IM473672

Modular Design, Design Standards, and Function Automation Using Inventor and iLogic

Demir Ali
Autodesk Ltd

Learning Objectives

- Learn some of the fundamentals for developing iLogic.
- Understand considerations for planning and building modular designs.
- Understand how to automate drawing annotations such as dimensions & balloons.
- See the benefit of utilising external data sources.
- Appreciate how automating certain functions improves product development times.

Description

This class will cover a broad range of elements that can be introduced by utilising the power of iLogic within Autodesk Inventor, from some of the fundamentals to how more-powerful functions can be applied.

Topics covered will include developing standards-based designs; capturing engineering knowledge; developing recipes from existing assemblies to help build new designs; creating drawings using the latest updates within Inventor; as well as retrieving data from other data sources in addition to automating functions within a model such as updating CAM toolpaths and FEA studies.

Speaker

Joined Autodesk in 2012 from Advanced Micro Devices (AMD), currently working as a Senior Technical Sales Specialist within the Design and Manufacturing team, covering the UK and Ireland. Over 20 years experience working with the Autodesk manufacturing portfolio, starting with AutoCAD R12 for DOS, then moving onto with Inventor when first released in 1999. Primarily now focused on the Product Design and Manufacturing Collection, Vault & VRED. When not at work, enjoys spending time sculling on the River Thames, mountain biking, getting out on the motorbike as well as a busy Dad of two.

Fundamentals

The gateway to automation needs to begin somewhere, with the start point often being the creation of intelligent models that will be used within the design and manufacturing process.

Over the years, organisations have been asking for a way to capture their design and engineering intent, taking labour intensive tasks out of their day to day activities by building engineering knowledge into their models. One benefit this brings, is a level of consistency and standardisation into the process.

In recent years, organisations have also been looking at how they can build a better up-front consumer experience. This usually develops into a requirement to provide a mechanism to make their design data available sooner, so they and their customers can take advantage of mass customisation workflows.

Last but not least, automation isn't purely down to design functions. Being able to benefit from any generated information downstream, such as business systems, is becoming a key requirement for most design and manufacturing companies.

INVENTOR

PARAMETRIC DESIGN

Build robust 3D models, utilising parameters, associative inputs, iParts, iAssemblies to define the basis of downstream design automation.

INVENTOR

DESIGN AUTOMATION

Define logical rules to control parts, assemblies and drawings. Enable faster design reuse, capture of engineering knowledge and enforce design standards. A foundation for consumption within Forge.

FORGE

ONLINE CONFIGURATIONS

Develop your own company specific configurator that consumes existing models and rules already embedded into your designs OR create a service to receive 3rd party requests to automate repetitive tasks.

PLM/ERP/MRP

BUSINESS SYSTEMS

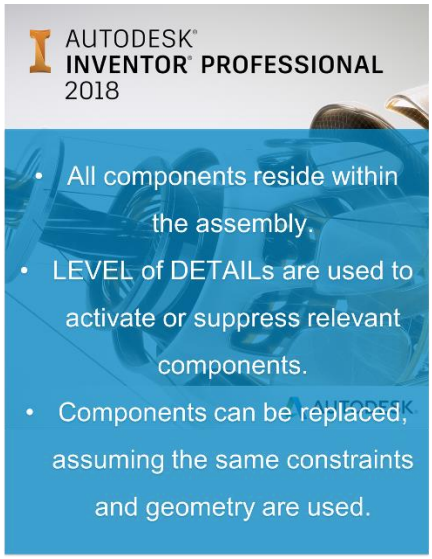
Connect relevant information to other systems, such as Bill of Material extraction, CPQ integration or other business related processes.

iLogic Improvements

Up to and including Inventor 2018, when creating an intelligent assembly, the user had to rely on creating all possible iterations within a single design. Effectively requiring the generation of a 150% model for each design type. By then suppressing or replacing parts, users were able to interactively drive assembly changes.

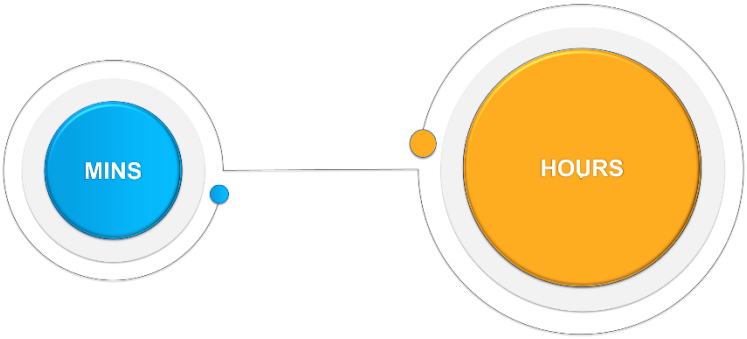
Within Inventor 2019, Autodesk introduced the capability to start capturing recipes for designs. This effectively allowed a user to start with a blank sheet each and every time to build up new assemblies.

Now with Inventor 2021, Autodesk have extended the capabilities within the drawing environment adding further flexibility enabling greater automation options, in addition to making the design automation within Inventor available on Autodesk Forge.



Time Saving

There are some quick wins, things that can make everyone's life easier by removing tedious tasks and saving minutes. Then there are the more complex solutions. Potentially taking longer to develop but ultimately saving hours of effort for every action.

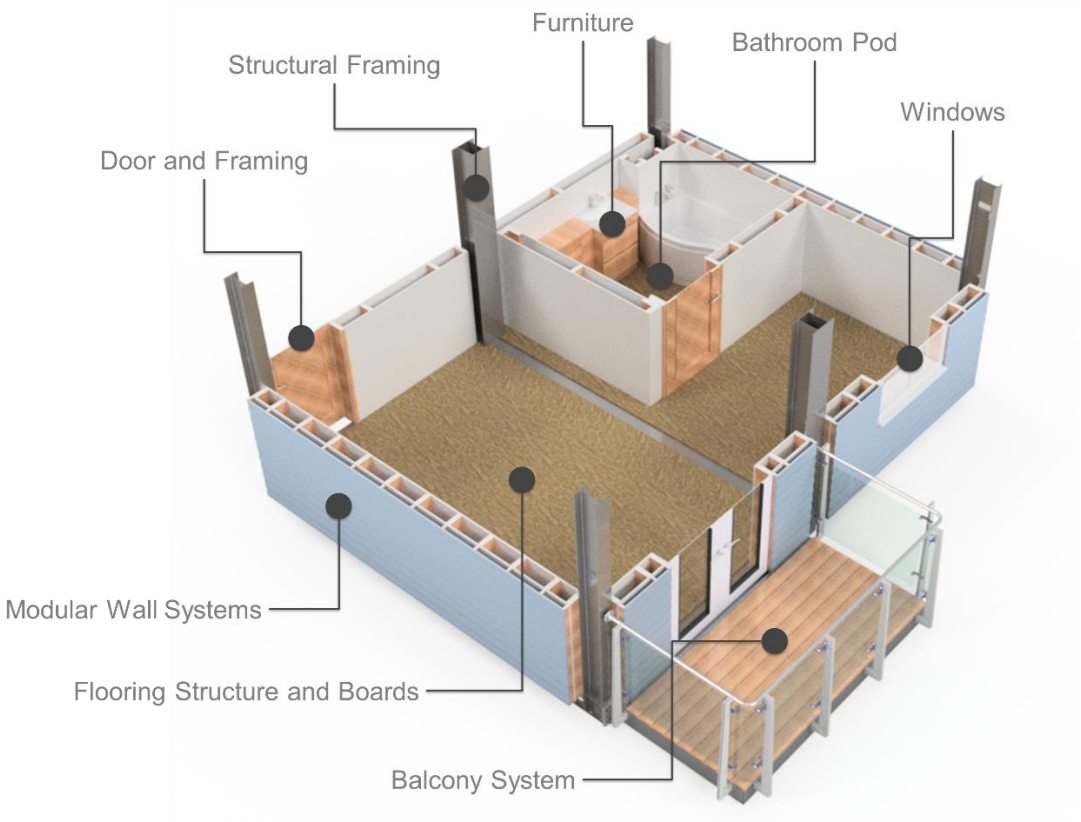


Modularisation

Let’s look at modular construction as an example of where design automation capabilities would benefit an organisation. Each aspect of the design needs to be positioned and sized to suit, allowing for enough variation to develop a unique outcome.

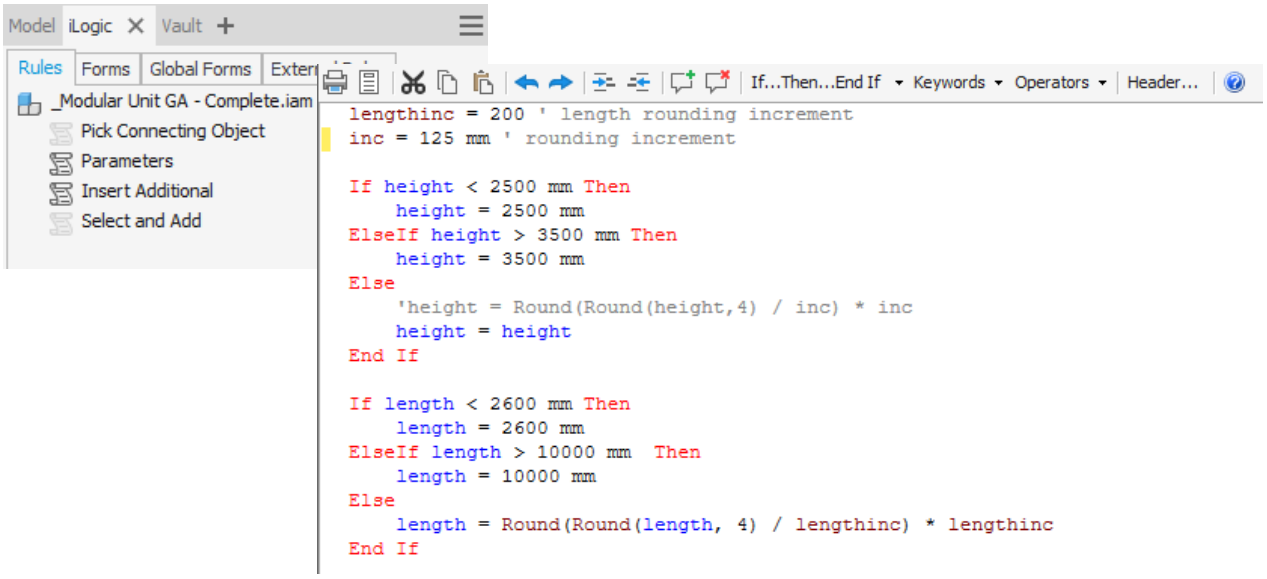
Every time a new design is required, rather than re-create and re-document for each variation, design automation allows for faster reuse of existing design data, the design knowledge to be maintained within the models and an overall increase in engineering productivity.

