AUTODESK UNIVERSITY

TR500670

Advanced Inventor Modeling for InfraWorks

Danny Lewis
Hatch

Learning Objectives

- Build Inventor models for use within Infraworks bridge and generic object workflows
- Develop and identify limitations of iLogic codes that work within Infraworks
- Create models with a high-degree of flexibility and adaptability in Infraworks
- Change Inventor model body colors and turn on/off Inventor components in Infraworks

Description

Inventor users: Have you heard of InfraWorks? And InfraWorks/Revit users: Have you heard of Inventor? This class bridges the knowledge gap that exists between Inventor and InfraWorks users, providing an advanced demonstration of the powerful functionality achieved when using these two products together. I'll show how to create dynamic Inventor models with capabilities that function once brought into InfraWorks. Instruction will include setting up iLogic codes to function in InfraWorks/Inventor server; providing examples of codes proven to work; showing how these models can be 'Flexed' and 'Instanced' in InfraWorks; highlighting the downstream benefits of models in Inventor/InfraWorks; and propagating models into Civil3D and Revit for further detailing and project drawings. For anyone curious about my 'street cred' for the content, be sure to check out my AU 2021 presentation titled Multibridge Cable Stay Intelligent Models in InfraWorks, Inventor, Civil 3D, and Revit.

Speaker

Nicknamed: “The Inventor Guru” at Hatch, Danny is well versed with a number of Autodesk software through his several years of experience in industry. Danny has worked on small machine design projects leveraging the Inventor HSM tools to develop CNC codes to model data all the way up to developing complex parametric models for use in gigantic InfraWorks projects.

He’s implemented Vault and setup entire Inventor ecosystems from scratch at several different companies. Throughout his career, he has setup robust & complicated parametric models leveraging everything from Excel, iLogic, VBA, Point Clouds, and anything else he can find. Danny's current role involves finding new ways to have detailed software like Inventor and AutoCAD integrate fully with larger infrastructure software such as Revit, Civil3D, Navisworks, and recently InfraWorks.

When Danny isn't busy modeling up things for clients or crazy contraptions for himself, he's busy goofing around with his two little girls, wife and baby boy at one of the local playgrounds. Danny is now serving as the Hatch Global Lead - Autodesk and coordinating all Autodesk software applications for projects around the globe and spanning numerous disciplines.

Preamble about InfraWorks, Inventor, and iLogic

I would encourage those reading this handout to first watch the presentation video associated with this class as I discuss a lot of the reasoning of WHY you'd want to use the tools listed below. In short, the headings listed below are like **mini-recipes** that users can use to create really cool features in their own Inventor models (and therefor their InfraWorks models). This class was not intended to be an in-depth dive on how to actually click buttons in Inventor and create the models… there's plenty of classes that teach basic Inventor modeling. Rather, this class is intended to instruct on what is necessary to do to the models so that they retain as much functionality as possible when they are used in InfraWorks.

The code below should be fully functioning, but if there's items that just aren't working for any reason, I would invite readers to go onto the Autodesk Forums and post a snippet of your code for others to try and diagnose. Feel free to tag me into the post, but no guarantees that I'll have the time to respond in a prompt timeline.

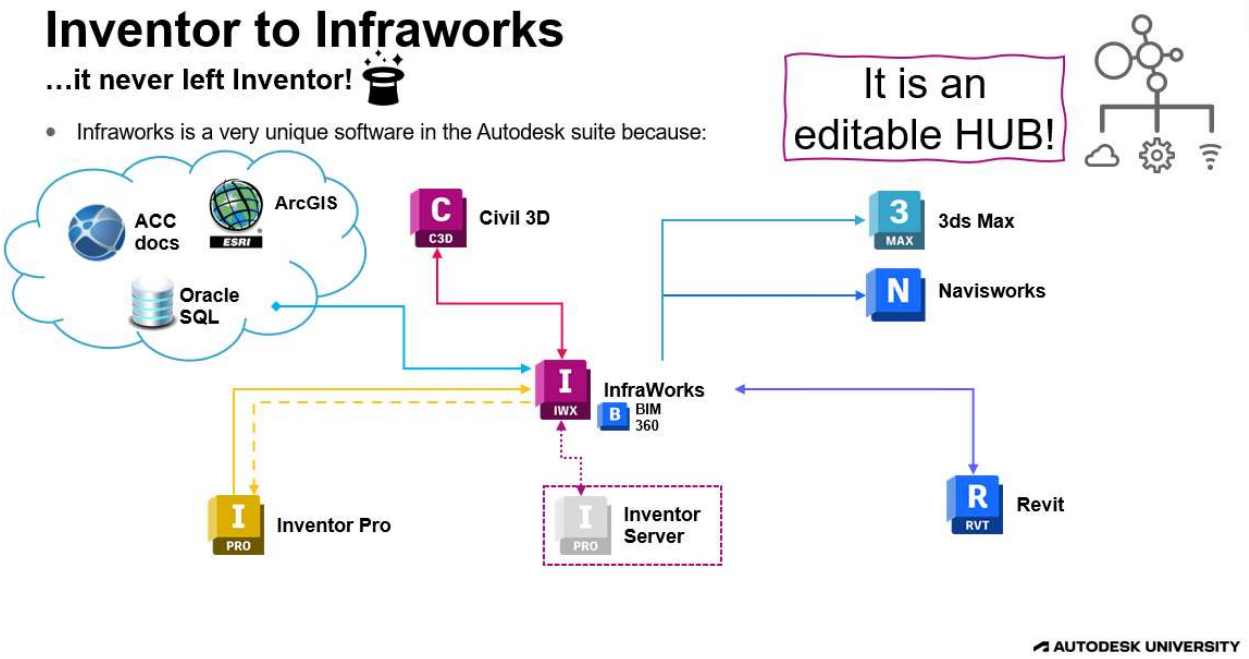
As well, if you are using InfraWorks… you can open up any of the default models from InfraWorks into Inventor and you'll see similar code in those models that you can directly reference.

