IM85128
Modular Design, Design Standards, Function Automation using Inventor & iLogic
Demir Ali
Senior Technical Specialist | @deali12
About the speaker

Demir Ali
Senior Technical Specialist in Design and Manufacturing for Autodesk UK and Ireland.
Joined Autodesk in 2012 from Advanced Micro Devices, having previously worked with the Autodesk manufacturing portfolio since leaving college.
Primary focus is the Product Design and Manufacturing Collection, Vault & VRED.
When not at work, enjoy spending time sculling on the Thames, exploring trails on the mountain bike or getting out on the motorbike.
Class Summary

• Learning everything you need to know about Inventors design automation in a 60-minute session might not be feasible, BUT this session will introduce you to some of the out of the box capabilities now available.

• This class covers, how to build intelligent models, lock in design standards and automate elements of your design including:
  o iLogic fundamentals.
  o Considerations when developing modular designs.
  o Building Inventor assemblies.
  o Automating drawings.
  o Connecting to other sources of information.
  o Design Automation with Forge.
  o Automating functions within a file such as CAM and FEA.
Gateway to Automation

INVENTOR

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

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.

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.

FORGE

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

PLM/ERP/MRP

BUSINESS SYSTEMS
Connect relevant information to other systems, such as Bill of Material extraction, CPQ integration or other business related processes.
Autodesk Inventor | iLogic Improvements

- All components reside within the assembly.
- LEVEL of DETAILs are used to activate or suppress relevant components.
- Components can be replaced, assuming the same constraints and geometry are used.

- iLogic new component placement methods supported.
- Components can be dynamically Added / Removed.
- Capture component placement and position as a snippet.

- iLogic now supports new drawing automation capabilities.
- Inventor has improved support for sheet formats and templates.
- Inventor Design Automation available on Forge.
Design Automation | Time Saving

- Automated exports
- Function automation
  - iProperties
  - Parameter limits
  - Data Entry

- Complex User Inputs
  - Design Reuse
  - Design standards
  - Drawing automation
  - Part & Assembly configuration

MINS

HOURS
Design Automation | Modularisation

• **Customer Need**
  - Control complex design variants
  - Modularise intelligent sub-systems
  - Automate tedious tasks

• **Benefits**
  - Faster reuse of existing design data
  - Ensure compliance with technical design standards
  - Maintain design knowledge directly within models
  - Increase engineering productivity
Utilising modular design techniques
Fundamentals | iLogic Rules

- Based upon Visual Basic, Inventor’s iLogic ‘programmes’ are known as Rules
- Rules can be a combination of both iLogic and Inventor API, without the need for it to be compiled
- Rules can be saved within a document, or as a separate file

```plaintext
lengthinc = 200 ' length rounding increment
inc = 125 mm ' rounding increment

If height < 2500 mm Then
    height = 2500 mm
ElseIf height > 3500 mm Then
    height = 3500 mm
Else
    height = Round(Round(height, 4) / inc) * inc
End If

If length < 2600 mm Then
    length = 2600 mm
ElseIf length > 10000 mm Then
    length = 10000 mm
Else
    length = Round(Round(length, 4) / lengthinc) * lengthinc
End If
```
Fundamentals | Browser Naming

• Override the browser name of any assembly or part that is going to change
  o This allows for rules to function, even when file names change.
    ▪ Ensure naming is unique, otherwise the rule won’t know which component to change.
Fundamentals | Parameters

• Utilise iLogic to easily drive parameters of sub-components
• Any parameter change automatically triggers iLogic rules to be executed
• Use consistent naming for parameters
  o ‘Diameter’ and ‘diameter’ are considered different parameters, as the code (like VB) is case sensitive

Parameter("Trowel Guard", “Diameter") = Diameter
Parameter("Trowel Guard", “Colour") = Colour
Parameter("Trowel Spider", “Trowels") = Trowels
• Rules determine & drive the conditional behaviour of your Inventor designs.
  o Often rules are used to control feature suppression & parameter definitions
• The most common iLogic applied is the use of If, ElseIf, Else, End If statements.