By capturing the knowledge and variation allowance, quicker interactions can be achieved whilst automating the documentation, and outputs also aid faster project turnarounds.



iLogic Rules

Within Inventor, iLogic uses visual basic as a foundation with predefined snippets that can be used to build rules. These rules can be stored locally within a model file or shared as a global rule that can be stored on a network or even compiled into a dynamic link library (via tools such as visual studio) to limit team members making changes.

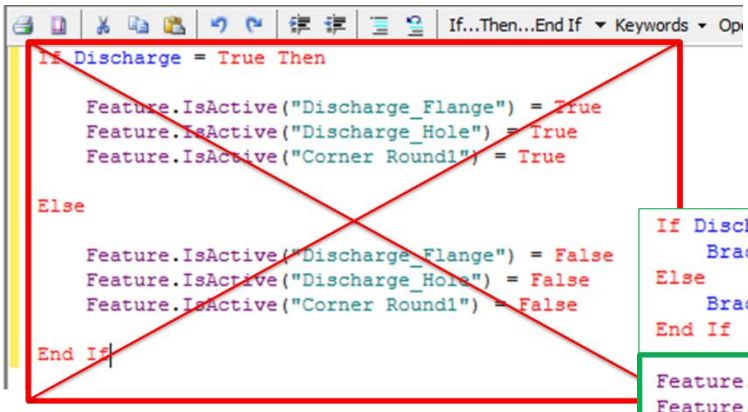


Logic Statements

The most common logic statement when generating rules is using the **If, ElseIf, Else, End If** rule. Most users will use this type of statement throughout their automation development.

One of the benefits of the **If, ElseIf, Else, End If** statement is, it can easily make allowances for outcomes that may be outside the expected result. This is why it is always important to finish a logic statement with an **Else** statement, as it always gives you an outcome no matter the result. Otherwise you may end up with errors that cause the rules to terminate prematurely.

It is also good working practice to avoid creating overcomplicated rules.



When relying on a Boolean parameter (True/False), it is always good practice if possible to directly reference it if needing to add / remove parts, suppress/unsuppress features.

```
If Discharge = True Then
    Bracing_Adjust = 1 ul
Else
    Bracing_Adjust = 0 ul
End If

Feature.IsActive("Discharge_Flange") = Discharge
Feature.IsActive("Discharge_Hole") = Discharge
Feature.IsActive("Corner Round1") = Discharge
```

It makes reading the code easier and removes unnecessary lines within a rule.

Other ways of accommodating logic is to use a **Select Case, Case, End Select** statement. This allows for easier interpretation of logic that have specific defined values and variation.

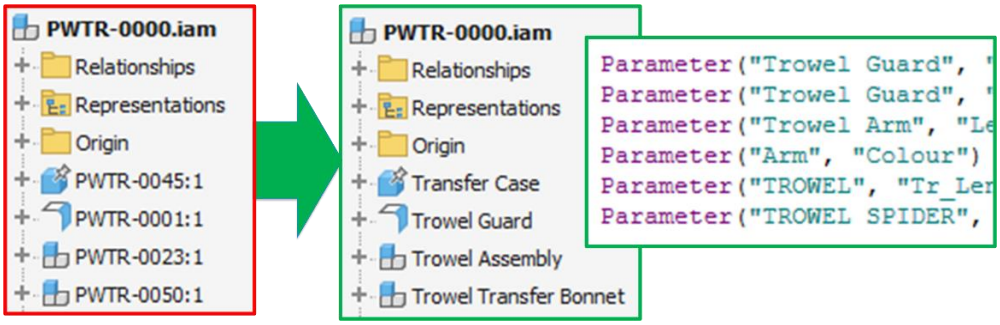
Rather than having an overcomplicated If, ElseIf, Else covering expected and intermediate values, the **Case** definition allows easier to understand sets ot values ot be defined.

```
Select Case Diameter
Case 775 mm
  arm_length = 225 mm
  trowel_length = 300 mm
Case 875 mm
  arm_length = 250 mm
  trowel_length = 350 mm
Case 975 mm
  arm_length = 275 mm
  trowel_length = 375 mm
Case 1075 mm
  arm_length = 300 mm
  trowel_length = 400 mm
End Select
```

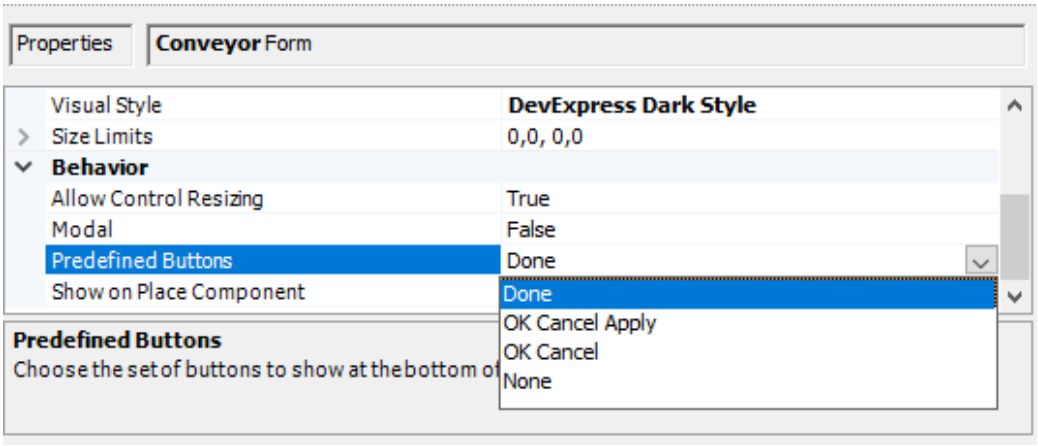
Best Practices

One of the main aspects to pay attention to is the instance names within Inventors browser. When capturing iLogic, Inventor by default, uses the instance name to define the reference within iLogic, for example to find and pass parameters.

By overriding the Inventor model instance name, not matter what happens to a design, whether files are renamed or copied, the iLogic rule will still be able to find it's associated model reference and run.



Also bear in mind, Visual Basic (which iLogic is based upon) and Inventors parameters are case sensitive, so you need to make sure there is consistency when defining and referencing any naming conventions.



Although some power users may take advantage of Visual Studio to create custom complex forms, there is much that can be done out of the box using Inventors embedded form creation capability.

However it is important to understand how the forms predefined button options will affect updates, after parameters are changed.

Leaving an iLogic form as the default means for every parameter change, the model will update, which can become time consuming when working with a large updates.

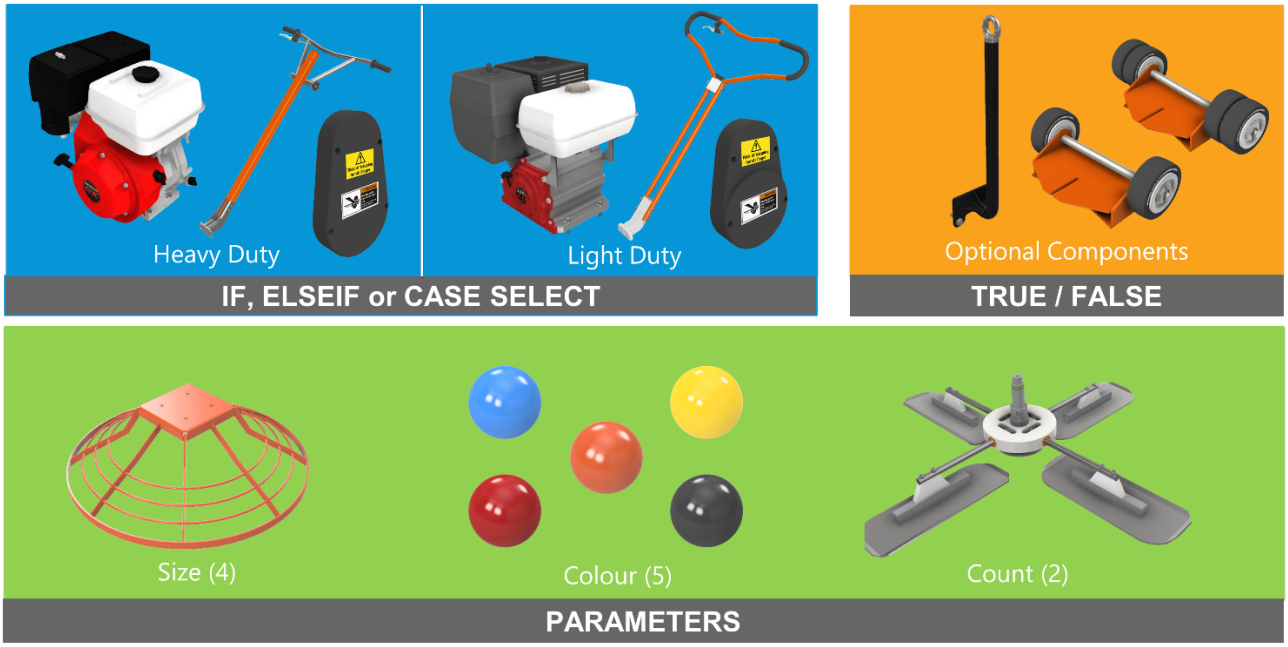
If it makes sense for a user to input all their options before a model update occurs, make sure you change the predefined button behaviour to allow for an OK or Apply.