The limits for what users can do with even the few 'tools' provided here are pretty wide.

I would hope that these snippets and explanations of what's going on in the background prevent countless hours of beating your head against a wall to figure "why the @#$*#@ isn't this stupid model working" when it lands into InfraWorks.

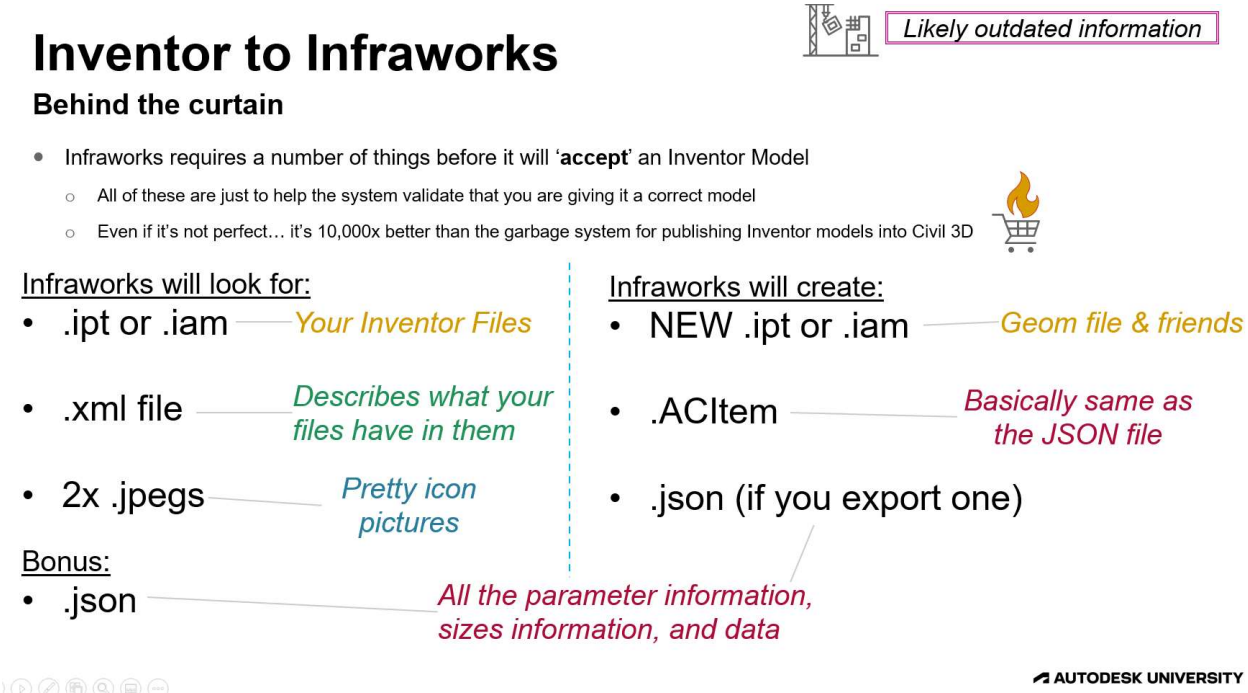
Path For Models To Go From Inventor To InfraWorks

As mentioned in the presentation (a reoccurring comment I'll make)… Inventor is actually baked into InfraWorks; so the term 'importing it into InfraWorks' isn't really accurate. You're just importing Inventor models back into Inventor. This means that while the file format doesn't change, it WILL take the files you give them and put them into the Inventor version that is associated with InfraWorks. If you're running InfraWorks 2023… the Inventor files will be Inventor 2023. You can still import older versions into InfraWorks, but you won't be able to open them on the way out anymore.



The other things that you import are more of a 'definition' that you're providing to InfraWorks as it begins to take control of the Inventor models. What type of file is this, what does it have associated with it, where are the textures that it references (broken.. but in theory that's what its supposed to do), that's what that data is providing.

The predominant way to put these files together is to use the Infrastructure or InfraWorks Add-in Inventor (not to be confused with the Infrastructure Parts Modeler tool… which is awful). Doing this task is covered well in other AU presentations, specifically the one I did in 2021, but essentially you click the add-in, it asks you for a file name, then **it creates the jpeg and .xml file**. The need for this add-in was removed in 2023+ versions of InfraWorks, but it's worth mentioning because although you don't need the add-in anymore… all the same files are still being created and referenced; just in the background of InfraWorks now.
At Hatch, we required more control of the contents of these add-in files (specifically because we were authoring the size information while still in Inventor), and so others may also find the need as well and end up going the same route.



