLogic Timer 17
- Animating Sheet Metal Forming via Model States 17

Extra Fun 18
- Linking Model States in Lieu of Derived Designs 18
- Working with Converted Levels of Detail 22
- Method for Tracking Model States in Vault 25

Acknowledgements 29

Appendix A: Weblinks to Video Demonstrations 30

Appendix B: Sample iLogic Code 31
- iLogic Rule for Animating Model States 31
- iLogic Rule for Single Model State iProperty 32
- iLogic Rule for Creating a Model State iProperty for Each Component 34

The Goal

Thank you so much for taking a look into this class on Model States tips. There are other super useful classes that cover the basics and usage of Model States, so our goals will simply be to build upon this knowledge with some tips that I've gleaned from my customers and my own experimentation. I will consider this class successful, if you're able to come away with some better methods for organizing, as well as enhanced usage, of Model States.

For example, why should we be wary of using the "Primary" (2023) or the "Master" (2022) Model State? How can I more efficiently organize my Model State variations? Is there any way that I can preview the fabrication of my designs, without tying up the CAM group? Why won't some of my designs update the way that I anticipated, while using Model States? These are some of the questions that I'll attempt to answer in this class!

Quick Notes:

With the Inventor 2023 release, the out-of-the-box Model State is now named the "Primary" state, instead of the 2022 name, "Master". Since this name change is likely to stay, I'll just be using the name "Primary" for the balance of this class.

Also, please see Appendix A for video links covering many of the topics in this class.

Finally, I'm assuming that users in this class are at least somewhat versed with Model States, so I will not cover all essential Model State operations.

What in a Name?

Naturally names are super important, especially in the CAD realm, but there are a couple of items to watch out for when naming Model States. Without a good practice for the naming and usage of Model States, there's a good bet that confusion will reign supreme!

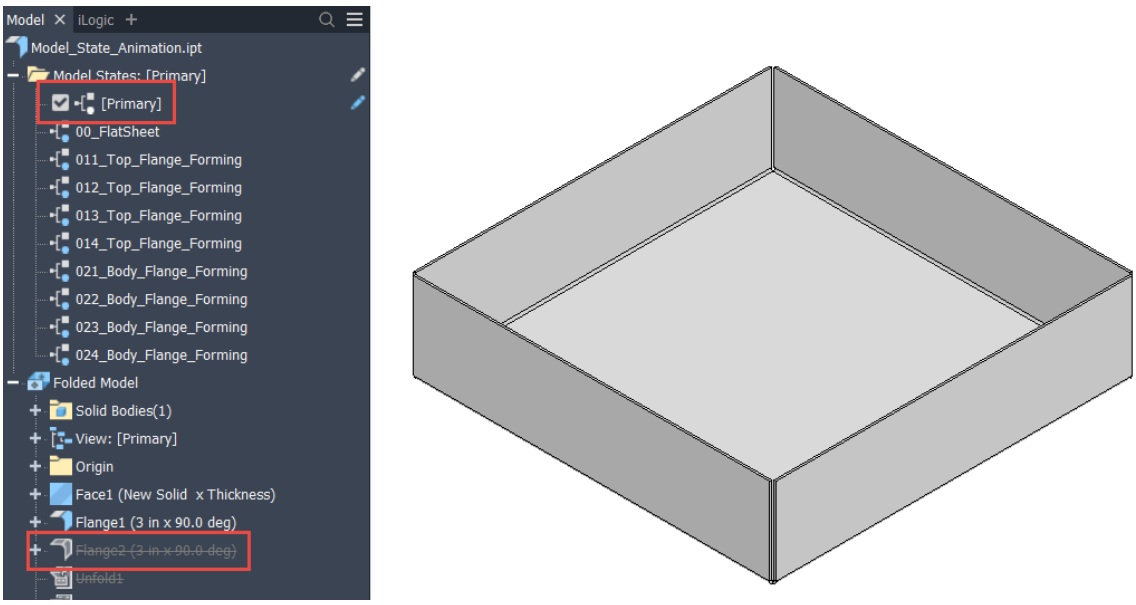
Caution with Using the Master / Primary Model State

Every Inventor design, starting with the 2022 release will come with a default Model State. This is sort of like the "Default" project file when initially running Inventor; you just need to have one. While one can create an entire design, solely using the "Primary" Model State, if one decides to add variations, using Model States, what constitutes the "Primary" state becomes a bit murky.

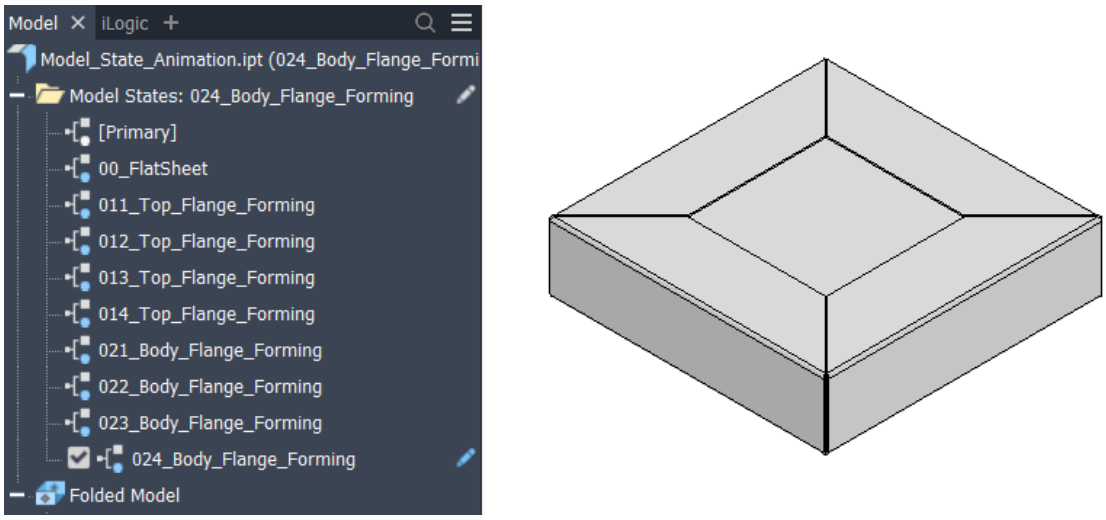
For example, does the "Primary" Model State mean that all features are active? Is the "Primary" state the final production design? Should we be editing the "Primary" state in the design. These are the types of questions that have come up in classes and when speaking with customers about how to use the "Primary" Model State. Here are a couple of concerns I have with the "Primary" Model State.

When Working with Parts:

In Part models, when one sees the term "Primary" it has the connotation of being the most important state, perhaps even indicating the final design. Since features can be suppressed in the "Primary" state, it's difficult to say if this is always the case. Also, what happens when we create a design that has multiple fabrication steps? What does the "Primary" state denote then.



FEATURES CAN BE SUPPRESSED WITH THE "PRIMARY" STATE ACTIVE



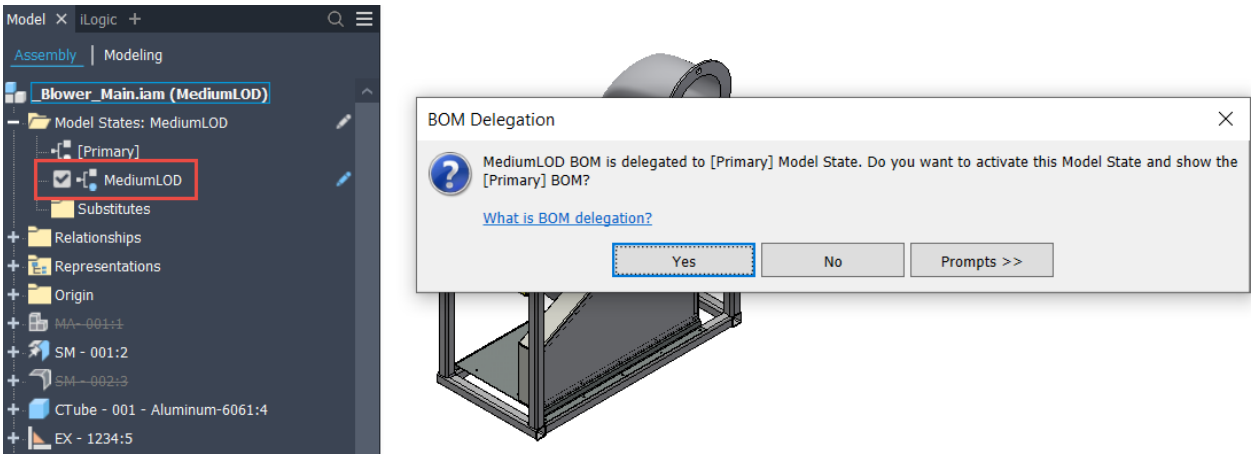
DESIGN WITH MULTIPLE FABRICATION STEPS

When Working with Assemblies:

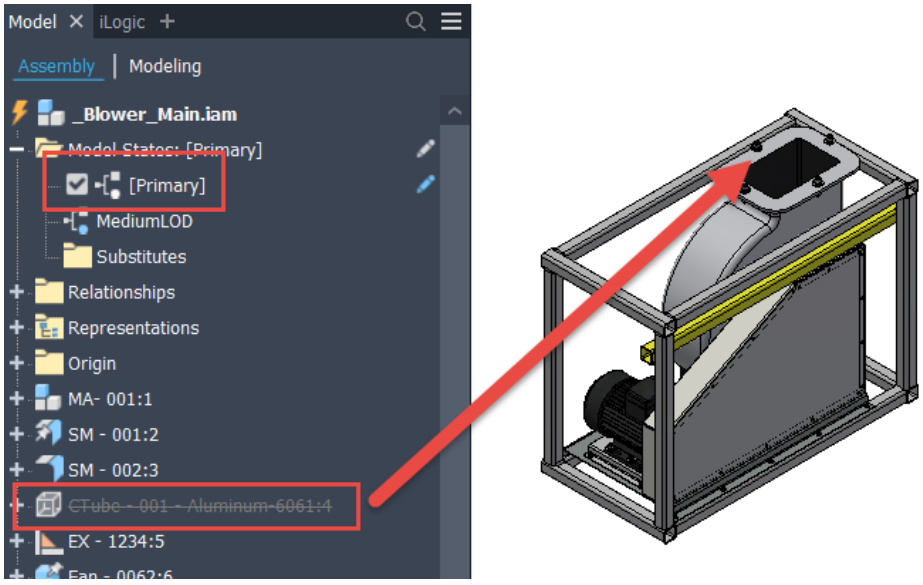
Working with assemblies can lead to some specific concerns with the “Primary” Model State, particularly when converted Levels of Detail (LOD) are involved.

As a brief reminder, before Model States, Levels of Detail controlled the suppression modes of assembly components. Levels of Detail maintained the quantities of suppressed components, which allowed for larger assemblies to be opened quickly, while still viewing the BOM list.