Automation Planning

Before you jump into creating any code, it's always good practice to plan out what the expected outcome should be and how you think you're going to get there.

Breaking down the design elements, understanding their complexity, the parameters and rules required, make it much easier to develop configurations using a more methodical workflow.

Taking a look at the example here, when seperating the changes and what's staying the same, you get a much better feeling of the controls that need to be in place, as well as the core 'data management' requirements for the model.



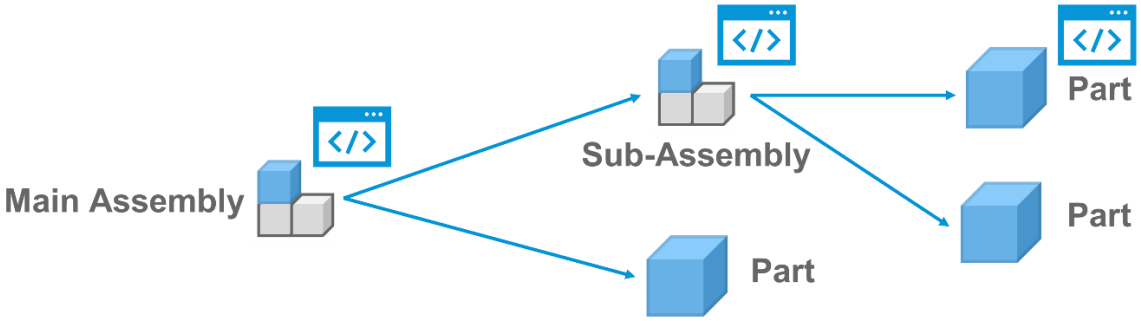
Highlighting the different changes within the design also give you a better understanding of the potential benefits design automation will deliver for your project.

Where to place your iLogic rules

When defining iLogic rules, it's important to understand where these should be created and stored. Users can create rules that are embedded into files or stored externally within a common shared location.

If rules are being embedded into models, there are additional considerations such as:

- What rules need to be defined within the top level assembly?
- What rules should exist within the sub-components?
- Are there any external references that needs to be considered, such as an Excel lookup and what direction will the information flow?



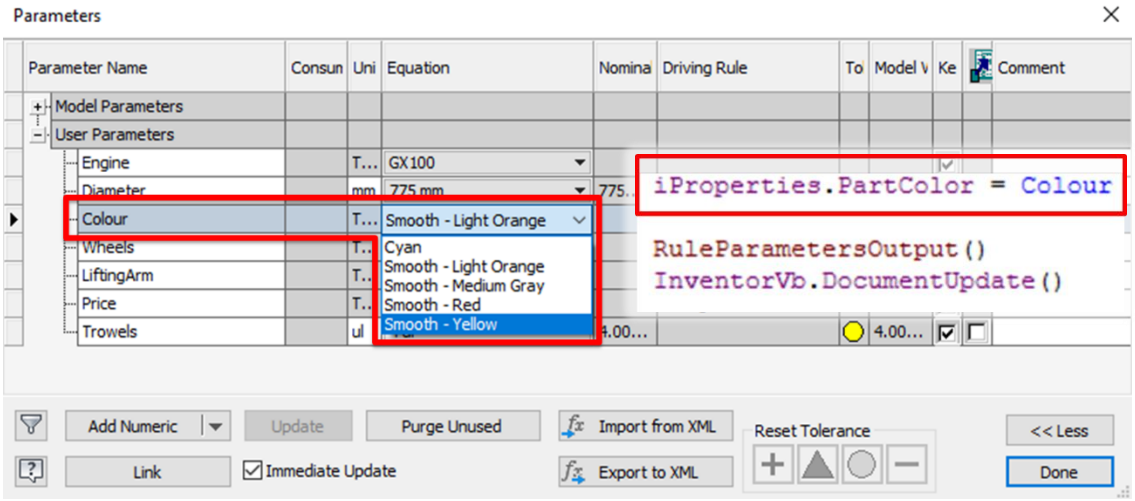
Sometimes, even when most of the parameters or options can be driven at the top level, it doesn't mean they should be.
There may be a reason to create rules further down in the model structure, for either simplification, model sharing or because some should be run at the part level. Rules placed at the part level are often there to make sure anyone using that part has access to the relevant logic as well as controls, such as material and appearance, that need to be driven from the part level.

Part appearance and material changes are often something updated. Within an assembly environment you can override a parts appearance, but remember, this is only applicable to the design view that is current at the time. This maybe as required because the component may be painted once assembled, so has no effect of the manufacturing detail of individual parts.

However, you might want to change the part appearance itself.

By defining a text parameter within the assembly that references a valid appearance and a rule to pass this to the part file, a part level rule rule can be created to update the part appearance relying on the part level parameter.

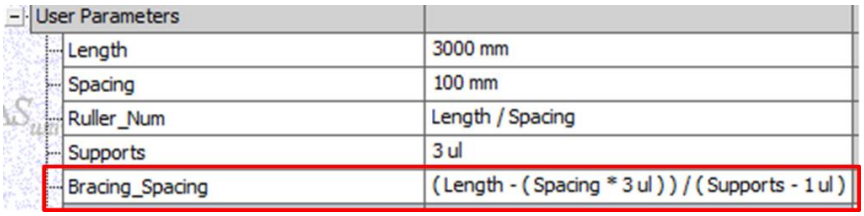
To start with, the part will only need a equivalent text parameter using a valid appearance name, even though it may be eventually overwritten just to ensure no error message is displayed on creation.



Equations

Just because you can, doesn't always mean you should. There are still lots of Inventor functions that can be utilised outside of iLogic. For example when generating parameter based equations.

Although you can build equations linked to parameters within iLogic rules, remember there is the native Inventor parameter dialog box. To understand how a model have been generated, people will often look first at the model parameters to try and understand the relationships between values. If this is hidden away within an iLogic rule it may be more difficult to appreciate and understand.

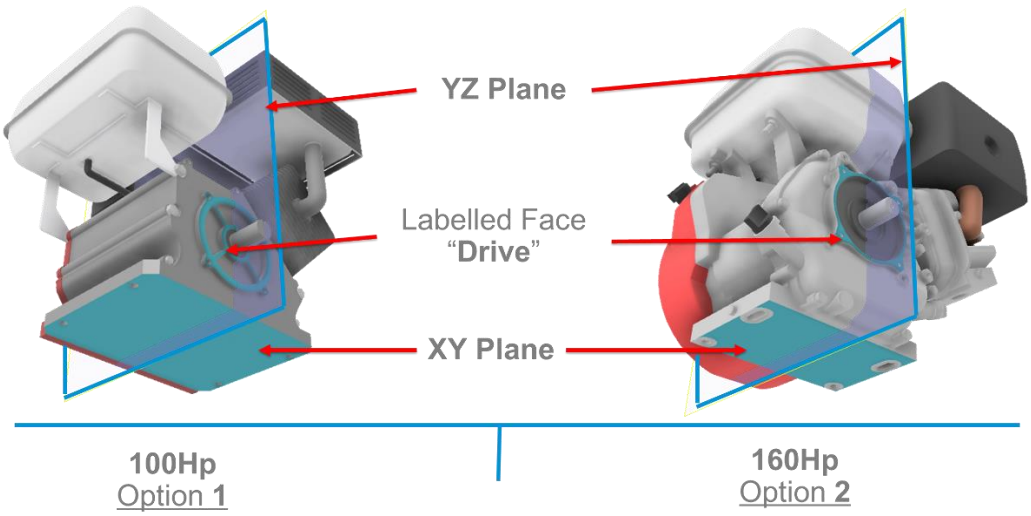


Building Assemblies

Defining Positions

Being able to build assemblies through code relies on knowing what parameters and positions are available, in addition to using standard naming conventions as well as appreciating how an assembly could be built.

For example, when assembling components, we can position them using fixed co-ordinates or with constraints relying on Faces, Edges or Vertex's, and Work Features.
We also now have the flexibility to create named geometry that can be referenced when defining any component position.

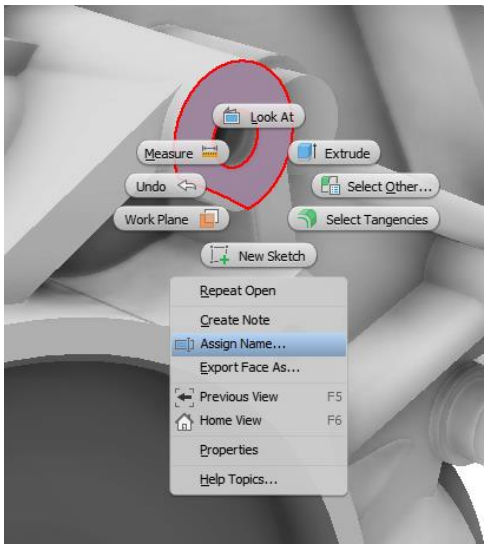


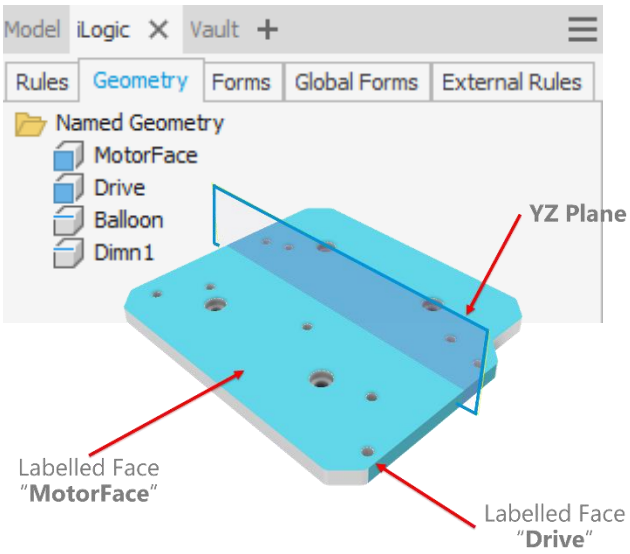
Geometry Labels

When creating a configuration that is switching out components, if constraining using default work features, as long as the normals are all in the right orientation, nothing more needs to setup or created.

