



# iLogic: 25 Tips and Tricks to Boost the Octane in Your Inventor Automation

MFG501293

Curtis Waguespack  
Automation Solution Consultant



# About Me

**Curtis Waguespack**  
[wag] [əs] [pak]



- I've used Inventor and AutoCAD in the real-world for over 20 years.
- This has included a great deal of Inventor automation with iLogic.
- I started out modifying LISP routines and writing VBA for AutoCAD.
- I currently work as an Automation Solution Consultant, where I help professionals automate their designs.
- I teach iLogic classes, and in the past I have taught general Inventor and AutoCAD classes.
- Additionally, I've authored and co authored multiple editions of the **Mastering Autodesk Inventor** book.



# About You

## Intended Audience

- ✓ Some of these tips are immediately useful to new iLogic users
- ✓ Some of these tips are more relevant to experienced iLogic users
- ✓ I think all of these tips are worth knowing about, so that you can work them into your automation efforts as the needs arise, regardless of experience level



# Questions and Comments

- ✓ Class format = 90 minutes:
  - 60 minutes of presentation
  - 30 minutes of Q & A
- ✓ Please hold questions and comments until the end
  - Note the number of the tip, for reference later



# Objectives

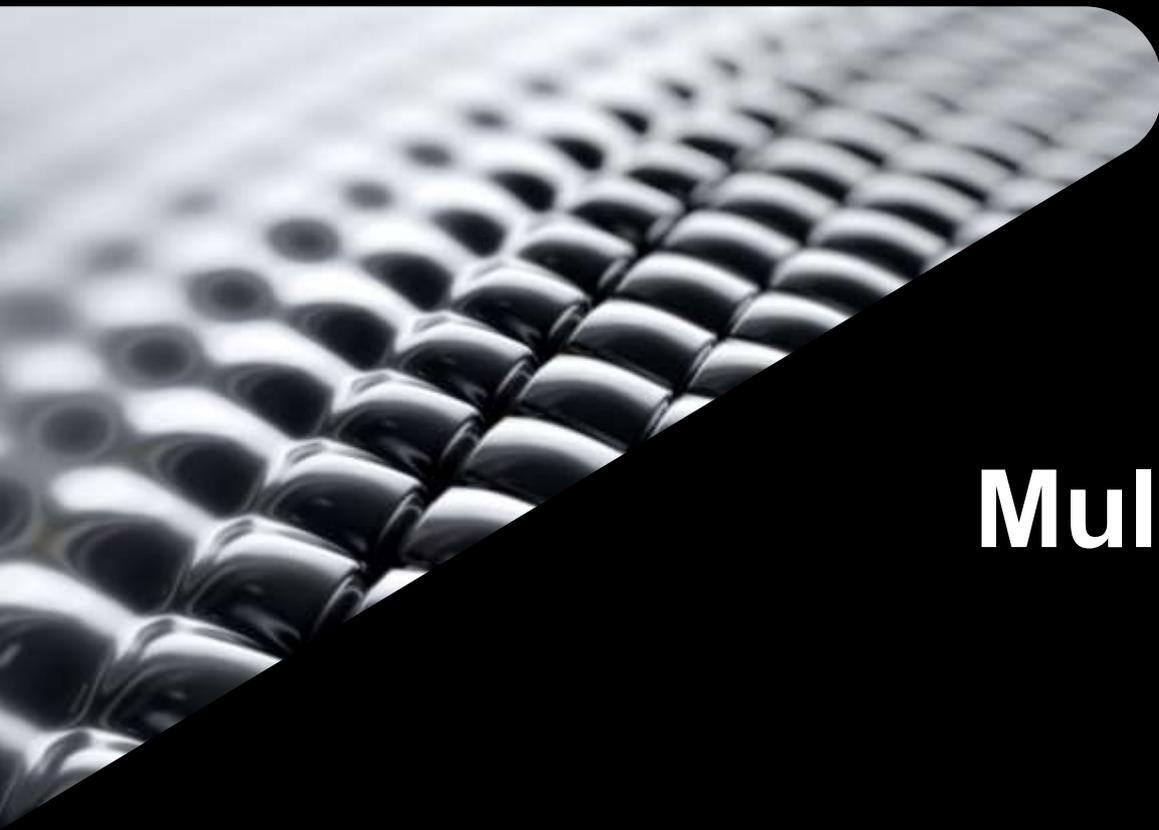
1. Tips and techniques using the **iLogic editor**
2. Finding our way around the **Inventor API**
3. Explore development and **error handling** tools and techniques
4. Create a better **interface** between your iLogic automation and the users who employ it