Once you've got the xml/jpeg files created, then the most important file would be **JSON files**. **These files contain ALL of the size information, UX layouts, etc. and are often as important, if not more important, than the Inventor file being imported**.

At Hatch, we had tried to gather together exported JSON files into a directory and control what version they were at, but then reading the data in any meaningful way was hopeless. Controlling the JSON content was the primary reason that we ended up building our own tool.

Creating Simple Toggle In Inventor

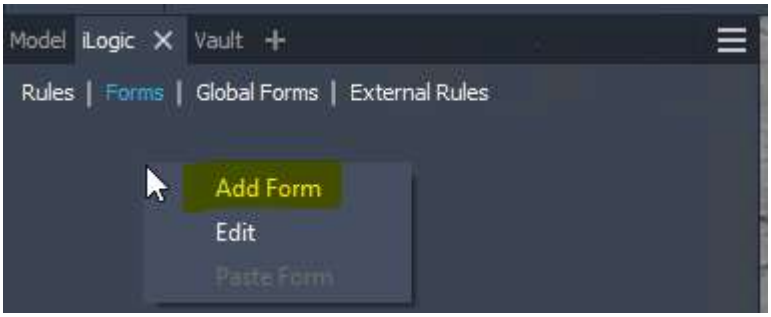
Now that you've figured out "how the sausage is made" for getting the Inventor content into InfraWorks, we can begin discussing some of the fancy models you can make. First step in that is to be able to 'flex' your model within Inventor (similar to how you eventually would in InfraWorks) to see if it explodes unexpectedly. Inventor is a very power software, but if you create models with circular references or references that break once you go to a certain number, then it's not going to be very useful when in InfraWorks.

Common issues you'd run in to would be putting an angle dimension onto a feature:
You can't dimension negative angles in Inventor, so if you want to be able to have +/- deg on a feature, then you need to dimension it so that it is 90deg+{parameter value}.

Simple things like that are easy to miss when you make a model and import it in, but that's why the toggles are so important.

In the video, I show an example of making a toggle, but the two big highlights would be:

1. Create a parameter that you're then going to toggle. For simple stuff, the parameter can just be a unitless value that you bounce between 0 and 1. To make life easier… make it a multi-value parameter and have it where 0 and 1 are the ONLY values it can take. You can limit the values of a parameter once in InfraWorks, but it's often better to just put proper limits in Inventor first.
2. Once you have a parameter, create a FORM in the iLogic section {shown below}. If the parameter is listed as 'KEY' (which it would need to be to show up in InfraWorks), then it shows up as an option for the form.



Once you have the form setup… You've got your toggle. In the video I then show a couple different ways to manipulate things with the toggle, but that greatly depends on what you want to do with your new button. If you want to just turn one thing off/on; then you won't even need to do any iLogic coding.

To have this new 'toggle' work in InfraWorks, you'd just tag it as a key column (in Inventor) and then set it as a toggle in InfraWorks.

Code for toggling lots of features at the same time

As demonstrated in the video(*hint hint), just having code that turns off features can be a slow operation if it is executed with the toggle command mentioned in the previous section. For one or two features being suppressed, it's fine, but **once you start using even just a few more features than that it quickly becomes very slow**. To rectify this, the solution is to group a bunch of features together into a single selection set and then have that selection set be toggled on and off as a group.Using this method is about as fast as you would have with the single toggle on/off, but obviously works for hundreds or thousands of features at the same time if you want.

The code to do this is listed below and without getting into too many details the gist of what it's doing is:

1. Define your selection sets. These are just transient objects collections by the definition.

```
'These are what are going to temporarily store all of our objects that we will then slecect and turn off
Dim oSelectOccOn As ObjectCollection = ThisServer.TransientObjects.CreateObjectCollection
Dim oSelectOccOff As ObjectCollection = ThisServer.TransientObjects.CreateObjectCollection
```

2. Then you'll set up all of your cases of what you want to have turned on and off and add them to those selections. In the example, I put them all into an array list and then I add to the list as I go, but you can use whatever you want as long as you've got a list of features put together.

```
'A list of params that we will toggle
Dim oTurnOff = New ArrayList
Dim oTurnOn = New ArrayList
oTurnOff.Clear
oTurnOn.Clear
```