However if constraints are reliant on Faces, Edges and Vertex's, users will want to ensure the same positions exist within all related components. This is were iLogic Geometry Labels can now assist users, not just with assembly but also drawing annotation automation.

Geometry Labels can be either manually created within Parts, or a user can rely on the default naming conventions when capturing the code within iLogic.



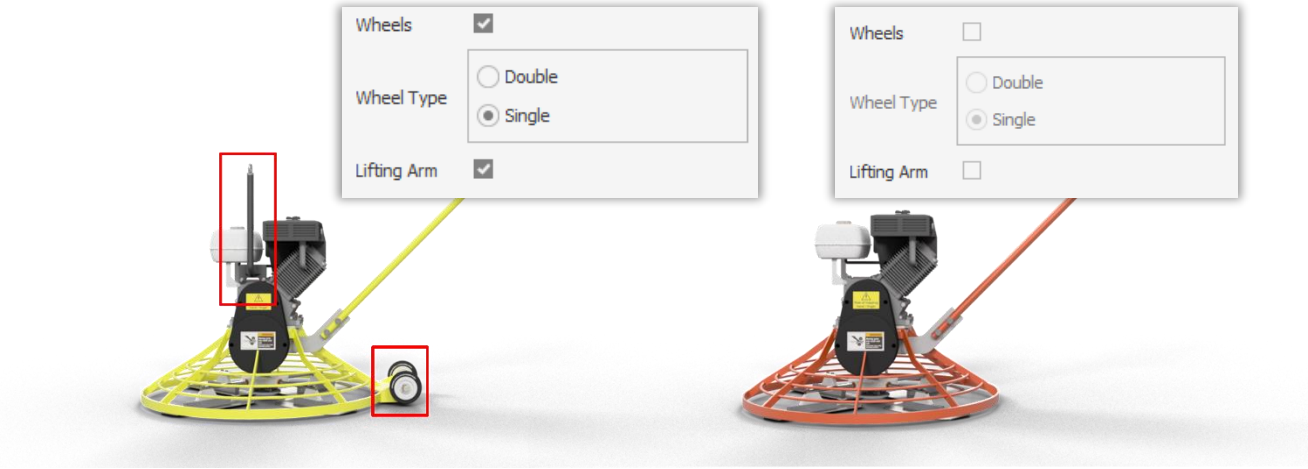


If an assembly has different options, with all choices being mated in a similar manner, we can use a combination of methods to fix their relative position.

TIP: When switching between options in the same position, to make things simpler, always try to make sure each Geometry Label or Work Feature has an identical name to make life easier.

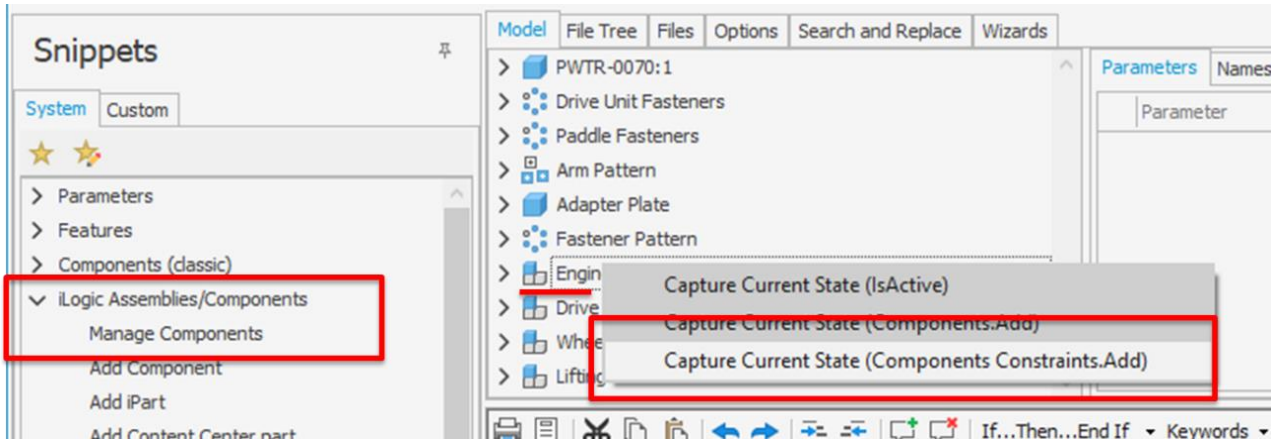
Boolean Parameters

When components are to be added or deleted, all that is required is a simple Boolean parameter. If the parameter is set to True, the components will be added, when False the related components will be deleted from the assembly (not suppressed).



Managing Adding / Removal of Components

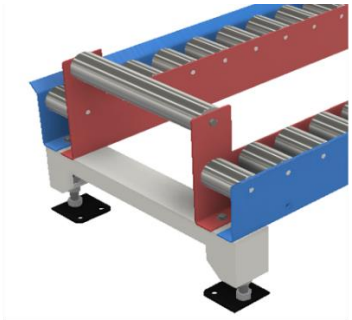
To define an assembly configuration, a user only needs to capture the snippet of the components wishing to be controlled. The captured code can be enhanced using the available snippets that includes the 'Manage Components' snippet.



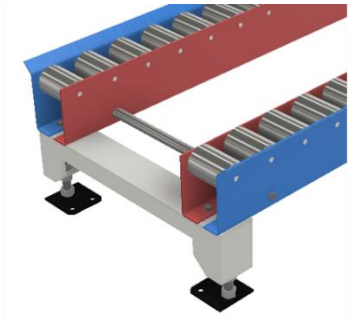
```
If Wheeltype = "Single" Then
   WheelName = "PWTR-0136.iam"
Else
   WheelName = "PWTR-0157.iam"
End If
ThisAssembly.BeginManage("AddWheels")
If (Wheels) 'One logic statement to add or delete component

Dim Wheel_Assembly = Components.Add("Wheel Assembly", WheelName)
Constraints.AddFlush("Wheel01", "Wheel Assembly", "YZ Plane","Trowel Guard", "XZ Plane")
Constraints.AddMate("Wheel02", "Wheel Assembly", "XZ Plane","Trowel Guard", "YZ Plane")
Constraints.AddFlush("Wheel03", "Wheel Assembly", "XY Plane", "Trowel Guard", "XY Plane")

Parameter("Wheel Bracket", "Diameter") = Diameter 'Parameter only written back if True
Parameter("Wheel Bracket", "Colour") = Colour Parameter only written back if True

End If
ThisAssembly.EndManage("AddWheels")
```

Bill of Materials

When automating assemblies, you still still need to consider what method works best for your situation. Depending on the type of configuration being applied, do you really need to switch out parts?



Configuration 1



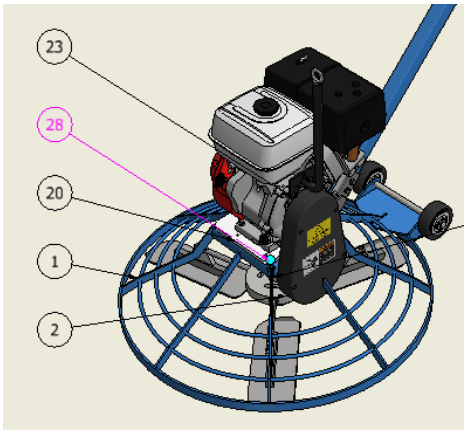
Configuration 2

In this example, Configuration #1 will have two unique parts (in red), the left and right hand. The bill of materials listing two different part numbers.

Configuration # 2 could be defined by deleting and adding new components (or replacing), however if managing separate files isn't a concern, another option may be to leave the model referencing two unique files.
An iLogic rule could then be defined to suppress the relevant features and override the part number to match. Although we will still have 2 different part files, the bill of materials will show a new part number with a quantity of two.

Just because we can utilise methods to remove, switch out and add components, again depending on the situation, it doesn't always mean we have too.

Drawing Automation



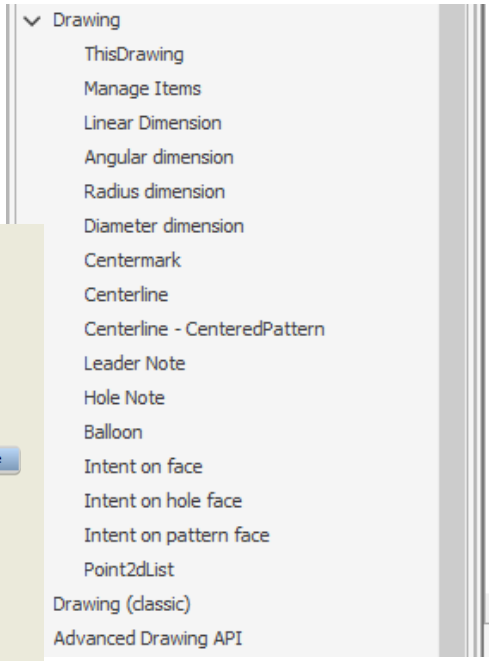
In previous Inventor releases, users would have had to rely heavily on the API to add any level of automation to their drawings. If a design had different components being added and removed, then dimensions would sometimes need to exist and other times not, resulting in considerationis on how to best to manage orphaned annotations.