Since Model States replaced Levels of Detail, the custom LOD’s are converted into corresponding Model States. However, since LOD’s quantified the components when suppressed, these converted Model States must activate the “Primary” Model State when viewing the BOM. This is all well and good, until someone suppresses a component, while the “Primary” Model State is active (which is absolutely possible). Therefore, I generally recommend leaving the “Primary” Model State alone, when working with assemblies.



CUSTOM MODEL STATE CONVERTED FROM A LOD AND VIEWING THE BOM



PRIMARY MODEL STATE WITH A COMPONENT SUPPRESSED

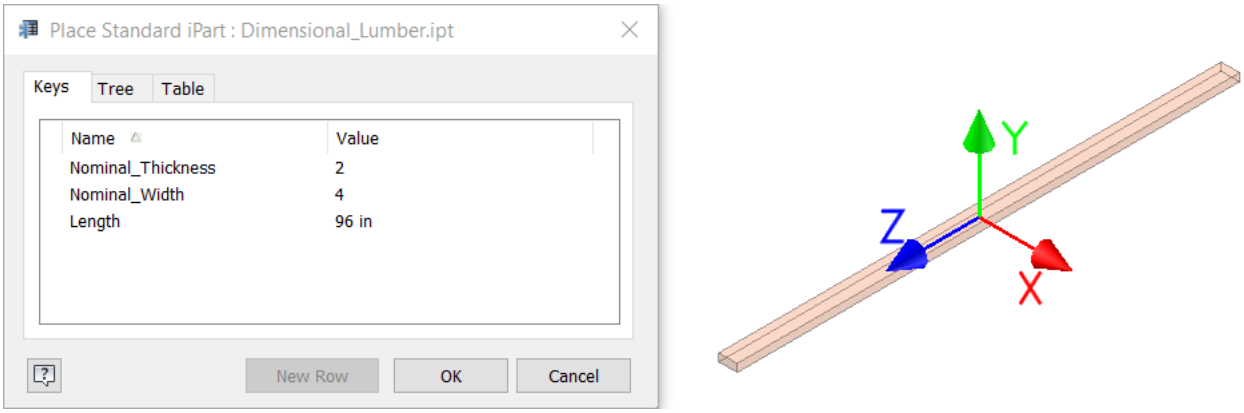
"Primary" State Conclusion:

For these types of reasons, I've started to think of the "Primary" Model State as the *Initial* Model State. It could be the final design, but it might not always be. To eliminate any possible confusion, for designs that have only ***ONE*** final variation, I've taken to creating a Model State "**Final_Design"** to designate the completed design. It is important to ensure that everyone is on the same page.

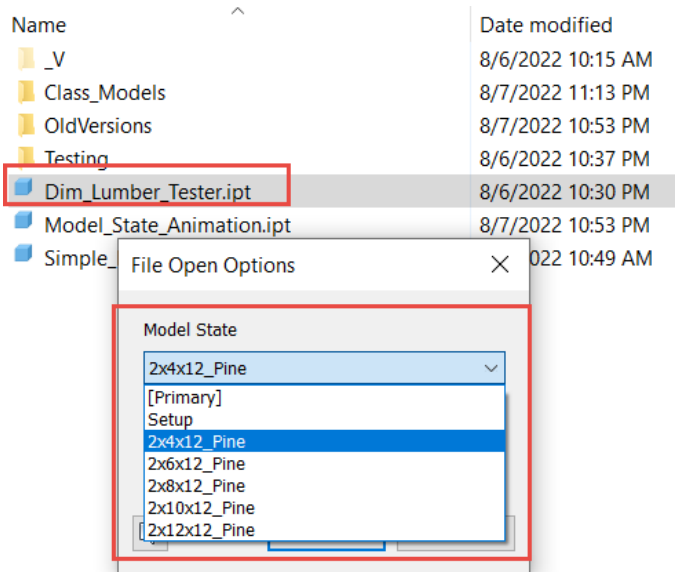
Consistent Naming is Key

If one is designing components with multiple final configurations or with multiple fabrication operations, then the naming of the Model States becomes paramount. On one hand, as with the "Primary" Model State, there can be confusion about the functionality of Model States without a clear name. So clearly understood names are important from an organizational standpoint.

However, there's another reason why the naming is important. When placing components in an assembly with multiple Model States, the name is the only characteristic available to distinguish between the versions, whereas with iParts Key values can be used to drill down to the desired configuration. For example, if there are ten different attributes being managed per model state, then somehow the Model State name must be robust enough to guide the end users to the desired design. This will take some careful forethought to incorporate all the necessary data.



IPART EXAMPLE WITH KEY PARAMETERS



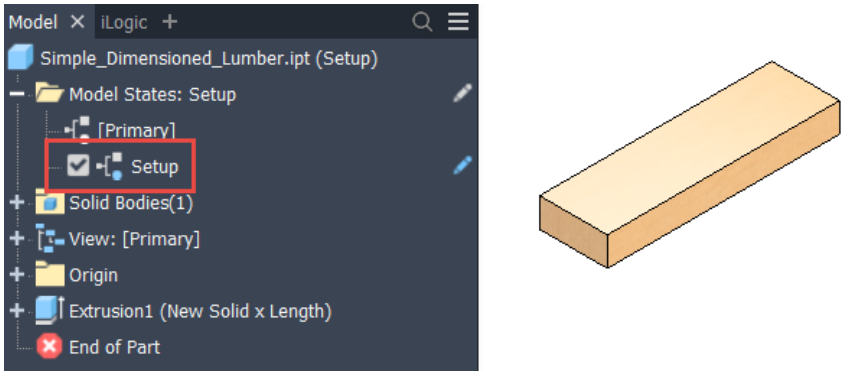
CORRESPONDING MODEL STATE NAME PLACEMENT

Let's Get Organized

If one is creating Model States of even moderate complexity, I find it useful to create a "Setup" Model State to help organize the design and allow for more efficient downstream modification.

Starting a "Setup" State to Build All the Features

Every design contains the "Primary" Model State, which allows all sorts of changes to parameters, iProperties, materials, etc. However, when changes are made to the "Primary" State, these do not add columns to the Excel table (which we'll see shortly). Therefore, I advise creating a "Setup" Model State which I use to make changes to all the desired metadata and suppress all controlled features. This also helps me think through the design variations to assist my "Swiss-cheese memory" and (hopefully) prevent me from overlooking something.

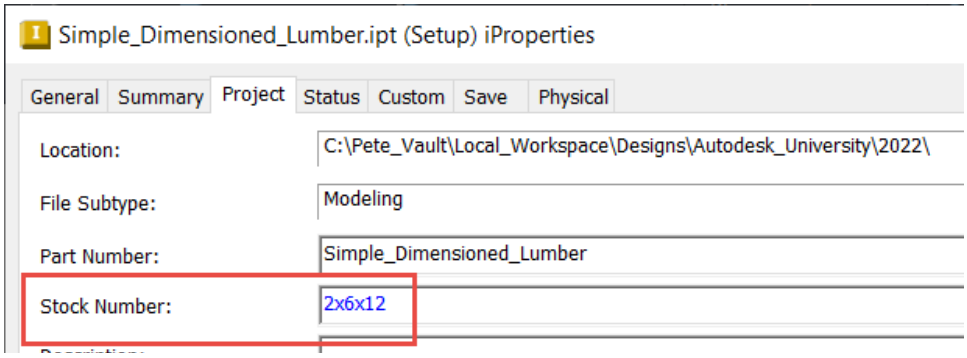


CREATE A SETUP ROW

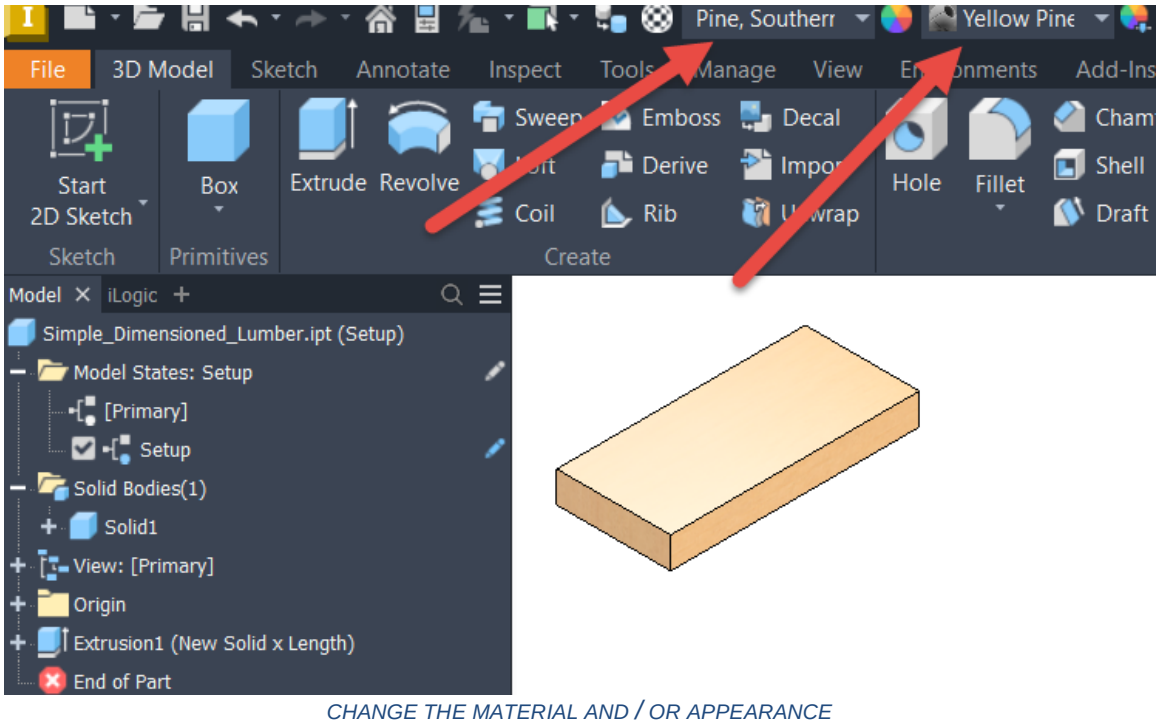
Parameters

Parameter Name	Consumed by	Unit/Type	Equation
Model Parameters			
User Parameters			
Thickness	d1	in	1.5 in
Width	d0	in	5.5 in
Length	d2	in	12 in