Objective:

# Tips and Techniques Using the iLogic Editor



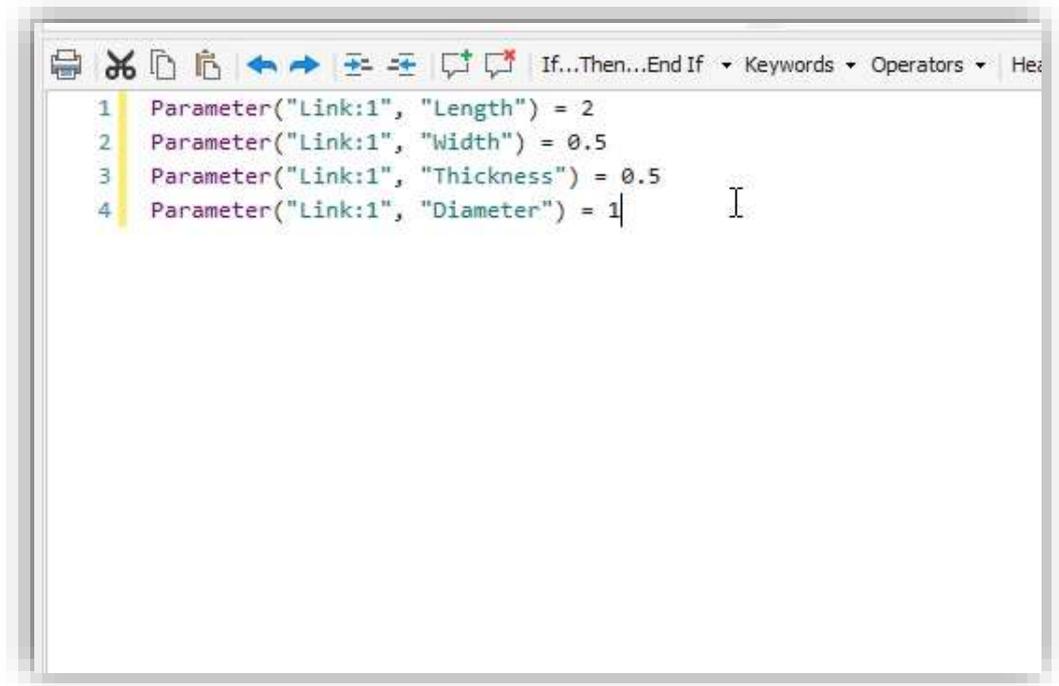
# Multi-Line Typing

Tip #1

# Multi-line editing

- When copying code or changing multiple lines you can enter text to multiple lines at once
- Place your cursor on the first line
- Then hold the **CTRL** key and then select multiple lines to create a multi-line cursor
- Type as you would normally