If Diameter < 700 mm Then
  Diameter = 700 mm
ElseIf Diameter > 1100 mm Then
  Diameter = 1100 mm
Else
  Diameter = Diameter
End If

Always finish with an Else statement to have as a ‘fall back’ option
Fundamentals | Logic Statements

- Avoid complex If, Else If, Else, End If statements
  - Reference controlling parameters for one statement
  - Use Select Case or Sub Functions for more complex rules

- In reality there is a much simpler way to define this type of configuration option.

- The feature suppression is now just reliant on a single Boolean parameter.
Fundamentals | Logic Statements

- Another common iLogic definition uses Select Case, Case, End Case statements.
- Select Case allows for easier definition of multiple fixed outcomes or to include ranges of values.

```plaintext
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
```
• Parameter changes will always trigger ACTIVE rules by default
• The default form will activate an update every time a parameter is changed
  - To allow multiple entries before changing, edit the form default to use the ‘OK’ option
Automation Planning
Design Automation | Considerations

196 COMPONENTS
92 UNIQUE FILES
10 FILE MODIFICATIONS
240 CONFIGURATIONS
Design Automation | Embedding Rules

• Avoid overly complex rules, break down the iLogic rules into manageable chunks
• What should be applied within the top-level file?
  o **Overall** parameters such as – Length, Width, Roller size, Colour
  o Rules that will control **ALL** component logic & parameters

![Diagram showing Main Assembly and sub-assemblies]

*.csv, *.txt, *.xlsx
• What should be defined within a part or sub-assembly?
  - Anything that will allow the sub-component to be easily reused elsewhere
  - Colour Finish / Material should be driven from the Part level
  - iProperty updates are typically applied at the Component level
You don’t always need to have an iLogic rule to drive every aspect of a configuration.

By driving a parts Length, other values can just rely on a standard parameter equation.

If a component will be used again elsewhere, it’s best to include the rule within the file.

- As an added benefit, another component would only need to pass a small number of parameters to drive complex rule changes within the component.
Design Automation | Changing a Parts Appearance

• Suitable for both applying components colour or material.
• Within an Inventor assembly it’s possible to override a components appearance.
  o You are not changing the parts appearance, only how it appears within the assembly.
  o To ensure the part colour changes, the appearance must be applied within the part.

• The finish can be driven with a text parameter using any valid appearance name.
Planning | Dependent Rules
Building Assemblies
Assemblies | Positional Relationships

• You no longer have to build a ‘150% model’ & suppress unnecessary components.
• Components can be added / deleted / switched out on demand.
• When creating the code, consider:
  o How are components positioned / constrained?
  o Are constraints reliant on geometry that changes?
  o Are origin or user defined work features used?
    ▪ A work feature Normal will dictate a **Flush** or **Mate** constraint.
  • You can override Origin/User feature names if required.
Assemblies | Defining Positions

- iLogic component positioning rely on using Assembly CONSTRAINTS NOT JOINTS.
- Constraints rely on defining positions at the using either:
  - Named Geometry (Face, Edge, Vertex)
  - Origins / User defined Work Features.
• Geometry Labels are defined at PART level only.
• When creating switchable options, apply the same names for the same location points.
  o This makes downstream coding easier.
• Labels can be referenced for Component placement AND Drawings dimension / balloon placement.