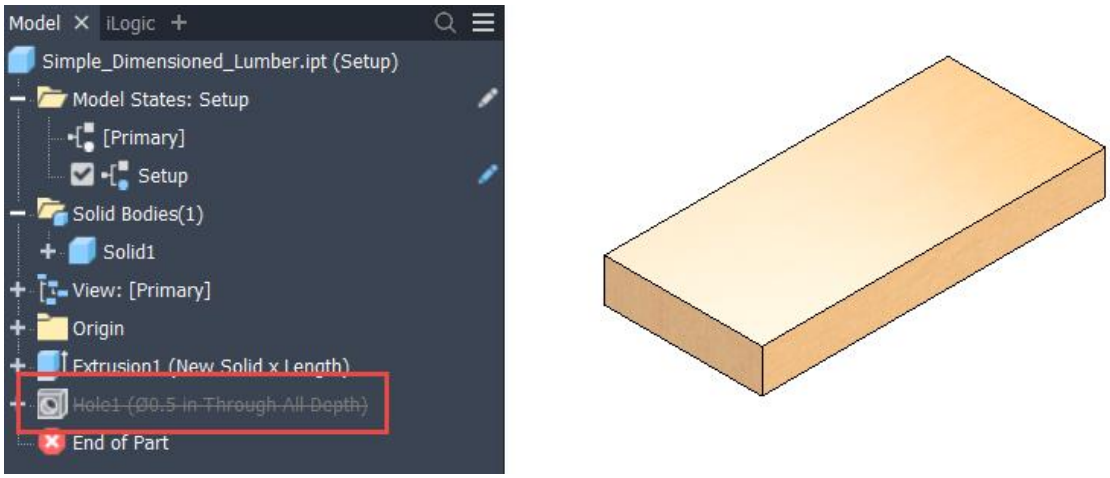
MODIFY A PARAMETER



MODIFY AN IPROPERTY



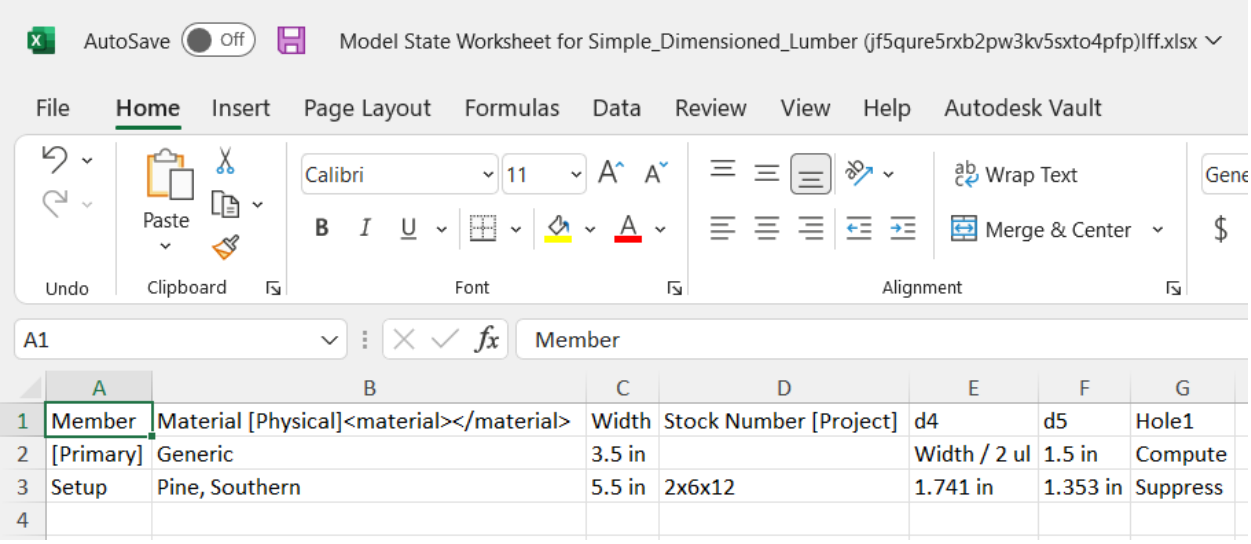
CHANGE THE MATERIAL AND / OR APPEARANCE



SUPPRESS A FEATURE

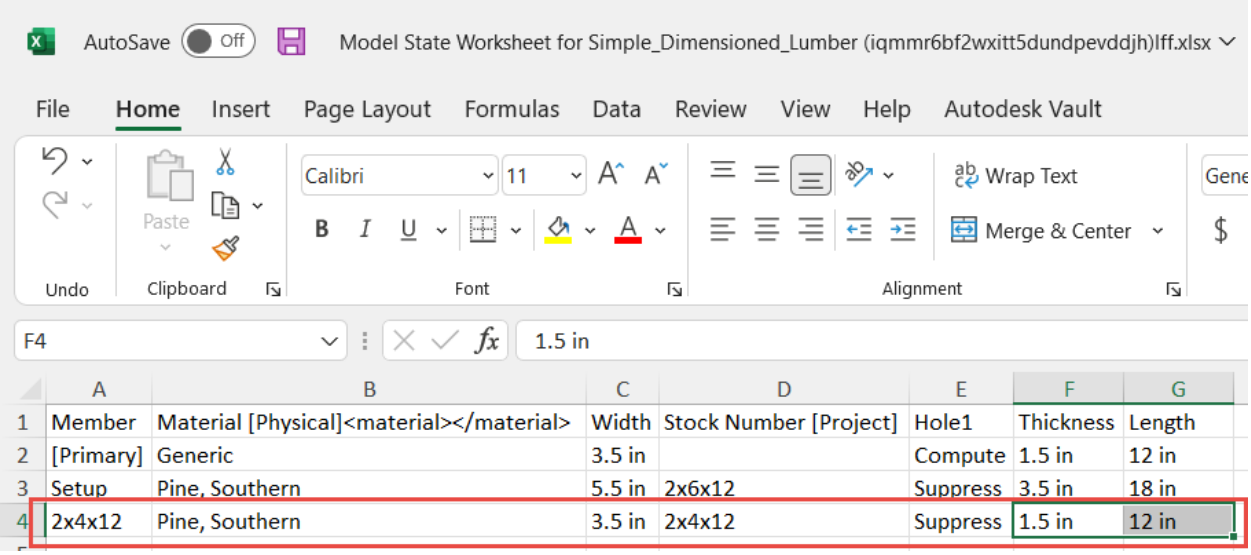
Excel Benefits

As I mentioned up above, the “Setup” State can help us think through a design and get organized. The biggest benefit, however, is that each modified value displays as a column when editing the Model States in Excel. While one can add the columns in Excel, once one gets a feel for the formatting, life is so much easier if the columns are added automatically.



	A	B	C	D	E	F	G
1	Member	Material [Physical]<material></material>	Width	Stock Number [Project]	d4	d5	Hole1
2	[Primary]	Generic	3.5 in		Width / 2 ul	1.5 in	Compute
3	Setup	Pine, Southern	5.5 in	2x6x12	1.741 in	1.353 in	Suppress
4							

EXCEL VIEW AFTER MODIFYING THE SETUP ROW



	A	B	C	D	E	F	G
1	Member	Material [Physical]<material></material>	Width	Stock Number [Project]	Hole1	Thickness	Length
2	[Primary]	Generic	3.5 in		Compute	1.5 in	12 in
3	Setup	Pine, Southern	5.5 in	2x6x12	Suppress	3.5 in	18 in
4	2x4x12	Pine, Southern	3.5 in	2x4x12	Suppress	1.5 in	12 in

EASILY ADD A NEW ROW IN EXCEL WITH ALL THE REQUIRED COLUMNS PRESENT

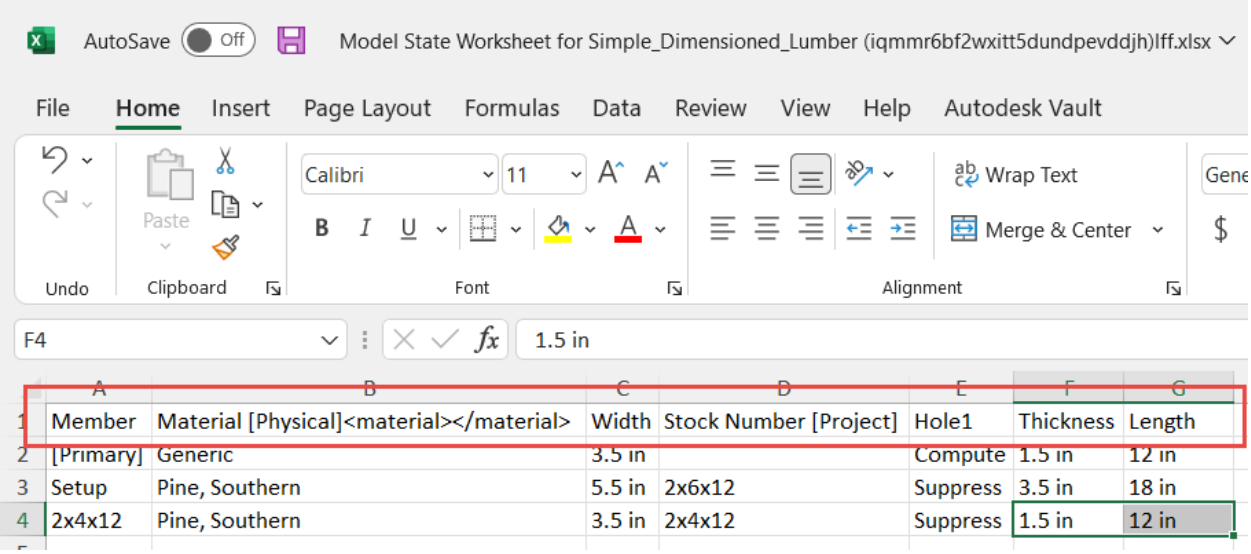
Everyone Loves a Good Remodel

Even though everybody is super good at what they do, but nobody is perfect. Sometimes we forget something. sometimes things simply just change. Either way, it will be very useful to reorganize the Model States' information when the need arises.

Rearranging Property Columns in Excel

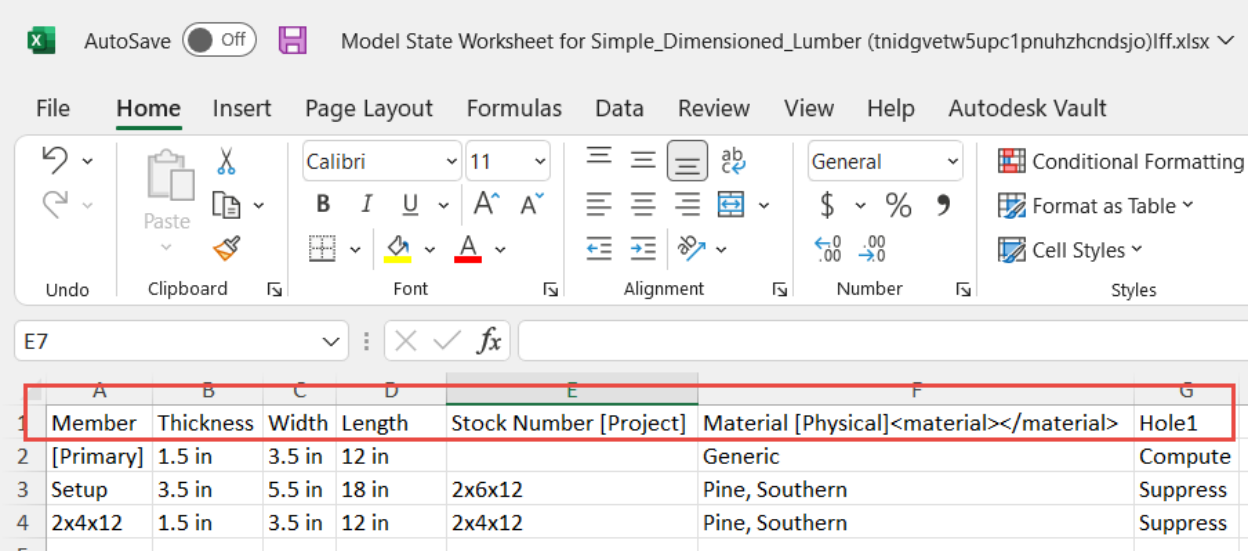
One of the most common reasons to reorganize the information within Model States is that the arrangement of the columns in Excel is not suited for efficient modification. For example, some of the size parameters may be together, but another one four columns away, with some iProperties sandwiched between them. While we can work that way, it is much easier to group common data together.