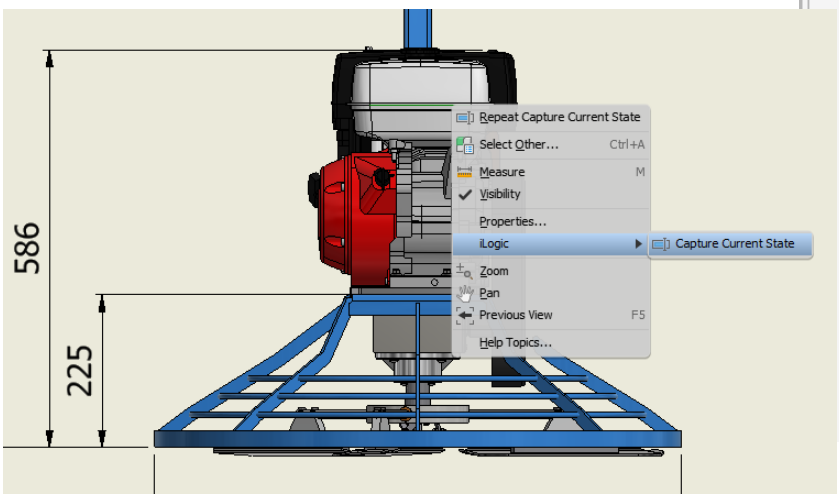
What's New

Now with Inventor 2021, users will see new iLogic snippets that can be used to create annotation such as: Hole Notes, Balloons, Centrelines and various Dimension types.

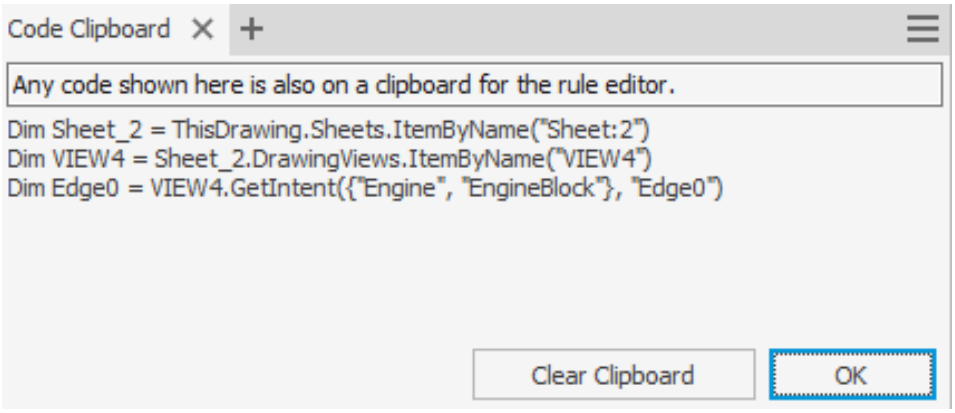
Annotations

As long as the model edges are visible within the drawing view, users can select and capture the current state. This places a code snippet into the new iLogic clipboard, whilst users can continue adding references until they are ready to build out the function they require.

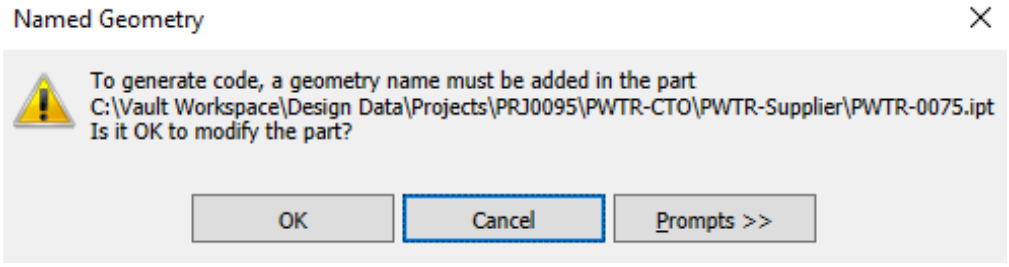




Ideally geometry should already be labelled within the modelling environment as it means the code can be standardised to allows look for certain named geometry.
TIP: Remember labelled geometry is applied to only parts.



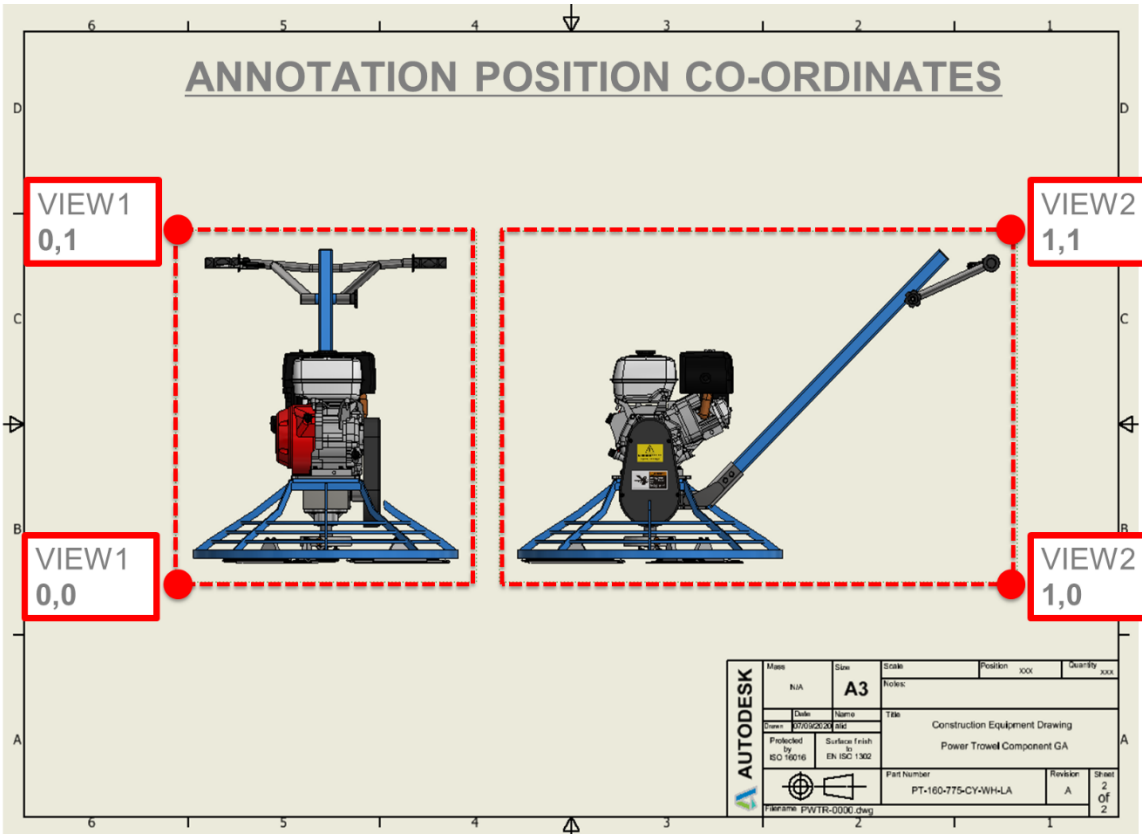
If edges aren't labelled, Inventor will start giving them generic identifications, whilst prompting the user that these parts will be modified and a subsequent label created.



Capturing of the code does a lot of the hard work for you, but when placing annotations, five things need to be specified to fully define it:

1. The sheet that will be used.
2. The view to be annotated.
3. Annotation position co-ordinate related to view extents (X,Y).
4. Geometry and name to annotate (may require one or two geometry names)
5. Annotation type with a unique name, position and geometry.

```
1. Dim oName = ActiveSheet.Name
   Dim oSheet = ThisDrawing.Sheets.ItemByName(oName)
2. Dim oView1 = oSheet.DrawingViews.ItemByName("View1")
3. Dim ptCyl1 = oView1.SheetPoint(0.0,0.5)
4. Dim geomInt1 = oView1.GetIntent({"Engine", "EngineBlock"}, "Balloon")
5. Dim balloon1 = oSheet.Balloons.Add("Balloon 1", {ptCyl1 }, geomInt1)
```

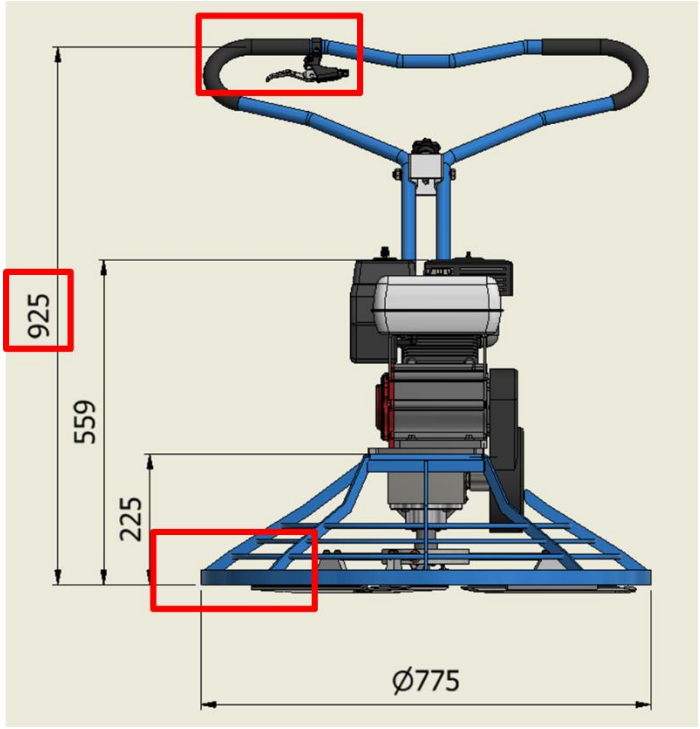


Anyone that has placed a dimension on a drawing will be aware there are multiple context sensitive methods that can control it's type and display options. When generating dimensions, the geometry you reference will dictate the how it is displayed by default.