Setup a bunch of cases for when features are On or Off

```
If Arrow Then
    Select Arrow_Type
    Case 0
        oTurnOn.Add("Arrow Tip")
        oTurnOff.Add("Arrow Stem")
        oTurnOff.Add("Arrow Tip Fill")
        oTurnOff.Add("Arrow Radii")
    Case 1
        oTurnOn.Add("Arrow Tip")
        oTurnOn.Add("Arrow Stem")
        oTurnOff.Add("Arrow Tip Fill")
        oTurnOff.Add("Arrow Radii")
    Case 2
        oTurnOn.Add("Arrow Tip")
        oTurnOn.Add("Arrow Stem")
        oTurnOn.Add("Arrow Tip Fill")
        oTurnOff.Add("Arrow Radii")
    Case 3
        oTurnOff.Add("Arrow Tip")
        oTurnOn.Add("Arrow Stem")
        oTurnOn.Add("Arrow Tip Fill")
        oTurnOn.Add("Arrow Radii")
    End Select
Else
```

```
ElseIf FeatureName = Check & Sign_Style & " Rads" Then
    If Rads Then
        Logger.Debug(FeatureName & "++ON")
        oTurnOn.Add(FeatureName)
    Else
        Logger.Debug(FeatureName & "--Off")
        oTurnOff.Add(FeatureName)
    End If
```

3. The next bit of code has the system cycling through all the part features in my model and if it's already turned off, leave it off otherwise turn it to the list that needs to be turned off. This is just a nice to have where your computer won't see things flicking on and off unnecessarily.

```
Dim oFeatures As PartFeatures = oDoc.ComponentDefinition.Features
```

```
For Each oPartFeature As PartFeature In oFeatures
    For Each name As String In oTurnOff
        If oPartFeature.Name = name Then
'           Logger.Debug("Turn-Off: " & name)
            If Not oPartFeature.Suppressed Then
            oSelectOccOff.Add(oPartFeature)
            End If
        End If
    Next
    For Each name As String In oTurnOn
        If oPartFeature.Name = name Then
'           Logger.Debug("Turn-ON: " & name)
            If oPartFeature.Suppressed Then
            oSelectOccOn.Add(oPartFeature)
            End If
        End If
    Next
Next
```

4. The last step is to take that selection list and do the suppress or unsuppressed features on mass with it.

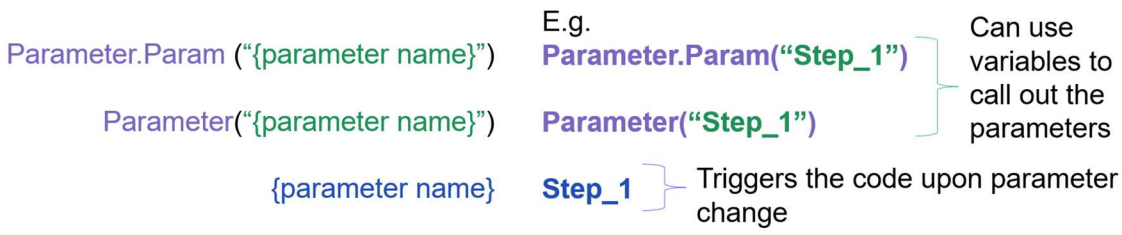
```
'Turn Off Selected Parts
oDoc.ComponentDefinition.SuppressFeatures(oSelectOccOff)

'Turn On Selected Parts
oDoc.ComponentDefinition.UnsuppressFeatures(oSelectOccOn)
```

Blue parameters vs. param()

One thing you'll run into very quickly when you start to do the programming is that there's a couple different methods to call out the parameters. At first, I didn't think there was any difference to them **however** once you start to play with him a little bit more, you realize that your code is doing funny things.

In a quick summary, Inventor is set up to call upon the parameters differently whether you're actually calling the specific parameter name (**The blue parameters**) then when you're just calling up a parameter from one of the object identifiers. **the blue parameters are set to trigger that specific ilogic code it's located in if they change their value.** this can be a good thing because setting up by Logic triggers can be a bit cumbersome and it allows you to control when things trigger, **but** it can also be a pain because you're code could want to trigger more than once sometimes. This is especially bad when you have multiple ilogic codes, with each one having blue parameters in it that are potentially calling up the other codes creating a circular loop.

In the end, it's not that difficult... it's just something to be aware of when you're writing your code.




```
StepX_Width = Parameter(This_Step & "_Width")
StepX_Height = Parameter(This_Step & "_Height")
StepX_LHSideAngle = Parameter(This_Step & "_LHSideAngle")
StepX_RHSideAngle = Parameter(This_Step & "_RHSideAngle")
GirderX_SlopeInDeg = Parameter("Girder" & Step_Num & "_SlopeInDeg")
```

```
If AbtSlopes_Toggle = True Then
    For x = 1 To MaxGirders
    Parameter.Param("Step" & x & "_LHSideAngle").Value = Math.PI / 2
    Parameter.Param("Step" & x & "_RHSideAngle").Value = Math.PI / 2
    Next
End If
```

‘ThisApplication’ vs. ‘ThisServer’

This next “ trick”, for lack of better words comment was a giant head-scratcher for the longest time. I had no idea why my code would work ‘sometimes’, and then other times just not work at all. I can credit ‘Team Autodesk’ for finally explaining why it wasn't working and then everything made sense.

If you have code that you have been trying to write for Inventor and InfraWorks… I would first recommend doing a find and replace on ThisApplication and switching it to ThisServer.

There's not a lot of magic to explain in this other than the fact that Inventor, when it's running in InfraWorks is Inventor server, not the application.

For some reason, you can run the server version in the Inventor application without issues, but you **cannot** run this application on the server version. It's messed up.