Thankfully it's super easy to rearrange columns when editing Model States in Excel. Right-click on the desired column heading (letter) and choose the Copy Option. Find the location you wish and Right-click on the column immediately right of that position and choose the Paste option. Repeat as needed until all the columns are ordered in the most efficient manner.



	A	B	C	D	E	F	G
1	Member	Material [Physical]<material></material>	Width	Stock Number [Project]	Hole1	Thickness	Length
2	[Primary]	Generic	3.5 in		Compute	1.5 in	12 in
3	Setup	Pine, Southern	5.5 in	2x6x12	Suppress	3.5 in	18 in
4	2x4x12	Pine, Southern	3.5 in	2x4x12	Suppress	1.5 in	12 in

EXCEL EXAMPLE OF ORIGINAL COLUMN CONFIGURATION

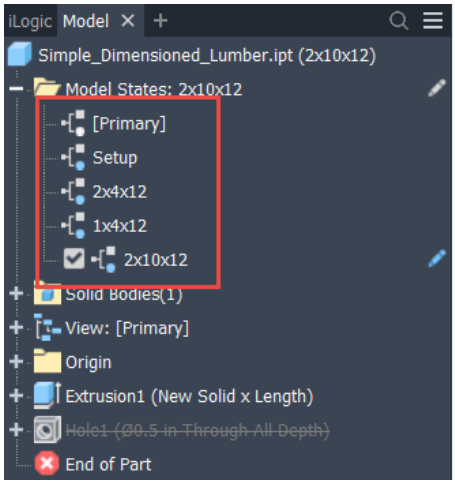


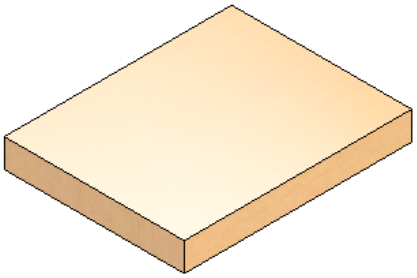
	A	B	C	D	E	F	G
1	Member	Thickness	Width	Length	Stock Number [Project]	Material [Physical]<material></material>	Hole1
2	[Primary]	1.5 in	3.5 in	12 in		Generic	Compute
3	Setup	3.5 in	5.5 in	18 in	2x6x12	Pine, Southern	Suppress
4	2x4x12	1.5 in	3.5 in	12 in	2x4x12	Pine, Southern	Suppress

EXCEL EXAMPLE WITH REARRANGED COLUMNS

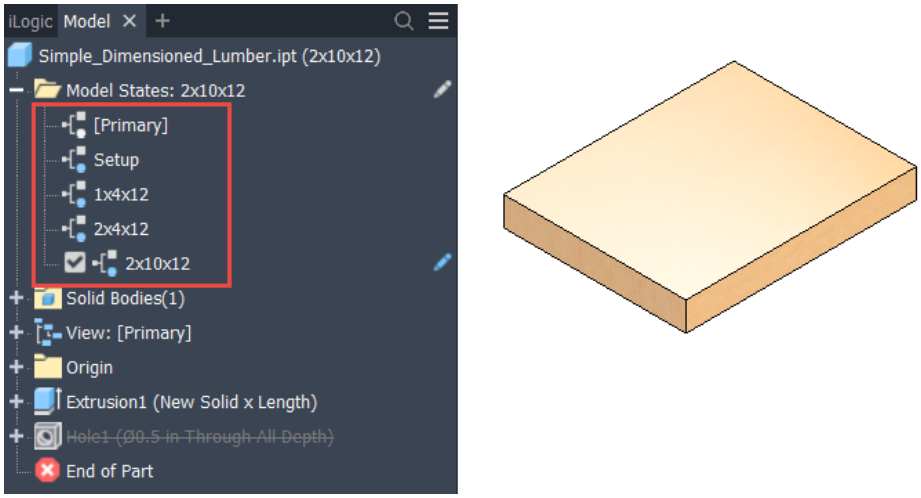
Rearranging Rows

While one can simply Left click, hold and drag Model States the Browser to rearrange them, which can be very beneficial like animating Model States in the next section, we can utilize the same procedure for copying Rows. Simply Right-click on the row one wishes to reposition and select the Copy Option. Find the desired location and Right-click on the row directly below and choose the Paste Option. This can be a useful technique when already performing modifications in Excel.





ORIGINAL MODEL STATE POSITION



POSITION AFTER LEFT-CLICK & DRAGGING THE MODEL STATES IN THE BROWSER

	A	B	C	D	E	F	G
1	Member	Thickness	Width	Length	Stock Number [Project]	Material [Physical]<material></material>	Hole1
2	[Primary]	1.5 in	3.5 in	12 in		Generic	Compute
3	Setup	3.5 in	5.5 in	18 in	2x6x12	Pine, Southern	Suppress
4	2x4x12	1.5 in	3.5 in	12 in	2x4x12	Pine, Southern	Suppress
5	1x4x12	0.75 in	3.5 in	12 in	2x4x12	Pine, Southern	Suppress
6	2x10x12	1.5 in	9.5 in	12 in	2x4x12	Pine, Southern	Suppress
7							

EXCEL EXAMPLE OF ORIGINAL ROW CONFIGURATION

	A	B	C	D	E	F	G
1	Member	Thickness	Width	Length	Stock Number [Project]	Material [Physical]<material></material>	Hole1
2	[Primary]	1.5 in	3.5 in	12 in		Generic	Compute
3	Setup	3.5 in	5.5 in	18 in	2x6x12	Pine, Southern	Suppress
4	1x4x12	0.75 in	3.5 in	12 in	2x4x12	Pine, Southern	Suppress
5	2x4x12	1.5 in	3.5 in	12 in	2x4x12	Pine, Southern	Suppress
6	2x10x12	1.5 in	9.5 in	12 in	2x4x12	Pine, Southern	Suppress
7							

EXCEL EXAMPLE OF REARRANGED ROWS

Bend it Like Beckham

When working with multi-operation designs, either sheet metal or progressively formed components, it can be helpful to assess the fabrication steps by animating the Model States in sequence.

Accessing Models States via the API

When Autodesk released Model States to Inventor, they also provided the ability to access Model States using the API. Thankfully accessing the Model States is fairly straightforward, via the ComponentDefinitions of Part and Assembly models. Here are some examples:

Example for accessing Model States for Parts or Assemblies:

```
'Set up the Document Settings to access the Model States
Dim oDoc = ThisDoc.Document
Dim oCompDef As ComponentDefinition = oDoc.ComponentDefinition

Dim oModelStates As ModelStates = oCompDef.ModelStates
Dim oModelState As ModelState


'Cycle through each Model State
For Each oModelState In oModelStates
```

Example for accessing Model States of Occurrences within Assemblies

```
Dim oDoc = ThisDoc.Document

Dim oCompDef As AssemblyComponentDefinition
oCompDef = oDoc.ComponentDefinition

Dim oOccs As ComponentOccurrences
oOccs = oCompDef.Occurrences

Dim oOcc As ComponentOccurrence

'This code will cycle through each ComponentOccurrence
'and display the name of each Model State in that Occurrence
For Each oOcc In oOccs

        'Must access the Occurrence Document
        oOccDoc = oOcc.Definition.Document
        'Must access the ComponentDefinition
        oOccCompDef = oOccDoc.ComponentDefinition
        'Can now access the ModelStates collection from
        'the Component Definition
        oOccModelStateCount = oOccCompDef.ModelStates.Count

        For i = 1 To oOccModelStateCount

                oModelState = oOccCompDef.ModelStates.Item(i)
                Logger.Info(oModelState.Name)
```

```
        Next

Next
```

Setting an iLogic Timer

Now that we can access the Model States for a given component, we will be able to animate them. However, computers can work REALLY fast, so in order to allow us to properly assess the viability of the design fabrication, we'll have to S…L…O…W… things down. We can do this by adding a bit of a delay to the code, by asking the computer to take a small rest (Sleep). That code is a pretty simple line that I gleaned from the internet.

```
'Delay the activation of the next Model State (in milliseconds)
Threading.Thread.Sleep(1500) 'in ms
```

Animating Sheet Metal Forming via Model States

With the ability to access the Model States and delay the computer between the activation of each model state, we're *almost* ready to animate these model states. The only thing we'll want to check is that the Model States are shown in the desired order, from top to bottom, or we'll get some "weird" forming results. Please refer to the previous section "Everyone Loves a Good Remodel – Rearrange Rows" to see how this can be accomplished.

Once the Model States have been rearranged, we can finally put all the iLogic code together to animate the Model States in sequence. Please refer to "Appendix B: iLogic Rule for Animating Model States" for the complete code.

```
'Cycle through each Model State and activate them in order
For Each oModelState In oModelStates
        If oModelState.Name <> "[Primary]" Then
        'If oModelState.Name <> "Master" Then 'For Inventor 2022
                oModelState.Activate
        End If

        'Update the document to view the active Model States
        InventorVb.DocumentUpdate()

        'Delay the activation of the next Model State (in milliseconds)
        Threading.Thread.Sleep(1500) 'in ms
Next
```

Extra Fun

Because I didn't read things very closely when I submitted this class, I didn't realize that I'd have **90 minutes** to fill, so here are some bonus topics that you may find beneficial…

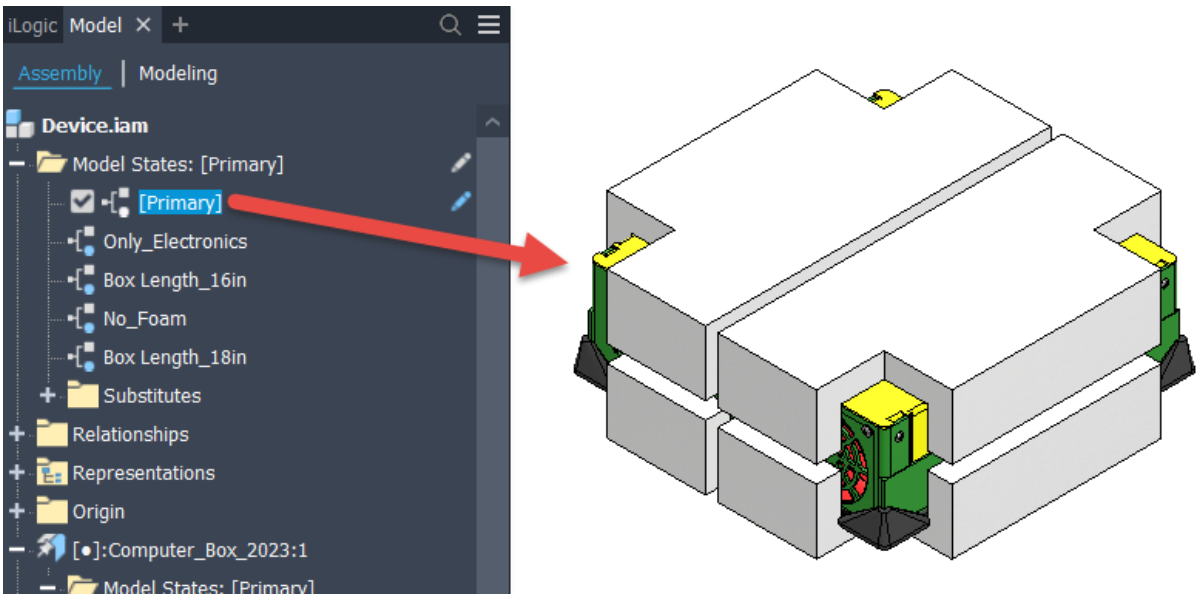
Linking Model States in Lieu of Derived Designs

Many people utilize the "Derive" tool to link designs together, which is an efficient way to link the values and geometry between models. For those not familiar with Inventor's "Derive" command but are familiar with AutoCAD; the AutoCAD functionality "XREF" is analogous to the "Derive" command. While the "Derive" command can still be utilized, there's a pretty hefty warning I want to offer and let you know that "Link"ing Model States together, may be a more reliable (and slightly tedious) workflow.