For example if you dimension using a single selection of the base of a cylinder. The default dimension will be a diametric dimension. If you were to define two parallel edges of a cylinder, you will create a normal aligned dimension. That's not to say you need to be an expert in the Inventor API. The tooltips you see within the rule creation environment will highlight the controls for what is possible.

```
'Dimension Engine Extents
Dim namedGeo3 = oView1.GetIntent({"Engine", "EngineBlock"}, "Dimn1")
Dim linDim2 = genDims.AddLinear("Dimension2", oView1.SheetPoint(-0.2, 0.5), namedGeo1, namedGeo3, )
linDim2.NativeEntity.P

'Dimension Handle
Try
    Dim handleCentre1
    Dim linDim3 = genD
    linDim3.NativeEnti
```

```
Function IManagedGeneralDimensions.AddLinear(name As String, textOrigin As LinearDimensionTextPositionSpec, intentOne As
GeometryIntent, Optional intentTwo As GeometryIntent, Optional dimensionType As DimensionTypeEnum, Optional alignmentGeometry As Object,
Optional arrowHeadsInside As Boolean, Optional dimensionStyle As DimensionStyle, Optional layer As Layer) As
IManagedLinearGeneralDimension
Adds or edits a linear dimension. Valid intent combinations are: Two points, Two curves, Point and a curve, One linear curve, One arc
curve (with DimensionType set to kAlignedDimensionType for chord length and kArcLengthDimensionType for arc length).
dimensionType: Specifies the linear dimension type. Valid values (based on the input intents) are kAlignedDimensionType,
kHorizontalDimensionType, kVerticalDimensionType, kArcLengthDimensionType, kSymmetricDimensionType and kDiametricDimensionType. If not
specified, the argument defaults to kAlignedDimensionType. An error will occur if the specified type is invalid for the input intents.
For instance, kHorizontalDimensionType is invalid for a vertical dimension and kSymmetricDimensionType and kDiametricDimensionType are
invalid if only the first intent (an edge) is specified. kArcLengthDimensionType is only valid if two intents are supplied and they
represent end points of an arc or a single intent is supplied and it represents an arc.
```

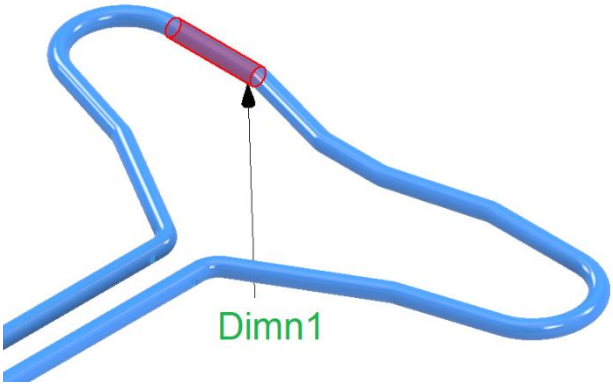


Looking at this example, you can see we want to create a dimension (925) from the base edge of the lowest part to the centreline of the handle.

This is a dimension that goes from one part edge to the centre of another.

The geometry selection will be relatively easy to define at the base, as it's a visible edge. However for the handle centre, it's a little more complicated.

The dimension will need to be based upon the handle centre.



In this scenario, to specify the centre of the handle, we can edit part of the snippet that would be used to create a centreline. The centreline would be generated by using the named geometry already within the model that references face of the handle tube.

As this example model will have different handles with different model names due to the configuration setup, we need to create a rule that will try to find the first handle to annotate and if unsuccessful fall back to another option. Below is an example of what this would look like.

```
Dim oName = ActiveSheet.Name
Dim oSheet = ThisDrawing.Sheets.ItemByName(oName)
Dim genDims = oSheet.DrawingDimensions.GeneralDimensions
Dim oView1 = oSheet.DrawingViews.ItemByName("VIEW4")

Dim namedGeo1 = oView1.GetIntent("Trowel Guard", "Dimn1")

Try
  Dim handleCentre1 = oView1.GetIntent({"ArmAssem", "Arm"}, "Dimn1", PointIntentEnum.kCenterPointIntent)
  Dim linDim3 = genDims.AddLinear("Dimension3", oView1.SheetPoint(-0.3, 0.5), namedGeo1, handleCentre1, kVerticalDimensionType)
  linDim3.NativeEntity.Precision = 0
Catch
  Dim handleCentre1 = oView1.GetIntent({"ArmAssem", "Bars"}, "Dimn1", PointIntentEnum.kCenterPointIntent)
  Dim linDim3 = genDims.AddLinear("Dimension3", oView1.SheetPoint(-0.3, 0.5), namedGeo1, handleCentre1, kVerticalDimensionType)
  linDim3.NativeEntity.Precision = 0
End Try
```

Baseline Bottom edge.

Try, Catch, End Try allows for an alternate model configuration to be dimensioned without an error occurring.

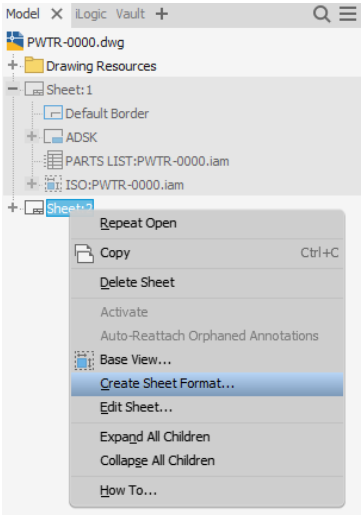
Only create a vertical dimension

Overrides precision to zero decimal places.

Specifies the centre of the selected geometry.

Drawing Sheet Formats

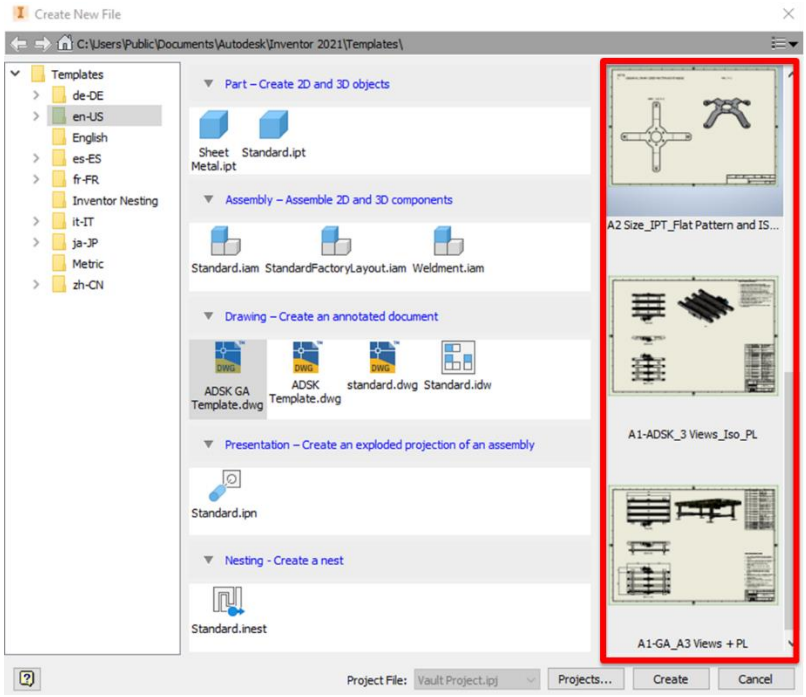
Within Inventor 2021, the saved sheet format capability has been enhanced with more annotation types being captured when generating a sheet format.



Sheet Formats within Inventor now can support:

- Text
- Parts Lists
- Create View from Model
- Flat Pattern Views
- View Settings
- Source Component
- Fit Views to Sheet

Not only does it make standardising view creation easier but when combined with iLogic, it can help automate the vast majority of functions required to create new drawings, views and their associated annotation.



When wishing to create a new drawing, users now get presented with an option to pick the sheet format they wish to use.

In combination with iLogic triggers within the saved template, this action could then run the relevant rules required to dimension, balloon and annotate the drawing.

External Lookup

There are many reasons why using external lookup tables can be of benefit.

They are often used to control anything, from complex parameter tables to referencing associated information that may not be relevant to store within the CAD model itself.

By using an external lookup, the information can be kept separate and up to date.

	A	B	C	D	E
1	ND	body_length	npt_depth	recess_length	npt_dia
2	1/4 in	1.75 in	.462 in	0.088 in	.4375 i
3	3/8 in	1.75 in	0.445 in	0.088 in	0.5625
4	1/2 in	1.975 in	0.485 in	0.088 in	0.7031
5	3/4 in	2.25 in	.525 in	.116 in	.9062 i
6	1 in	2.9 in	.685 in	.213 in	1.141 i
7	1 1/2 in	3.625 in	.72 in	.24 in	1.734 i
8					

A1

	A	B
1	ADSK-MH-10003	30
2	ADSK-MH-10007	30
3	ADSK-MH-10009	30
4	ADSK-MH-10011	8
5	ADSK-MH-10013	50
6	ADSK-MH-10015	20
7	ADSK-MH-10016	10
8	ADSK-MH-10018	10
9	ADSK-MH-10022	50
10	ADSK-MH-10025	3

One such use case is, if there is a requirement to reference data that resides within other business systems, such as an ERP platform.