Assemblies | Geometry Labels

YZ Plane
Labelled Face “Drive”
XY Plane

100Hp Option 1

160Hp Option 2
Assemblies | Creating Geometry Labels
Assemblies | Setting up the Add / Delete / Replace

- Define multi value text parameters to select which component is needed.
  - If switching components in the same position, remember try to keep identical Origin and Label naming
- Boolean parameters can dictate if a component is added/removed.
- When using forms, selection of components can be enabled/disabled with the Boolean parameter.
Assemblies | Parameter Setups
Assemblies | Capturing Add / Delete / Replace

- First, build the completed version and if switching components, include one of it’s variants.
- When specifying filenames, components can reside within any folder in the active project.
- The ‘Manage Components’ snippet alongside a Boolean will act as both Add / Delete

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")
Assemblies | Managing Components
Assemblies | Bill of Materials

- The below shows two configurations.
  - Configuration 1 will always have 2 unique components (the red ones)
  - Configuration 2 could reuse the same part for both sides.
- The user has a decision to make when creating a model to accommodate both configurations.
  - When going from 1 to 2, remove the initial part and replace with the alternative.
  - Edit the initial part, suppress the relevant features and update the Part Number to the other component.
Drawing Automation | Added Intelligence

- In the past a user would have to rely heavily on the API to automate drawings.
- Accommodating annotations for geometry that may or may not exist was always a challenge.
- Users can now capture code, to enable automation of annotation creation.
It's difficult to automate annotations, if there are many model configurations available.

With consistent component and geometry naming, annotation can now be automated with minimal programming knowledge.

Supported types of Annotations now include:

- Linear, Angular, Radius & Diameter Dimensions
- Centerlines and Center Marks
- Leader Notes
- Hole & Thread Notes
- Balloons

```vbnet
' Dimension Views
Dim oName As ActiveSheet.Name
Dim oSheet = ThisDrawing.Sheets.ItemByName(oName)
Dim oView = oSheet.DrawingViews.ItemByName(" STEM")

' Dimensions on Front View
Dim namedSec1 = oView.GetIntent(" Trowel Guard", " Dim1")

' Dimension Handle
Try
  Dim handleCenter1 = oView.GetIntent({" Front Assem", "Arrow", " Dim1", PointIntent})
  Dim linDim3 = genDima.AddLinear(" Dimension3", oView.SheetPoint(-0.3, 0.5),
                                 oView.SheetPoint(0.3, 0.5), Precision = 0)
Catch
  Dim handleCenter1 = oView.GetIntent({" Arrow Assem", "Bar", " Dim1", PointIntent})
  Dim linDim3 = genDima.AddLinear(" Dimension3", oView.SheetPoint(-0.3, 0.5),
                                 oView.SheetPoint(0.3, 0.5), Precision = 0)
End Try
```
• Snippets are created via the context menu, available when selecting model edges within a drawing.
  o If a part edge isn't already labeled, the user is notified the file will be modified to suit.
  o A new Code Clipboard is displayed with the code pasted into it for use.
  o **NOTE:** You cannot utilise labelled geometry if it is not visible or selectable within the view.
Drawing Automation | Annotation Creation

• When placing annotations, 5 things need to be defined:
  1. The sheet that will be used.
  2. The view to be annotated.
  3. View extents annotation placement (X,Y).
  4. Geometry to annotate.
  5. Annotation type with 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)
```

Annotation Co-ordinates

Values are a percentage of the view boundary and can include negative values to offset past it.
Drawings | Capturing the code
Drawing Automation | Dimension Considerations

• To create Balloons, you need only to specify one geometry location.

• When dimensioning, the challenge is more complicated because of a number of factors, such as:
  o What type of dimension do you want to create?
  o Are you dimensioning one edge of one part?
  o Is the dimension from one component to another?
  o Should the dimension be to an edge OR centreline position?
  o What precision / roundup do you need?

• Within the iLogic rule editor, a user will see a tooltip, helping them know the available code options.
• Looking at the 925 dimension, we need to accommodate the following:
  o Dimension from an Edge to the Handle Centre.
  o Make an allowance for two different types of handle needing dimensioning.
  o Round the value to the nearest whole number.

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.

Overrides precision to zero decimal places.

Specifies the centre of the selected geometry.

Only creates a vertical dimension.
Drawings | Creating Dimensions
Drawing creation can be sped up using Sheet Formats.
Sheet Formats can be accessed and selected from the File > New command
Within Sheet Formats, they also store:
- Text
- Parts Lists
- Create View from Model
- Flat Pattern Views
- View Settings
- Source Component
- Fit Views to Sheet
Drawings | Sheet Formats and iLogic
External Lookup | Why?

- Lookups from Excel or a *.csv file can be used to drive model changes or calculate computed values.
- Offers greater flexibility, such as Parameters being referenced externally and bi-directional updates.

- It also provides a mechanism for business system lookups, such as pricing.
  - Cost information should rarely be embedded into CAD models because:
    - It changes regularly.
    - Is usually managed by other business systems.
    - If pricing is ‘hard coded’, it makes it difficult to update.
    - How will the total will be calculated (Sub-Assembly, Parts Only, Logic controlled).
External Lookup | Accessing a shared data file

- Building an iLogic rule to interrogate a separate source of data