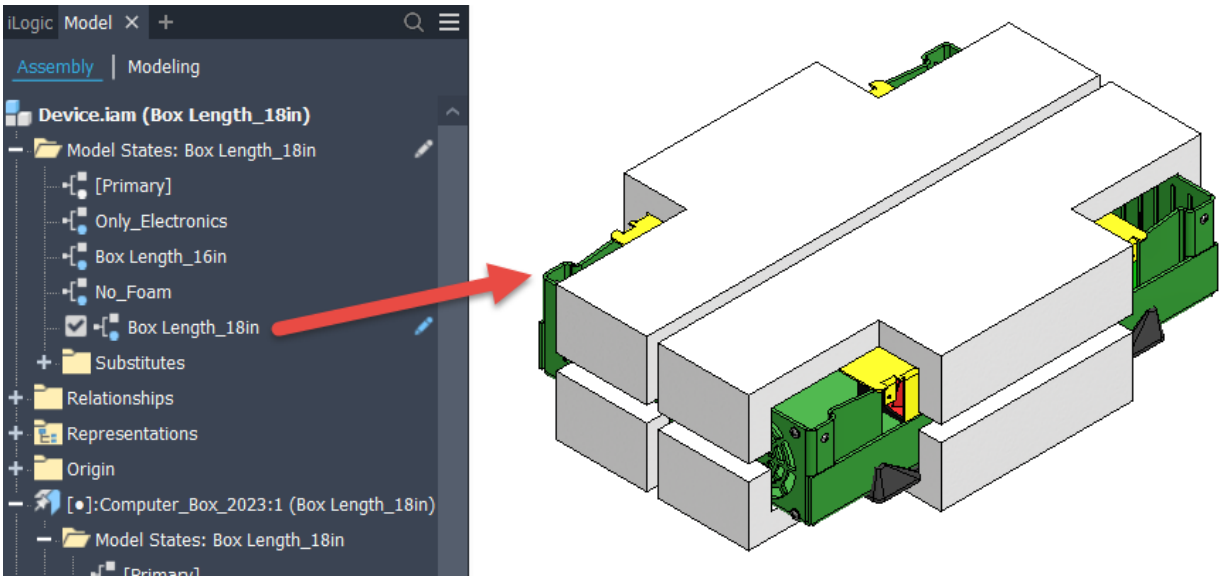
Warnings about "Derive" and Model States

The biggest caution when using the "Derive" command is that ONLY one Model State is referenced into the design. Additionally, ONLY the Model State linked at the time of "Derive" command execution is available in the referencing design.

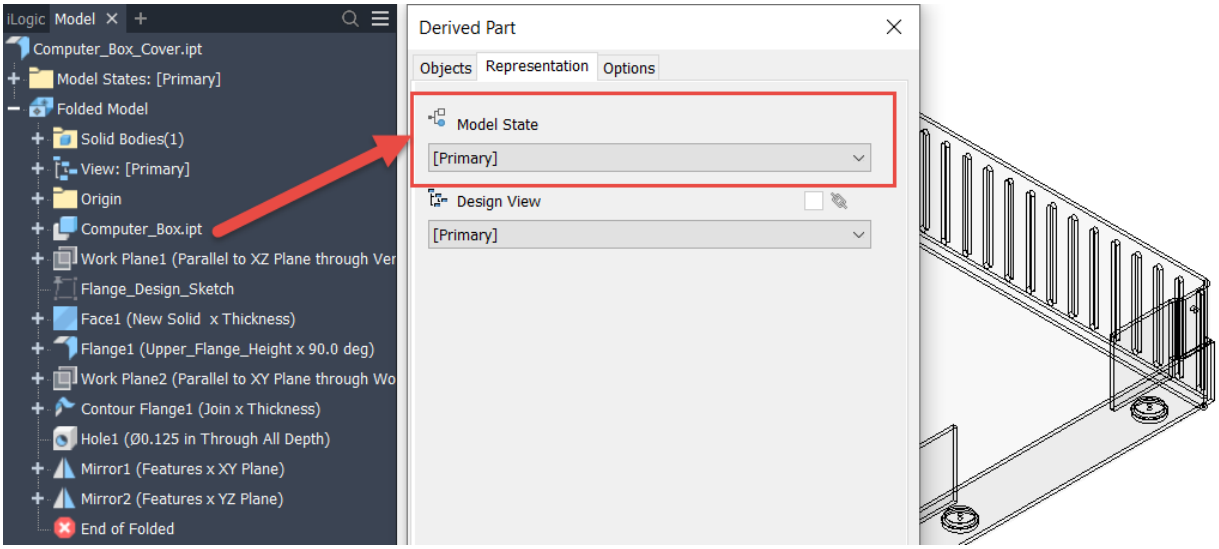
This means that if a different Model State is activated in the referenced model, the referencing model will still be using the original Model State, from the time of the "Derive". While I think this is appropriate, confusion can result if you're expecting the referencing design to update. One can edit the "Derive" and switch the referenced Model State, if that's desired.