A design engineer may need to start looking at a designs overall cost OR start undertaking a value engineering activity.

CAD users would rarely be encouraged to start embedding pricing into their CAD models, however aslong as the source data can be accessed, for example, part numbers and current prices can be extracted to Excel or a text based format such as comma delimited file (CSV), then we can reference that information within Inventor when necessary.

Referencing the Bill of Materials

Planning is still a key aspect of using this type of capability, because you need to think how a price can be calculated ideally by using the Bill of Materials (BOM).

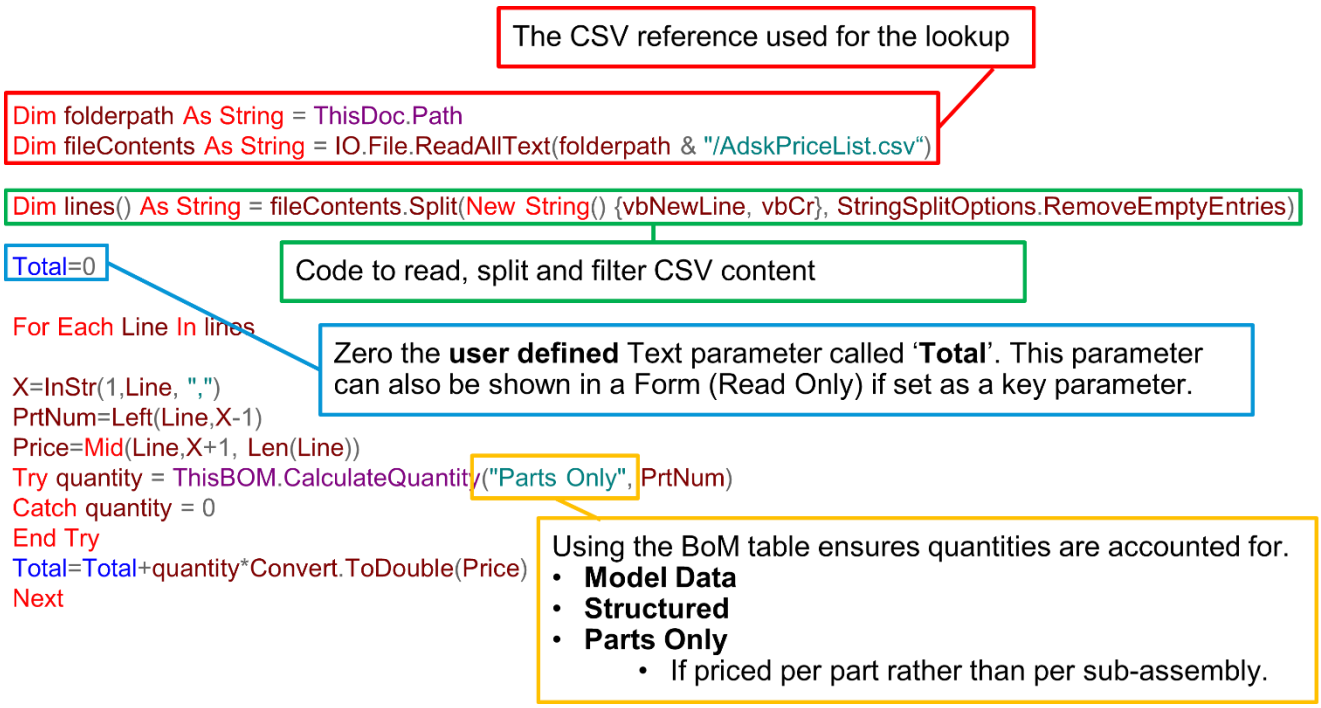
Should we be referencing the Structured BOM, and to what depth, OR calculate everything at a Parts Only level. Or should there be some type of logic to define when to lookup a part or an assembly reference, such as relying on it's BOM Structure.

Next is a sample iLogic snippet that references a comma delimited file.

Highlighted are the different elements that are required to reference an external data source and run through an entire assembly.

- Initially we are referencing the data source, in this case a comma delimited file.
- Next we break down the data, so each column can be read.
- By using a Text Parameter called **Total**, we then set any existing total to zero.

- Next a loop runs through each line of the file, finding the part number and price multiplied by the quantity within the assembly ‘Parts Only’ bill of materials.
- The bill of materials control will dictate how the quantification and lookup with be calculated.
- Each time a loop is run, the total will be added to the last known value withing the cycle.

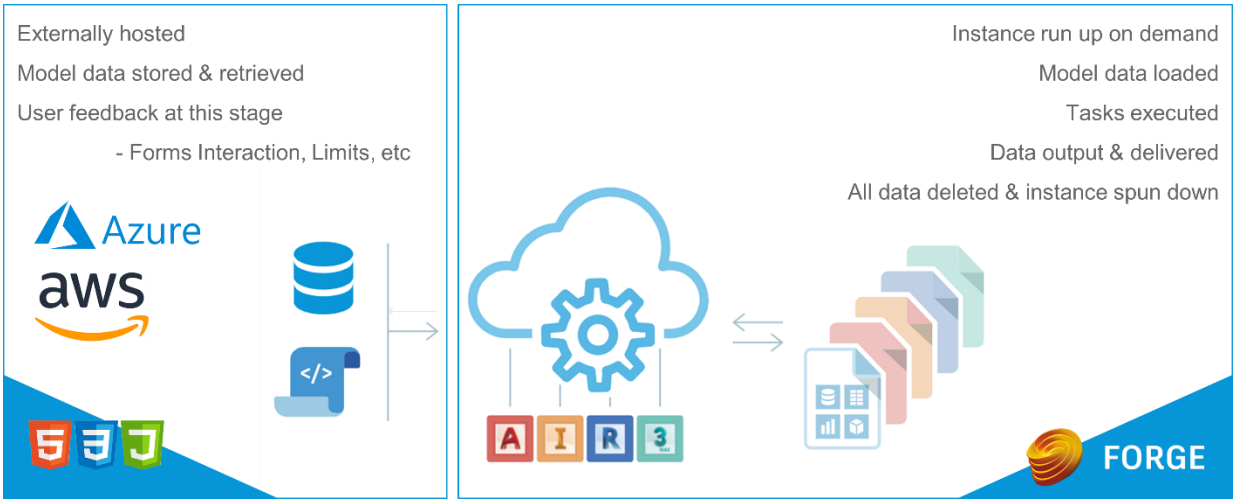


Autodesk Forge

Up to this point, we have worked through the key elements required to automate assembly creation, generate drawings and even connect to a separate data source to accommodate aspects such as pricing.

For certain types of products, especially if they are configured to order, most companies are looking at ways that allow their sales teams and end users the ability to mass customise the design, without relying on experienced engineers time and resource to undertake this repetivite work. The ideal scenario allowing for design engineers to spend more time on value add activities that their experience and knowledge is required to deliver.

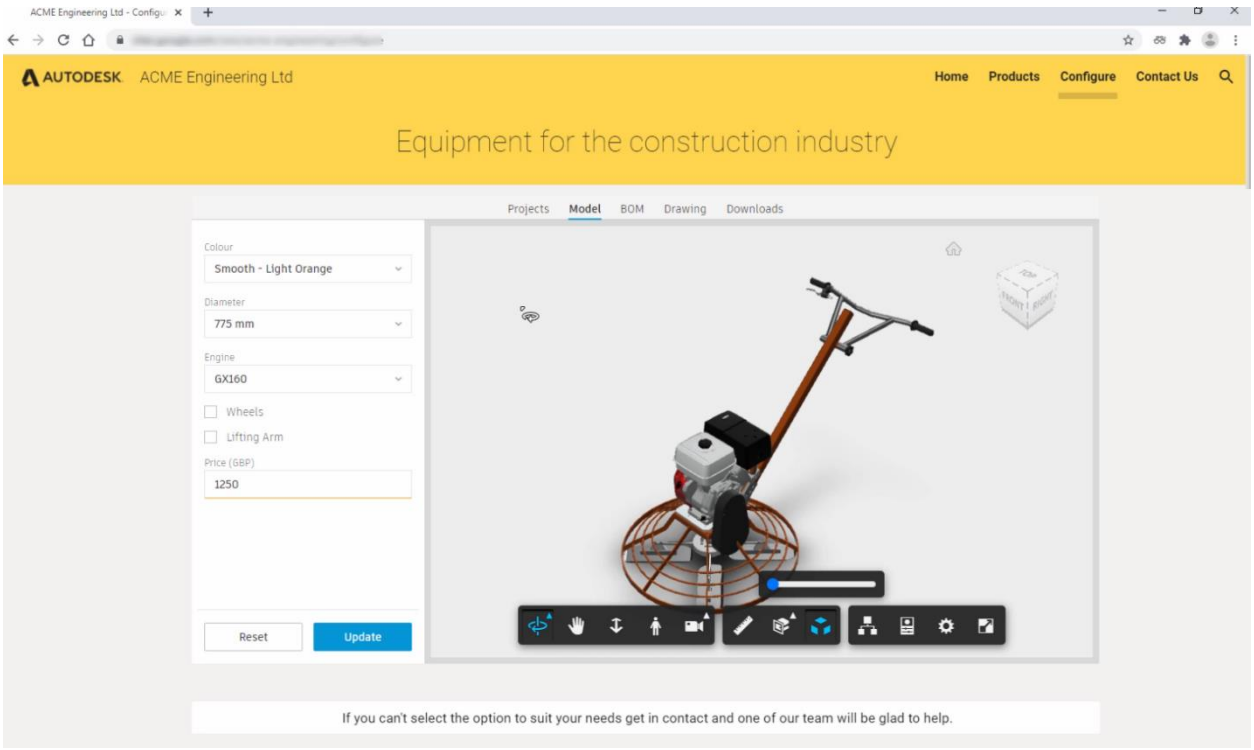
By utilsing Autodesk Forge and it’s Inventor Design Automation service, customers now have the capability to make their configurable design available to their customers or sales engineers much sooner in the buying cycle.