So if you're writing code that's going to eventually land in InfraWorks put ThisServer undo it, or just refer to ThisDoc.


```
Dim oPartDoc As PartDocument = ThisDoc.Document
Dim oAssyDoc As AssemblyDocument = ThisDoc.Document
Dim oDoc As Document = ThisDoc.Document

Dim oThisApplication As Document = ThisApplication.Documents.Item(1)
Dim oThisServer As Document = ThisServer.Documents.Item(1)
Dim oDoc As Document = ThisDoc.Document
```

How to setup a log File

The ability to create a log file external to Inventor is very useful; especially when trying to diagnose problems within an InfraWorks file.

Is it taking really long?
Is it showing up funny when you put it in the InfraWorks model?
What steps is it getting to before crashing and failing?

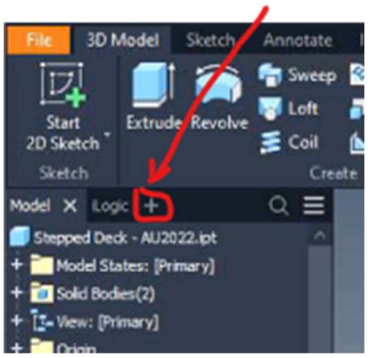
All of these questions and more can resolved by putting together a log file that you can then open and review.

Within Inventor (and within any code-editing tool worth it's salt), you can create logs that actively pop out data during the running of the code, but since you can't pull up the log screen from Inventor in InfraWorks, it doesn't help you out in that environment.

In Inventor, the log tools are:
Logger.Debug("")
Logger.Info("")
Or similar ones under the 'Logger Definition'.

The thing to watch for these is what level of debugging is turned on in your iLogic (drop down near the bottom of the coding window). If you select Debug, it will list everything; if you select Info, it will only show the Info ones (and ignore the debug ones).

To view these logs, navigate to the model feature tree and selet the [+] sign. One of the options in there is the iLogic Log window.



To create the external log file, you have to create a text file first and then write to that (and then close it when you're done.

The code to create that is:

```
Try
    oWrite = System.IO.File.AppendText("C:\Temp\~IFx Logger\Base Rule.txt")
Catch
    oWrite = System.IO.File.CreateText("C:\Temp\~IFx Logger\Base Rule.txt")
End Try

oWrite.WriteLine("Running Rule for Step #" & RuleArguments("StepCount") & " at: " & System.DateTime.Now)
oWrite.Close()
```

The reason for the 'try/catch' is that: **If you have already created the text file… then you're just going to append to it, not create a brand new one with the same name.** You might also have errors if you put the text file into a folder that hasn't been created yet. There's methods to create folders (obviously), but it's often easier to just stick the file into a location that exists on every windows computer (like C:\Temp). In the example in the image I put the file into a sub-folder, but I specifically created the folder before running my codes.

Once you have the external log file created, then you can put whatever you want in it. The 'oWrite.WriteLine("{stuff in here}") code will then dump whatever you want into there as the next line in the text file. If you put nothing in there (eg. ""), then it will just make a new blank line.

How you format your log file is more a matter of how nice you want to be to your future self or the next person who has to decipher what's going on. Starting your log entry with the 'System.DateTime.Now' is also a nice touch as it then puts a timestamp to when the code was triggered.

How to suppress things in parts and things in assemblies

This feature is helpful if you're going to have multiple parts, bodies, features in your models that you're toggling on and off based on parameters. As I mentioned before, you can turn off the features in multiple different ways and how it is smart to group some of them together when you do some of the suppressions.

This section is saying **what you should suppress** depending on what you want to do with it.

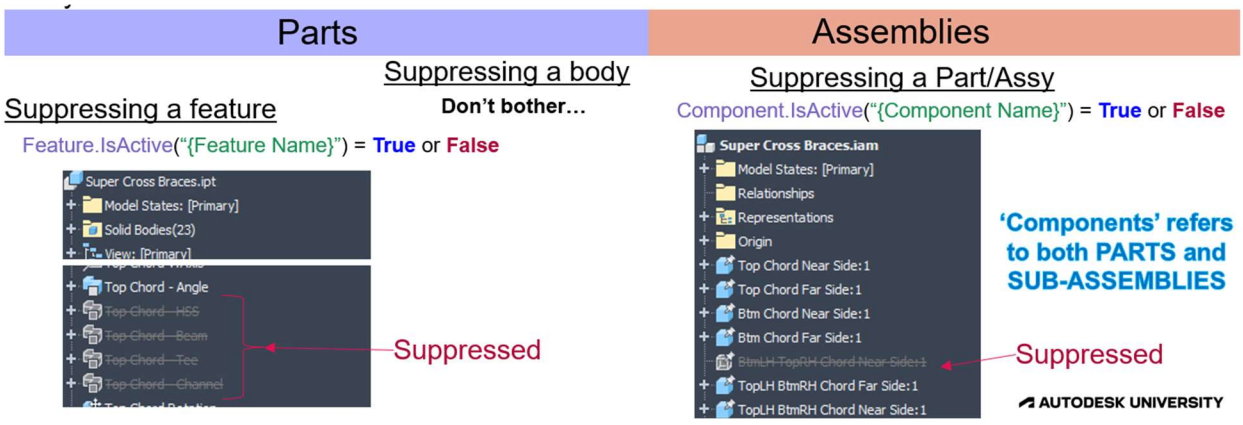
If you have an assembly and you're wanting to suppress the parts in the assembly then you should be using the **component is active command** at the assembly level.