ORIGINAL DERIVE RESULT



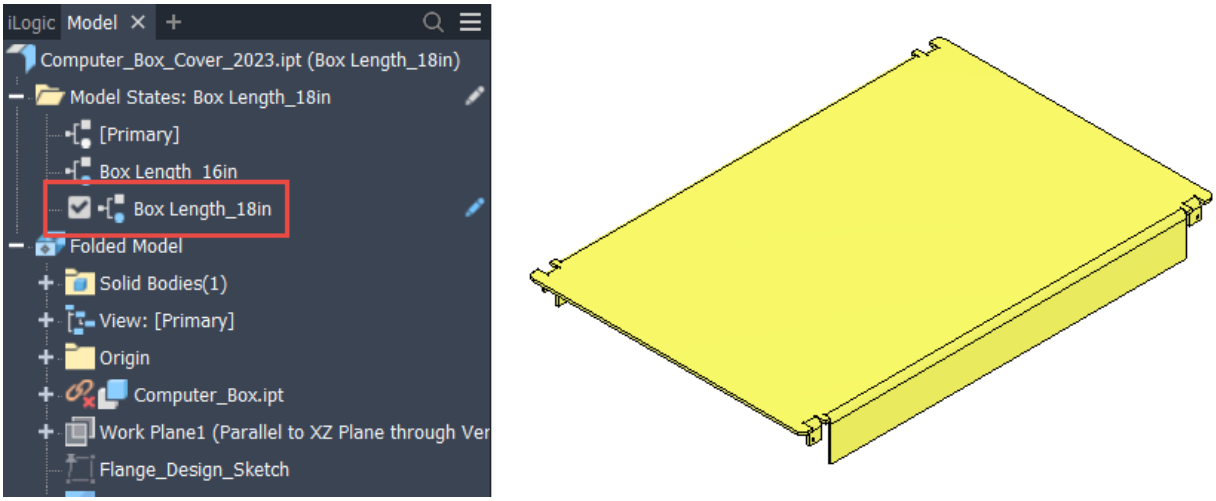
DERIVE RESULT AFTER MODEL STATE CHANGE IN THE REFERENCED DESIGN



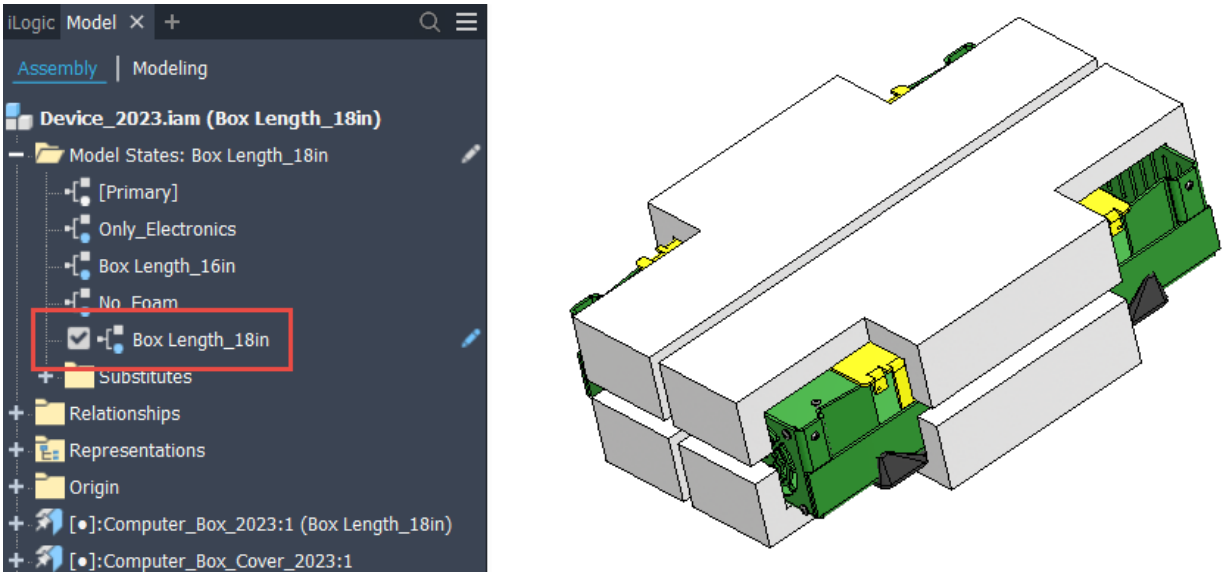
VIEW OF DERIVE DIALOG BOX

Linking Model States

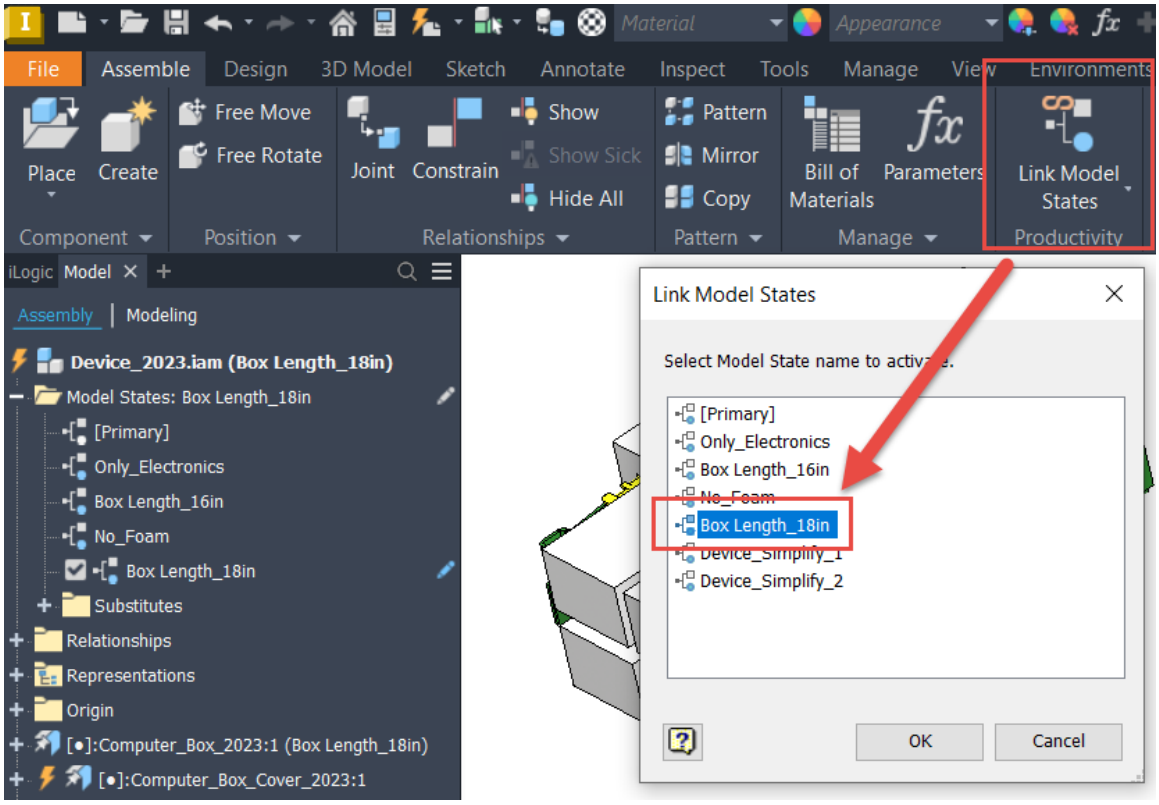
While the “Derive” command is limited to reference only one Model State, linking Model States allows Model States to be activated across many components. While Model State linking can take some effort to set up, if the Model State names are identical between components, they can all be activated simultaneously to an assembly level Model State change. One can link multiple Model States together, including across subassemblies, when creating Model State linking inside the upper-level assembly.



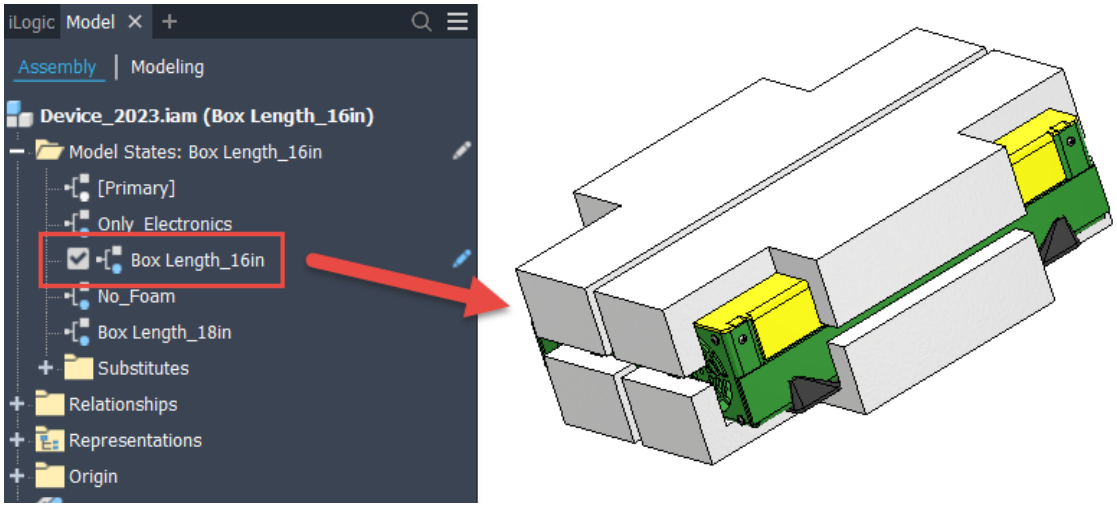
ENSURE THAT ALL DESIRED COMPONENTS HAVE IDENTICALLY NAMED MODEL STATES



CREATE AND IDENTICALLY NAMED MODEL STATE IN THE ASSEMBLY



LINK THE ASSEMBLY AND COMPONENT MODEL STATES



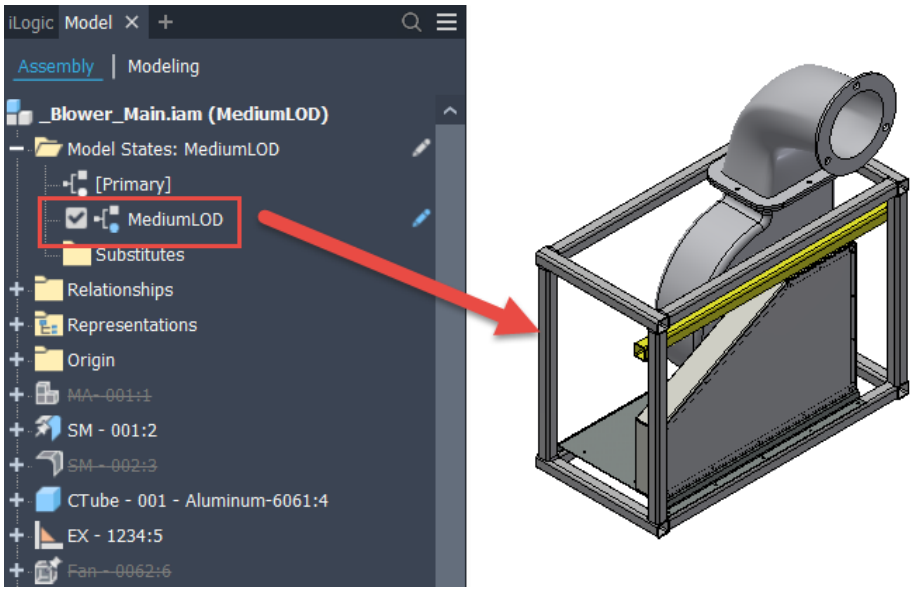
MODELS ADAPT SIMULTANEOUSLY TO ASSEMBLY MODEL STATE CHANGES AND MULTIPLE STATES CAN BE LINKED

Working with Converted Levels of Detail

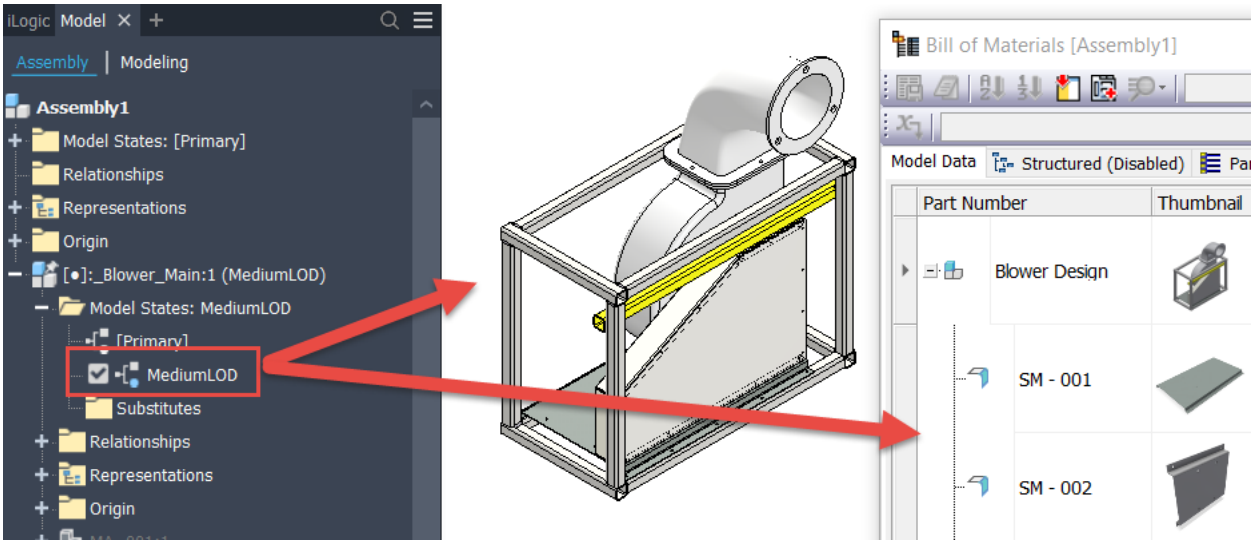
With the introduction of Model States, starting with Inventor 2022, Levels of Detail (LOD) are no longer available, with only custom LODs being migrated to custom Model States. As we saw in the first portion of this class, these converted LODs utilize the “Primary” Model State for BOM purposes. However, what if we want these converted LODs to behave like actual Model States? What options do we have at our disposal?

Converted LOD as a Subassemblies and Simplified Components

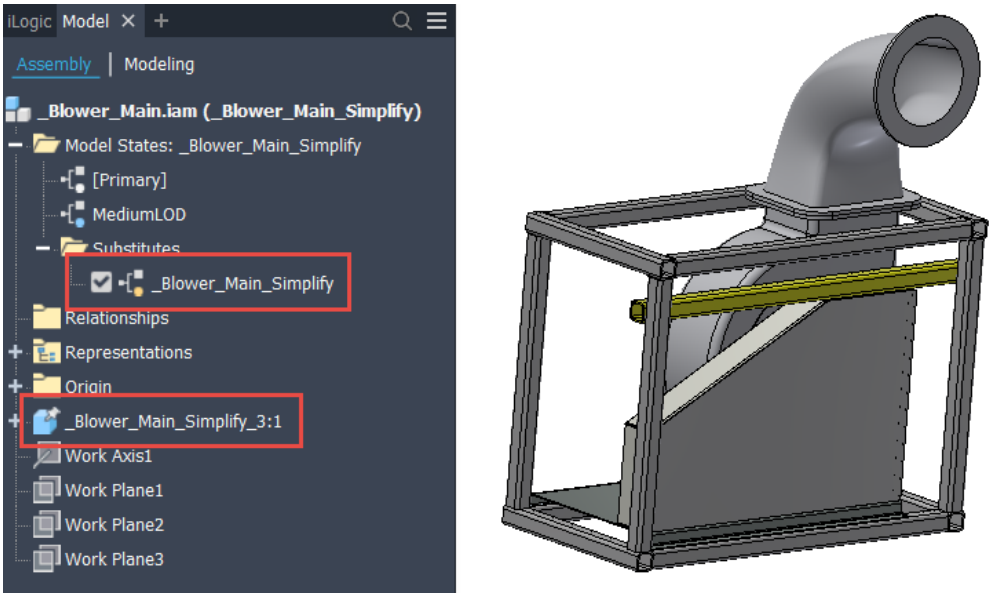
Since Model States, particularly Model State Substitutes, can be used as subassemblies, converted LODs can serve the same purpose. One can simply place an assembly with the converted LOD active, and it will have a simplified appearance, yet carry the whole BOM. One can also use the converted LOD as the basis for a Substitute Model State, using a simplified component, which will also bring across the complete BOM to a higher-level assembly.