Companies can now develop and host their own cloud environment, that can consume their intelligent design data, connecting to existing sales tools, business platforms or to just visualise what the product will look like by using an interactive configuration environment.

Sample Application

Using the core capabilities of Inventor, design automation on the cloud is now a reality.



However you're not expected to have to come up with something from scratch, using the source code available free of charge on Github and if you have the web development skills, you can take the reference app and edit it to suit your own requirements.

Source Code

Below is the link to the sample application on GitHub:
https://github.com/Developer-Autodesk/forge-configurator-inventor

You will need to host the application on your own web service, however the sample app demonstrates how you can view and interact with Inventor models, views its Bill of Materials, drawings and download files such as the assembly, drawing, PDF and a simple Revit family file.

Automating Functions

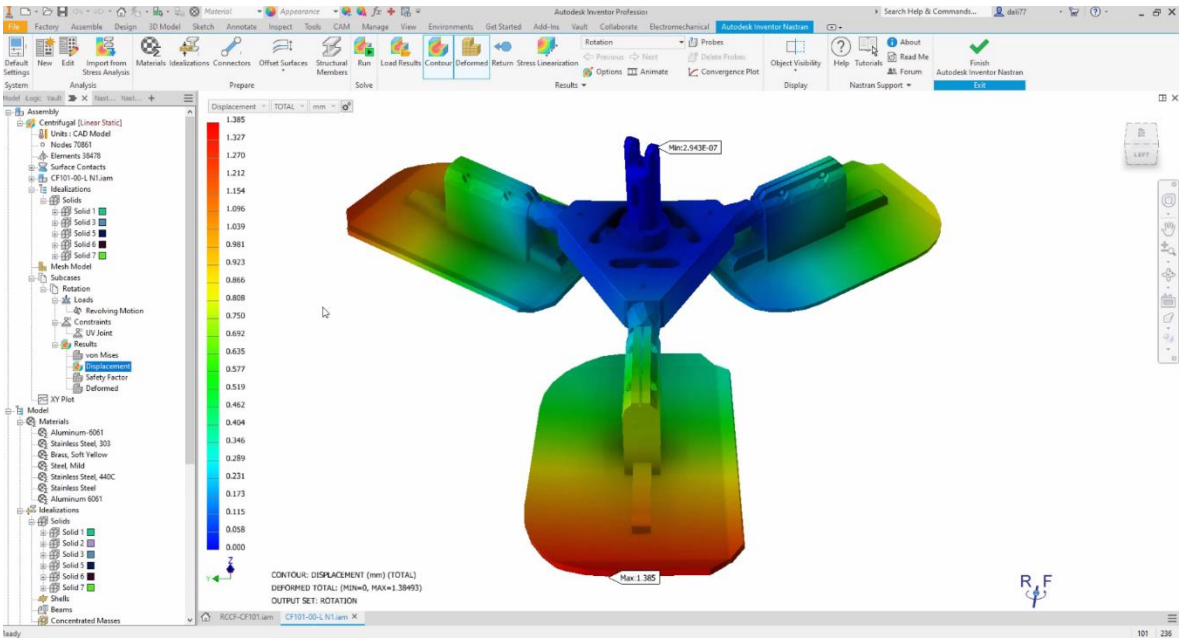
A design engineers working day often includes a great deal of trial and error, making changes to existing information and then having to understand the impact these modifications can cause.

Two such examples are if a design needs to be simulated once a change has been made, or if the related manufacturing information needs to be updated to understand the impact on machining time, tool change and invalid operations caused by the modification itself.

Automating everything to do with these processes could be a little excessive. But by just removing specific user interactions, such as environment switch, contact calculation, mesh generation, toolpath updates and the running of the simulation itself, users will be able to gain faster insights into how any change has potentially impacted an existing setup.

Inventor Nastran

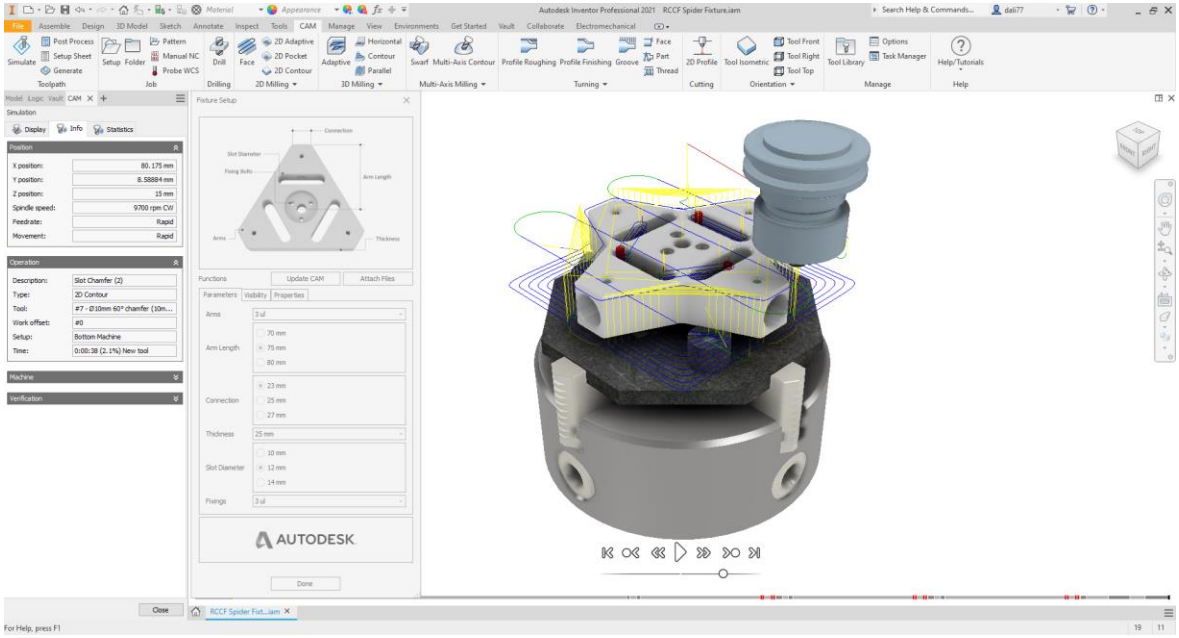
Looking at Inventor Nastran. The user may be continuously making model changes and wishing to understand how those changes have impacted the simulation results.



This code allows a user from the model environment to force the mesh and contacts to update as well as rerunning an existing simulation. Removing a number of steps currently required to re-run the analysis.

Inventor CAM

Similar to the Inventor Nastran workflow, a user can run toolpath updates from within the normal modelling environment. Not just limited to this, we can also take advantage of the API & iLogic to start attaching related documents such as the setup sheets and NC code to the model (via a OLE link).



Not only can a user benefit from a toolpath update to understand the model change impact, but we can also ensure all the respective information is attached the model when we start managing files within Autodesk Vault.

Below is a generic code snippet that can be created to prompt for a selection of files to attach as an OLE link. Ideally this should be created within a rule and that rule suppressed, so it is only activated deliberately. This could be if the rule is added as a button into a form or via another specific trigger function.

```
Option Explicit On
'Used to OLE Link so checkin to Vault includes files.
'current document
Dim doc As Inventor.Document = ThisDoc.Document
'Verify the current document has been saved.
If doc.FullFileName = "" Then
MessageBox.Show("This file must be saved first.")
Exit Sub
End If
'default folder
Dim FolderName As String = Left$(doc.FullFileName, InStrRev(doc.FullFileName, "\") )
```

```
Dim selectedfile As String = String.Empty
Dim oFileDlg As Inventor.FileDialog = Nothing
InventorVb.Application.CreateFileDialog(oFileDlg)
'oFileDlg.Filter = "Dwg files (*.dwg)|*.dwg|Excel files (*.xlsx)|*.xlsx|pdf files
(*.pdf)|*.pdf|Inventor parts (*.ipt)|*.ipt"
oFileDlg.InitialDirectory = FolderName
oFileDlg.CancelError = True
oFileDlg.MultiSelectEnabled = True
Try
oFileDlg.ShowOpen()
selectedfile = oFileDlg.FileName
Catch
Return 'operation was cancelled by the user
End Try
Dim oleReference As ReferencedOLEFileDescriptor
If selectedfile.Contains("|") Then ' we have multiple files selected.
Dim file As String() = selectedfile.Split("|")
For Each s As String In file
oleReference = doc.ReferencedOLEFileDescriptors _
.Add(s, OLEDocumentTypeEnum.kOLEDocumentLinkObject)
oleReference.BrowserVisible = True
oleReference.Visible = False
oleReference.DisplayName = Mid$(s, InStrRev(s, "\") + 1)
Next
Else
oleReference = doc.ReferencedOLEFileDescriptors _
.Add(selectedfile,OLEDocumentTypeEnum.kOLEDocumentLinkObject)
oleReference.BrowserVisible = True
oleReference.Visible = False
oleReference.DisplayName = Mid$(selectedfile, InStrRev(selectedfile, "\") + 1)
End If
```

Reference Material

Automating the execution of Inventor Nastran via the API/iLogic is covered within within the Inventor help system here.

A dedicated AU class MFG227120 from 2018, goes into detail how to automate aspects of Inventor CAM and can be found here.

Summary

Although the material in this class has covered a number of different topics, hopefully it has given you an understanding of how designs can be modularised and automated, as well as an appreciation for what is possible.

If you want to ask questions or reference previously solved challenges, take a look at the Inventor Forums, especially the one below for iLogic and coding advice.
https://forums.autodesk.com/t5/inventor-customization/bd-p/120

---End of document---