In the example that I show below, what I've done is I have Top Chord – Angle and all the other versions of the Top Chord all as the same solid body. This means that while I'm turning on and off features on, I'm never actually turning off that body.

However in the assembly level, I've got two top chords and at this level I determine whether I'm going to show 1 chord, 2 chords or have both of them turned off.

To do this properly in any Inventor assembly model, it is likely going to be a mixture of both of these suppression techniques. It will get a bit confusing because technically you are suppressing the features from within the part file, and then the assemblies from within the assembly file; but **InfraWorks is really only going to see the assembly (or rather, the parameters in the primary file you give it)**, so you're controlling the part file features via the assembly parameters.

It's a bit of a head-scratcher to get this working the first time, but after enough trial and error it will become more clear.



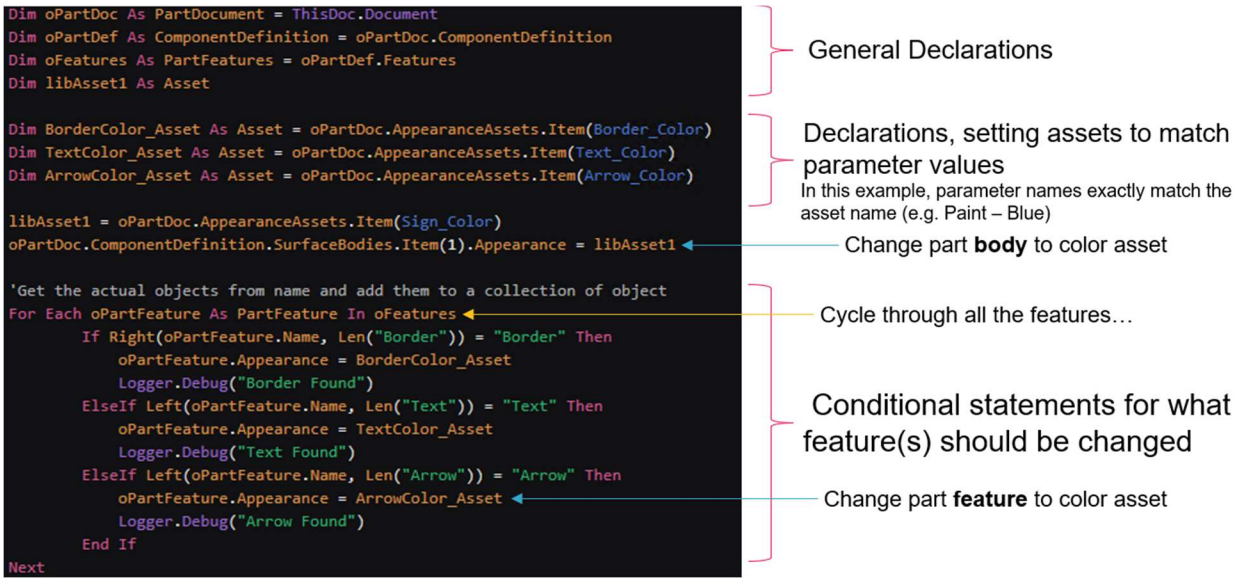
How to change part colours

This one's actually a bit more of a fun one, especially if you're someone like me who really wants your models to look as clean and accurate as possible. The ‘tricks’ to this rely more on having your appearance assets named correctly and having the match up to what you're describing in your code. I would recommend testing out simple things and seeing if the code will select your appearances correctly, and then move on to more difficult stuff.

The code to actually do the colour changes is a bit weird but it's all laid out below and really you just need to copy/paste it.

I don't list it out in the code, but whatever I would describe for ‘Arrow colour’ or ‘text colour’, the string of words in that parameter **exactly match how it is written in the appearance asset**.

I wouldn't recommend that the appearances have a lot of Textures in them (or really any textures) because it is doubtful that the mapping to those textures will carry through properly into InfraWorks. Maybe it will... but it's a gamble.



How to change text

This one I'll admit is a bit messed up, because apparently the way the text is written into Inventor uses a different language of code (possibly Javascript?) and so you will have to call out a different set of code from within your VBA based ilogic code. Fun Stuff!

The example that I showed below has technically 3x text boxes that I am using to better control whether things are on the first second or third line. I don't think you can control it well enough to say where the wrap should be on the text oh, so this is the solution that I came up with.

As I mentioned in the video (*Hint again) you can change a lot more than just the value of the text boxes, you can change the FONT TYPE, **the bold**, *the italics*, the colour, the size… I just didn't bother looking up what the entire list was. **In theory, it would be whatever you can edit from the text boxes within Inventor.**
I would recommend not trying to get too fancy with the specific style override, because sometimes it's just easier to do different feature depressions and manage the style changes that way.