CONVERTED LOD AS A CUSTOM MODEL STATE



CONVERTED LOD PLACED AS A SUBASSEMBLY

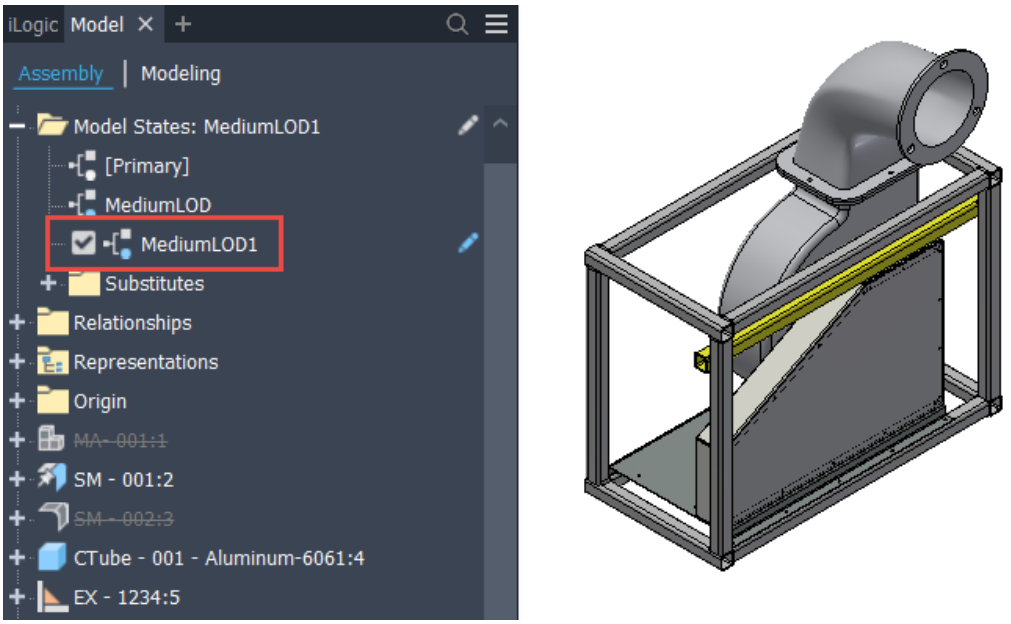


CONVERTED LOD AS A MODEL STATE SUBSTITUTE

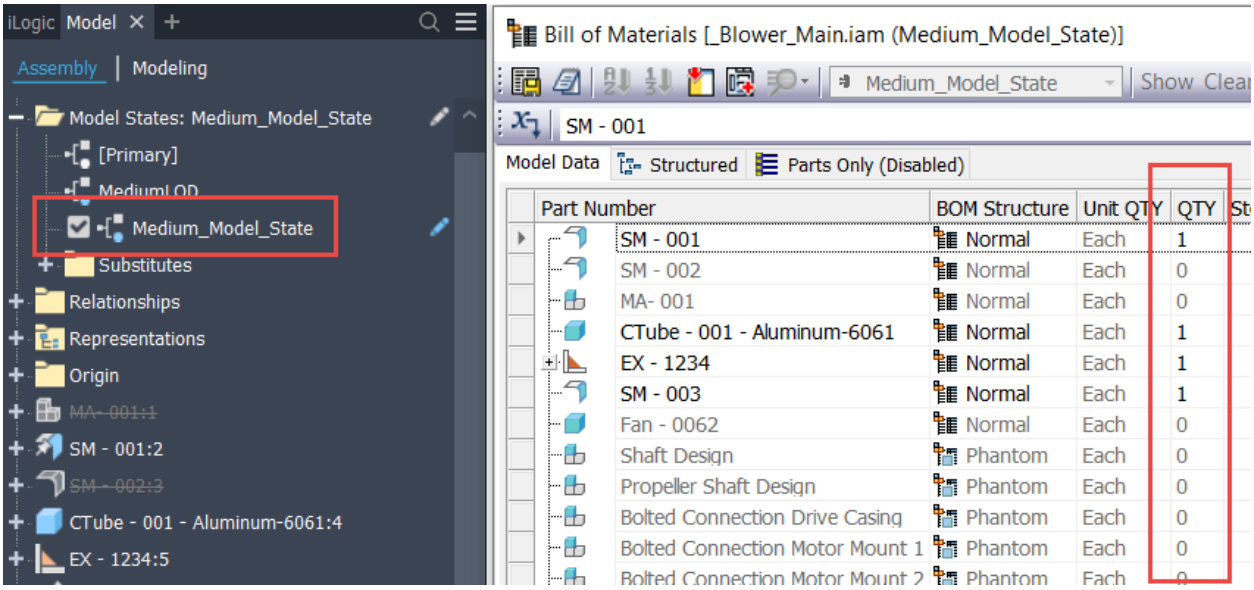
Converted LODs Behaving Like Proper Model States

Out-of-the-gate, these converted LODs must activate the “Primary” Model State to access the BOM, which will mimic the performance of LODs, prior to Inventor 2022. However, what if one wishes these converted LODs to behave like every other Model State? Is it possible to modify these converted LODs to set suppressed component quantities to 0?

The answer is YES! And this is really easy to accomplish, as one must only copy and rename the converted LODs, to make them behave like all other Model States.



RIGHT-CLICK AND CREATE A COPY OF THE CONVERTED LOD



RENAME THE COPIED MODEL STATE AND VERIFY THE RESULTING BOM VIEW

Method for Tracking Model States in Vault

One of the great limitations of Model States in Vault, is that we cannot tell which component's Model State is active inside an assembly, when viewing the "Uses" tab. While individual Model States can be identified with Vault Items, not everyone uses Items, and it is more useful to simply check the "Uses" tab.

Unfortunately, there is no simple solution to this, that I'm aware of. However, using a bit of creativity (and a fair amount of iLogic), a workaround exists that will pass along the unique component Model States used to the host assembly's iProperties. These iProperties can then be searched in Vault. Below is a brief explanation of the method.

Determining the Unique Model States

The components that are being used in an assembly can have several different types of Model States. If one is creating a single version component, then perhaps only the "Primary" Model State is used (and wouldn't needed to be tracked). If one has a multi-operation component, where each Model State represents a stage in the fabrication process, then there may only be one final Model State. If you've followed my naming convention from this class, you may have named that Model State "Final_Design" and we don't need to track that either. Really the only Model States that we'd like to track would be those where a component has several different variations, because we want to know exactly which version is being used in that particular assembly.

To accomplish this, we'll use some iLogic code to check all the top-level components in an assembly and determine the active Model State for each one. I'm using this approach because I'm assuming that we'll perform these steps at any desired assembly level. As long as the Model State is unique (not the "Primary", "Setup" or "Final_Design" States), then we'll add both the component File Name (without the path) and the Active Model State.

```
'Set up Array lists to capture the components that have the unique Model
States Active
Dim File_Names_Active_Model_States As New ArrayList


'defines backslash as the subdirectory separator
Dim strCharSep As String = System.IO.Path.DirectorySeparatorChar


'Cycle through all the top-level occurrences and collect the components into
the lists
For Each oOcc In oOccs

        'Check against the Primary, Master and Final_Design Model States
        If Not oOcc.ActiveModelState = "[Primary]" And Not
oOcc.ActiveModelState = "Master" _
                And Not oOcc.ActiveModelState = "Setup" And Not
oOcc.ActiveModelState = "Final_Design" Then
                'Find the file name, including the file extension to later
find in Vault
```

```
                oOccDoc = oOcc.Definition.Document
                'Find the postion of the last backslash in the path
                FNamePos = InStrRev(oOccDoc.FullFileName, "\", -1)
                'Get the file name with the file extension
                oOccDocName = Right(oOccDoc.FullFileName,
Len(oOccDoc.FullFileName) - FNamePos)
                'Add the component names and corresponding unique Model States
to the array lists

                'Check the current combination of name and model state against
the current list
                'Gather the current combination
                Dim ModelState_ID As String = oOccDocName & "_" &
oOcc.ActiveModelState

                'Check to see if the ArrayList contains the ModelState_ID and
add if it does not.
                If File_Names_Active_Model_States.Contains(ModelState_ID) =
False Then
                        File_Names_Active_Model_States.Add(ModelState_ID)
                End If
        End If
Next
```

Assigning the Unique Model States to the Host Assembly iProperties

There are a couple of different ways to capture and display the Model State data. I chose to use the host assembly, because, at this point, only the "Primary" model state is visible for components in Vault. This means that no matter which Model State is active for a particular component, the Vault "Uses" tab for an assembly, will only display that component's "Primary" Model State data. However, each assembly can display unique data in the Vault "Uses" tab, so that's where I decided to place this data.

Another decision is HOW to display this assembly data. All of the model states could be stored in a single assembly custom iProperty, delimited in some fashion. While is a really easy way to search for and find data, I was a little concerned about the potential character counts that could be populated into these iProperties. A second method is to create a unique assembly iProperty for each component's Model State. This may be a little more difficult to search in Vault but will not run into any kind of character limit this. I'll present the code examples below.