```plaintext
Dim folderpath As String = ThisDoc.Path
Dim fileContents As String = IO.File.ReadAllText(folderpath & "/AdskPriceList.csv")
Dim lines() As String = fileContents.Split(New String() {vbNewLine, vbCr}, StringSplitOptions.RemoveEmptyEntries)
Total=0
For Each Line In lines
    X=InStr(1,Line, ",")
    PrtNum=Left(Line,X-1)
    Price=Mid(Line,X+1, Len(Line))
    Try
        quantity = ThisBOM.CalculateQuantity("Parts Only", PrtNum)
    Catch
        quantity = 0
    End Try
    Total=Total+quantity*Convert.ToDouble(Price)
Next
```

The CSV reference used for the lookup

Code to read, split and filter CSV content

Zero the user defined Text parameter called 'Total'. This parameter can also be shown in a Form (Read Only) if set as a key parameter.

Using the BOM table ensures quantities are accounted for.
- Model Data
- Structured
- Parts Only
  - If priced per part rather than per sub-assembly.
Referencing external sources of data
Forge Design Automation
Design Automation on Forge

Externally hosted
Model data stored & retrieved
User feedback at this stage
  - Forms Interaction, Limits, etc

Instance run up on demand
Model data loaded
Tasks executed
Data output & delivered
All data deleted & instance spun down
Design Automation with Forge

• Inventor data can now be used in cloud-native applications to automate at scale.
• Create workflows linking Autodesk web applications to other enterprise applications.
• Reference the Design Automation app available now on GitHub.
  o https://github.com/Developer-Autodesk/forge-configurator-inventor
Inventor Design Automation on the Cloud
Automating Functions
After updating a design, a user often wants to quickly see how it impacts downstream.

- This could be something relatively simple like:
  - How does it impact an existing simulation?
  - What effect does it have on existing CAM toolpaths?

- This often means, activating the respective environment and rerunning the commands manually.

- However, there are sample API snippets how to execute these commands from within iLogic.
  - For Inventor Nastran, there is an iLogic Automation topic within the help system [here](#).
  - There is an AU class covering automating Inventor CAM updates [here](#).
Automate Functions | Updating Nastran
Automate Functions | Updating CAM
Class Summary

• This session gave you an overview of what is needed to build intelligent models, how information can be captured and how elements of the design can be automated.

• Design Automation is a big topic and there are many AU classes available for reference covering different aspects of design automation.

  o Here’s some other classes from this years AU you might want to check out:
    ▪ MFG468558 Maximizing your Inventor Templates
    ▪ IM469414 Drawing Automation with API and new iLogic snippets in Inventor 2021
    ▪ MFG467190 Take It From The Top – iLogic Best Practices and Fundamentals for Success
    ▪ MFG473705 Using iLogic with Design Automation for Inventor to create a Configurator
    ▪ IM468712 iLogic & Vault | Vault & iLogic
    ▪ MFG463406 Why hire an API programmer for your CAD department?
    ▪ FAB466294 Anybody Can Do IT! Easily Build Revit Content in Inventor