Your call… I'm not your mom.

Key Parts of the Code:

```
Dim oPartDoc As PartDocument = ThisDoc.Document
Dim oPartDef As ComponentDefinition = oPartDoc.ComponentDefinition
Dim oFontSize As Long = Text_Line1Height/10
Dim Check As String
Dim x As Integer
```

General Declarations
Convert to **mm**; default iLogic is ALWAYS **cm**

```
For Each oSketch In oPartDef.Sketches
    Check = Left(oSketch.Name,Len("Text - "))
    Logger.Debug("TEST" & x & ": " & Check)
'   x = x + 1
    If Check = "Text - " Then
        Logger.Debug("Text Sketch Found")
        x = 0
        For Each oTextbox In oSketch.TextBoxes
            x = x + 1
            Select x
            Case 1
                oTextbox.FormattedText = "<StyleOverride FontSize = '" & Text_Line1Height/10 & "'>" & Text_FirstLine & "</StyleOverride>"
            Case 2
                oTextbox.FormattedText = "<StyleOverride FontSize = '" & Text_Line2Height/10 & "'>" & Text_SecondLine & "</StyleOverride>"
            Case 3
                oTextbox.FormattedText = "<StyleOverride FontSize = '" & Text_Line3Height/10 & "'>" & Text_ThirdLine & "</StyleOverride>"
            End Select
        Next
    End If

Next
```

In other terms:
oPartDef.Sketches.Sketch({number}).TextBoxes.Textbox({number}).FormattedText

Equals:
"<StyleOverride FontSize = '{font size value}'> {your text} </StyleOverride>"

How to setup drop down lists

Drop down list are… *loud sigh* something that I hope the InfraWorks team has resolved by the time you're reading this… but I also doubt they will; they've got other things that are higher priorities to be honest.

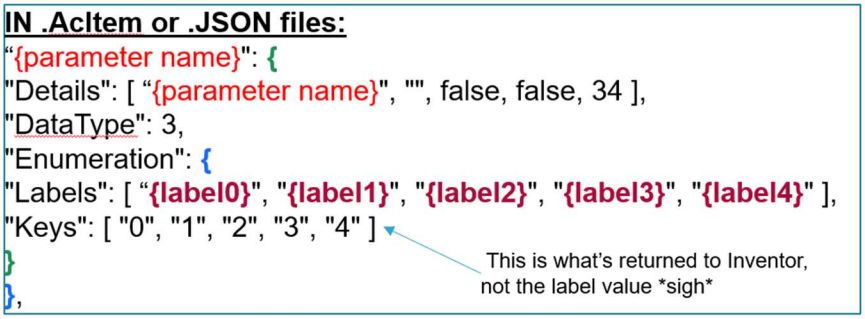
The obvious benefit of putting in drop-down lists is that you can then **control what parameters your users in InfraWorks can send to the Inventor models in the background**, but obviously it's a giant pain to have to create these drop-down lists structures manually. I'm not giving away the code here, but at Hatch we've totally automated this process (humble brag) because doing it manually over and over would just be awful.

The way you're going to be having these drop-down lists is to go and first publish a Json file, then open up the Json file and edit one of the enumeration sections to show that it has all of these values for the drop-down menu. Oh… and change it to data type 3 (which is the drop-down menu), but that's just the first step. *Meniacle Laugh*

If you do it that way, and then re-import your Json file, you will get the drop-down list in Infraworks **BUT** it will be **passing a bunch of numbers to your Inventor part/assembly** which you may or may not have planned for. It will also show up as a -1 on your InfraWorks UX screen unless you go and change the **default value** for that component to match one of the labels you have in your enumeration list.

Again, this is a giant pain because you might have a really nice drop-down list that you've created in Inventor and now… you have to manually one by one rewrite it out into a stupid text file so that it shows up properly in Infraworks.

I'm sure if enough people complain they will just finally finish this properly, but I guess we'll see won't we. :P



Note:
Some of the bridge structure models will be in GenericObjectStructure

Path: C:\Users\{username}\Documents\Autodesk InfraWorks Models\Autodesk 360\Sandbox\1252178\{Ifx Model Name}.files\unver\Content\Parts

Generated number for the Infraworks model

Infraworks version or Sandbox

By Pattern Function in Inventor

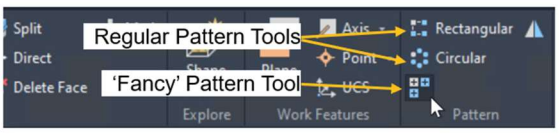
This one here is more of an Inventor feature, but it's a **freaking amazing one** and one that I wish I'd known about a lot earlier in my career.