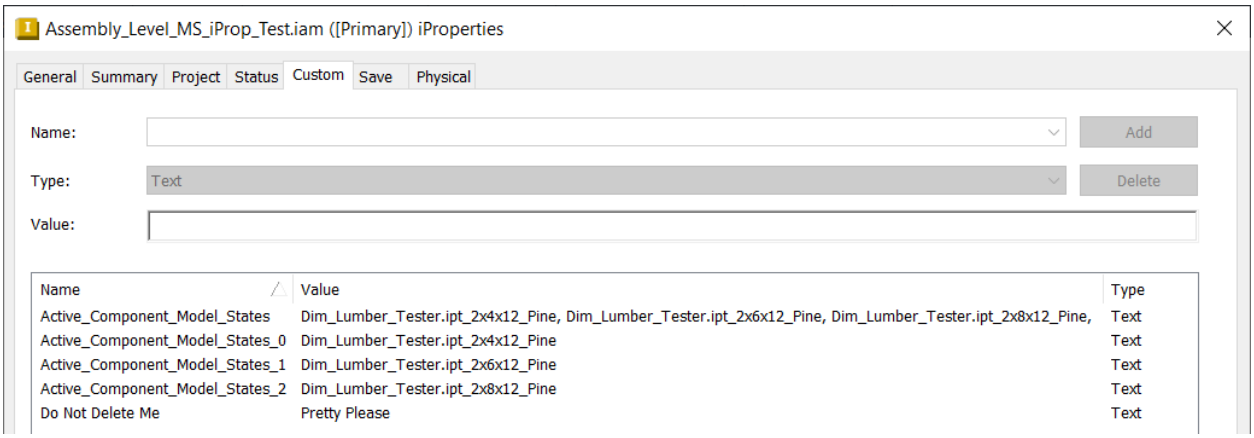
Example of Assigning to a Single Custom iProperty (Note: I've left some testing code lines in there for verification):

```
'Assign the values from the array lists to an iProperty in the Assembly
iProperties.Value("Custom", "Active_Component_Model_States") = ""

For i = 0 To File_Names_Active_Model_States.Count - 1
'       Logger.Info(i) 'For testing only
```

```
'       Logger.Info(File_Names(i)) 'For testing only
        'Get the current value for the custom iProperty
        Initial_String = iProperties.Value("Custom",
"Active_Component_Model_States")

        'Add the new values to the text string
        iProperties.Value("Custom", "Active_Component_Model_States") =
Initial_String & File_Names_Active_Model_States.Item(i) & ", "
Next
```

Example of creating a unique custom iProperty for each component (Note: see the full code example in Appendix B – iLogic Rule for Creating a Model State iProperty for Each Component, which shows how to clear out any of these pre-existing custom iProperties):

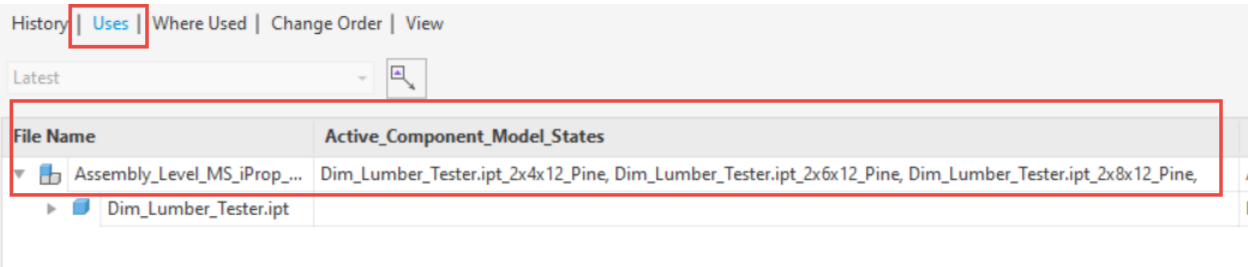
```
'Assign the values from the Array to an iProperty in the Assembly
For i = 0 To File_Names_Active_Model_States.Count - 1
'       Logger.Info(i) 'For testing only
'       Logger.Info(File_Names(i)) 'For testing only
        'Create a new iProperty for each component in the array lists
        iProperties.Value("Custom", "Active_Component_Model_States" & "_" & i)
= File_Names_Active_Model_States.Item(i)
Next
```



EXAMPLE OF IPROPERTIES WITHIN INVENTOR

Viewing the Custom iProperties in Vault

Setting up the iProperties in the host assembly is the major portion of this workflow, but we must still create and map the properties in Vault, so we can see them in the “Uses” tab. There’s a plethora of information for this topic on the web, so please see the video in Appendix A - Inventor - Method for Showing Active Model States for Vault Uses to see an example.



IPROPERTY VIEW IN VAULT

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 I take!
- Tim Wilson, my current boss, and my former bosses for allowing me the time and space to explore these zany ideas
- James Jung, one of my Vault colleagues who assisted me with the Vault property mapping and searches
- All of my many customers who have taken classes and asked really great questions to help drive my research.
- Andrew Humiston, one of my customers who has really pushed Model States and was the genesis of the idea to display a custom iProperty for Model States in Vault

Appendix A: Weblinks to Video Demonstrations

Video demonstrating the use of a setup row for Model States
https://youtu.be/ZLkmzULtURo

Video demonstrating how to rearrange rows, and rows & columns in Excel
https://youtu.be/zD-f6ERw0Dk

Video on animating Model States (Note: one must now update the model after each Model State activation)
https://youtu.be/y1TpHenyiao

Video on working with converted Levels of Detail
https://youtu.be/4zS4zUs5-Yg

Video on custom iProperties in Vault
https://youtu.be/Xralyt8JOcc

Appendix B: Sample iLogic Code

iLogic Rule for Animating Model States

```
'Set up the Document Settings to access the Model States
Dim oDoc = ThisDoc.Document
Dim oCompDef As ComponentDefinition = oDoc.ComponentDefinition

Dim oModelStates As ModelStates = oCompDef.ModelStates
Dim oModelState As ModelState


'Cycle through each Model State and activate them in order
For Each oModelState In oModelStates
	If oModelState.Name <> "[Primary]" Then
	'If oModelState.Name <> "Master" Then 'For Inventor 2022
		oModelState.Activate
	End If

	'Update the document to view the active Model States
	InventorVb.DocumentUpdate()

	'Delay the activation of the next Model State (in milliseconds)
	Threading.Thread.Sleep(1500) 'in ms
Next
```

iLogic Rule for Single Model State iProperty

```
'Cycle through all of the components and see if they have the Primary Model
State active
'If they don't then record the component name and model state in a custom
Assembly iProperty
'This will only be used for the immediate component level, not the leaf
components

'Set up the necessary AP calls
Dim oDoc = ThisDoc.Document

Dim oCompDef As AssemblyComponentDefinition
oCompDef = oDoc.ComponentDefinition

Dim oOccs As ComponentOccurrences
oOccs = oCompDef.Occurrences

Dim oOcc As ComponentOccurrence


'Set up Array lists to capture the components that have the unique Model
States Active
Dim File_Names_Active_Model_States As New ArrayList


'defines backslash as the subdirectory separator
Dim strCharSep As String = System.IO.Path.DirectorySeparatorChar


'Cycle through all the top-level occurrences and collect the components into
the lists
For Each oOcc In oOccs

        'Check against the Primary, Master and Final_Design Model States
        If Not oOcc.ActiveModelState = "[Primary]" And Not
oOcc.ActiveModelState = "Master" _
                And Not oOcc.ActiveModelState = "Setup" And Not
oOcc.ActiveModelState = "Final_Design" Then
                'Find the file name, including the file extension to later
find in Vault
                oOccDoc = oOcc.Definition.Document
                'Find the postion of the last backslash in the path
                FNamePos = InStrRev(oOccDoc.FullFileName, "\", -1)
                'Get the file name with the file extension
                oOccDocName = Right(oOccDoc.FullFileName,
Len(oOccDoc.FullFileName) - FNamePos)
                'Add the component names and corresponding unique Model States
to the array lists

                'Check the current combination of name and model state against
the current list
                'Gather the current combination
```

```
                Dim ModelState_ID As String = oOccDocName & "_" &
oOcc.ActiveModelState

                'Check to see if the ArrayList contains the ModelState_ID and
add if it does not.
                If File_Names_Active_Model_States.Contains(ModelState_ID) =
False Then
                        File_Names_Active_Model_States.Add(ModelState_ID)
                End If
        End If
Next

Logger.Info(File_Names_Active_Model_States.Count)


'Assign the values from the array lists to an iProperty in the Assembly
iProperties.Value("Custom", "Active_Component_Model_States") = ""

For i = 0 To File_Names_Active_Model_States.Count - 1
'       Logger.Info(i) 'For testing only
'       Logger.Info(File_Names(i)) 'For testing only
        'Get the current value for the custom iProperty
        Initial_String = iProperties.Value("Custom",
"Active_Component_Model_States")

        'Add the new values to the text string
        iProperties.Value("Custom", "Active_Component_Model_States") =
Initial_String & File_Names_Active_Model_States.Item(i) & ", "
Next


iLogicVb.UpdateWhenDone = True
```

iLogic Rule for Creating a Model State iProperty for Each Component

```
'Cycle through all of the components and see if they have the Primary Model
State active
'If they don't then record the component name and model state in a custom
Assembly iProperty
'This will only be used for the immediate component level, not the leaf
components

'Set up the necessary AP calls
Dim oDoc = ThisDoc.Document

Dim oCompDef As AssemblyComponentDefinition
oCompDef = oDoc.ComponentDefinition

Dim oOccs As ComponentOccurrences
oOccs = oCompDef.Occurrences

Dim oOcc As ComponentOccurrence


'Set up Array lists to capture the components that have the unique Model
States Active
Dim File_Names_Active_Model_States As New ArrayList


'defines backslash as the subdirectory separator
Dim strCharSep As String = System.IO.Path.DirectorySeparatorChar


'Cycle through all the top-level occurrences and collect the components into
the lists
For Each oOcc In oOccs

        'Check against the Primary, Master and Final_Design Model States
        If Not oOcc.ActiveModelState = "[Primary]" And Not
oOcc.ActiveModelState = "Master" _
                And Not oOcc.ActiveModelState = "Setup" And Not
oOcc.ActiveModelState = "Final_Design" Then
                'Find the file name, including the file extension to later
find in Vault
                oOccDoc = oOcc.Definition.Document
                'Find the postion of the last backslash in the path
                FNamePos = InStrRev(oOccDoc.FullFileName, "\", -1)
                'Get the file name with the file extension
                oOccDocName = Right(oOccDoc.FullFileName,
Len(oOccDoc.FullFileName) - FNamePos)

                'Check the current combination of name and model state against
the current list
                'Gather the current combination
                Dim ModelState_ID As String = oOccDocName & "_" &
oOcc.ActiveModelState
```

```
                'Check to see if the ArrayList contains the ModelState_ID and
add if it does not.
                If File_Names_Active_Model_States.Contains(ModelState_ID) =
False Then
                        File_Names_Active_Model_States.Add(ModelState_ID)
                End If
        End If
Next

Logger.Info(File_Names_Active_Model_States.Count)


'Remove any pre-existing iProperties of this type, in case someone switches
'a component to use "[Primary]", "Master" or the "Final_Design" Model States
Dim oPropSets As PropertySets = oDoc.PropertySets
Dim oUserDefProps As PropertySet = oPropSets.Item("Inventor User Defined
Properties")
For Each oiProp In oUserDefProps
        If oiProp.Name.Contains("Active_Component_Model_States") Then
                oiProp.Delete
        End If
Next


'Assign the values from the Array to an iProperty in the Assembly
For i = 0 To File_Names_Active_Model_States.Count - 1
'       Logger.Info(i) 'For testing only
'       Logger.Info(File_Names(i)) 'For testing only
        'Create a new iProperty for each component in the array lists
        iProperties.Value("Custom", "Active_Component_Model_States" & "_" & i)
= File_Names_Active_Model_States.Item(i)
Next


iLogicVb.UpdateWhenDone = True
```