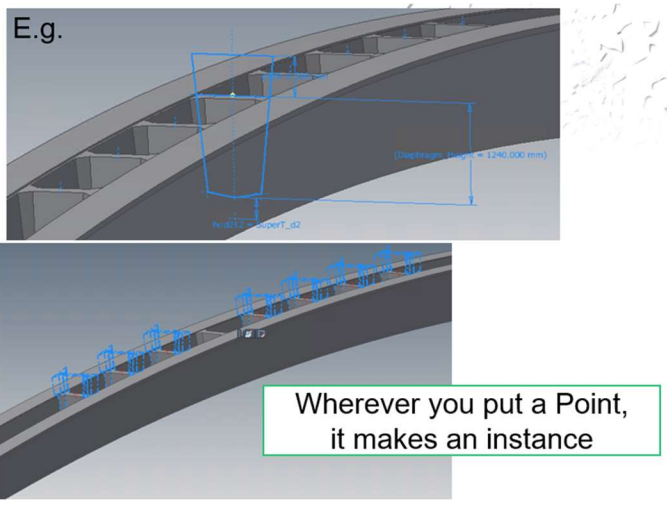
Essentially what it's doing is that I can have a solid or a feature (but not a part; This tool only works in part models by the way) and then I can have a different sketch with just has a bunch of work points in it. This tool will then take that feature and (using some datum point that you define) land an instance of that body or feature **wherever that point is**.

It's super useful because you can easily change the number of things in the pattern end control multiple different directions of where the pattern goes without having to create nested patterns and weird other things. Things can quickly get messy if you try to do this by having a pattern within a pattern within a pattern, because then you have to manage each pattern as a nested functionality of a different pattern. Blegh, might as well be using Revit…

The only downfall with this one is that you have to manually place those points using code; which is a little cumbersome sometimes. I don't mention it in the presentation (I think) but the data were those points of stored could easily be an external Excel file.

What I show in the code list of below, is the way to have your reference point sketch thing and essentially wipe the slate clean and then stick on a whole bunch of points wherever you want them. Another good feature of this patterning tool is I don't have to have any points in my sketch. It's going to do a pattern of absolutely nothingness… and it's okay with that! Other features would be squaking throwing up error messages like someone with their hair on fire.





Key sections of the code:

```
Dim oDoc As PartDocument = ThisDoc.Document
Dim oDef As PartComponentDefinition = oDoc.ComponentDefinition
Dim oTG As TransientGeometry = ThisServer.TransientGeometry
Dim oTO As TransientObjects = ThisServer.TransientObjects

Dim oPoint As SketchPoint
Dim NewPoint As Point2d

Dim NewPointY As Double
Dim i As Integer
Dim Interval As Integer
Dim Flip As Integer = 1

Dim PatternSketch As Sketch = oDef.Sketches.Item("Parabolic Diaphragm Pattern")
Dim PointCount As Integer

PointCount = PatternSketch.SketchPoints.Count
Logger.Info("Starting PointCount == " & PointCount)

Try
    For Each oPoint In PatternSketch.SketchPoints
        oPoint.Delete
    Next
Catch
End Try
```

Define Transients

Define Points

Point2d needs X & Y both defined

Delete all the old points

```
NewPointY = (Interval * Diaphragm_Spacing) * Flip + Diaphragm_CLOffset + Diaphragm_OR
Logger.Debug("NewPointY =" & Interval & "= " & NewPointY)
NewPoint = AddPoint(NewPointY)
If Not NewPoint.X = 50000 Then
PatternSketch.SketchPoints.Add(NewPoint)
End If
Flip  = -Flip
```

Add new points

```
Public Function AddPoint(Y_Coord As Double)

Dim oTG As TransientGeometry = ThisServer.TransientGeometry
Dim Coord(1) As Double
Dim Add_Point As Point2d

If Y_Coord > CurveEqn_StartDiaphragmX And Y_Coord < CurveEqn_EndDiaphragmX Then

Coord(0) = Y_Coord*(1/10)
Coord(1) = (-(Y_Coord^2)/((Half_Segment^2)/Hog))*(1/10)

Logger.Info("New Point Added at: " & Coord(0)*10 & "|Y & " & Coord(1)*10 & "|Z")

AddPoint = oTG.CreatePoint2d(Coord(0),Coord(1))

UsedDiaphragms = UsedDiaphragms + 1
    Logger.Debug("UsedDiaphragms =" & UsedDiaphragms)
Return AddPoint

Else
AddPoint = oTG.CreatePoint2d(50000,0)
Return AddPoint

End If
```

Function for how you want to add your points

Closing

Getting used to this kind of workflow from Inventor is not for the faint of heart, but if you can get it to work, **you can do some amazing things and create some amazing models in InfraWorks**.

Most people don't realize, when they look at InfraWorks, that **because it is connected to Inventor you can do anything Inventor can do**.

Since Inventor has integrated programming tool that is incredibly powerful…
…InfraWorks essentially has an integrated programming tool as well.

People can whine and complain that InfraWorks should use Revtit more and more so that they can use Dynamo, but it **already has an integrated (and better) tool** for doing most of the things that they would want to do in Dynamo.

When you're going to compare creating Revit families to what you can do in Inventor, **Inventor will crush Revit every single time**. Inventor is just more suited to that type of work in the first place, whereas Revit is focussed on a giant area/building models, with the Revit family builder as an afterthought it seems.

I'm sure as Revit Integrations to InfraWorks becomes available people will use those tools a lot, but I would be more excited to see better Inventor to Revit Integrations so that the InfraWorks models coming into Revit are as amazing as they are in Inventor.