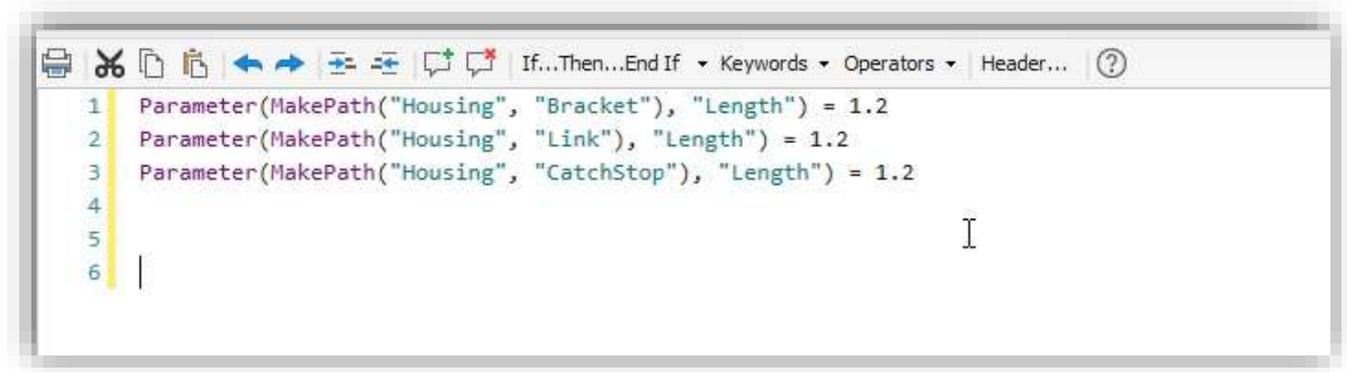
★ Tip provided by **Scott Hallmark**

A screenshot of a code editor window. The editor has a toolbar at the top with icons for print, cut, copy, paste, undo, redo, and zoom. Below the toolbar, there are four lines of code, each starting with a line number (1, 2, 3, 4) in a yellow margin. The code is: 1 Parameter("Link:1", "Length") = 2; 2 Parameter("Link:1", "Width") = 0.5; 3 Parameter("Link:1", "Thickness") = 0.5; 4 Parameter("Link:1", "Diameter") = 1;. A multi-line cursor is visible, consisting of a vertical bar on the left side of the first line and a vertical bar on the right side of the fourth line, indicating that all four lines are selected. The text between the bars is highlighted in yellow. The editor also shows dropdown menus for "If...Then...End If", "Keywords", and "Operators".

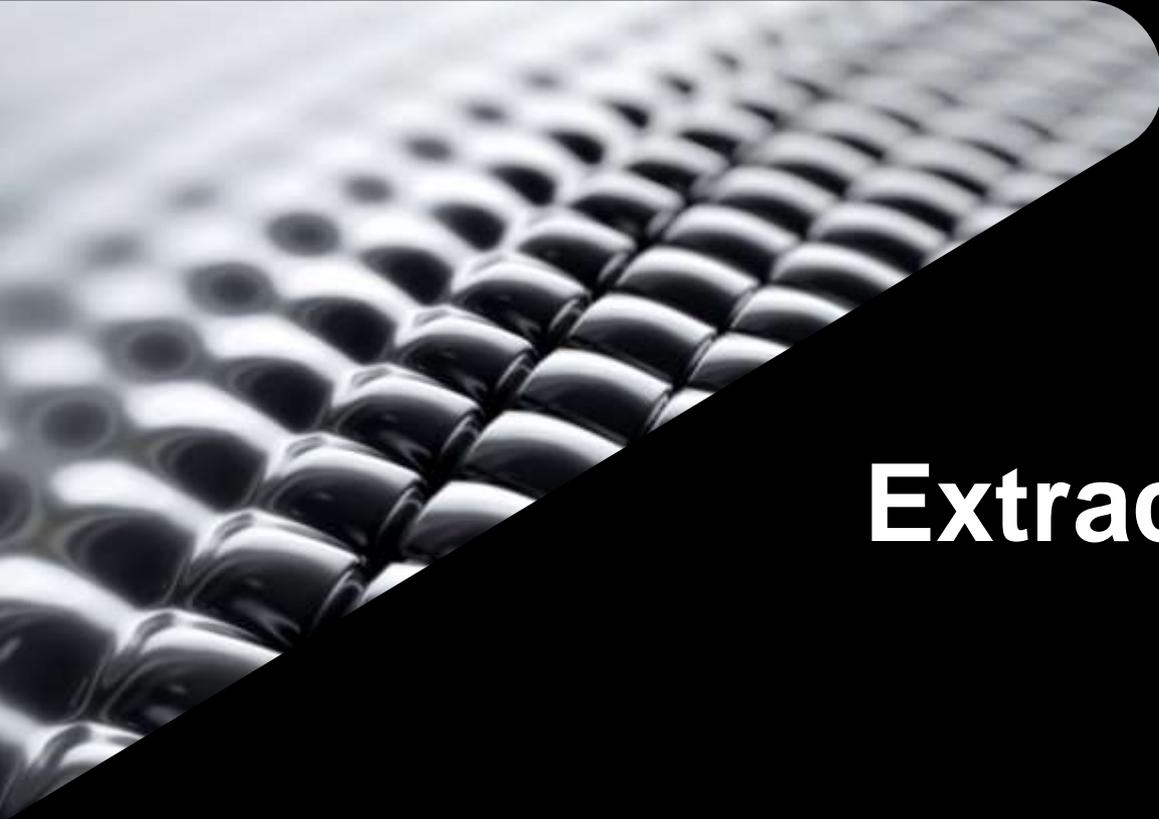
```
1 Parameter("Link:1", "Length") = 2
2 Parameter("Link:1", "Width") = 0.5
3 Parameter("Link:1", "Thickness") = 0.5
4 Parameter("Link:1", "Diameter") = 1
```

# Multi-location line editing

- We can do this in multiple places within the same line as well



```
1 Parameter(MakePath("Housing", "Bracket"), "Length") = 1.2
2 Parameter(MakePath("Housing", "Link"), "Length") = 1.2
3 Parameter(MakePath("Housing", "CatchStop"), "Length") = 1.2
4
5
6 |
```

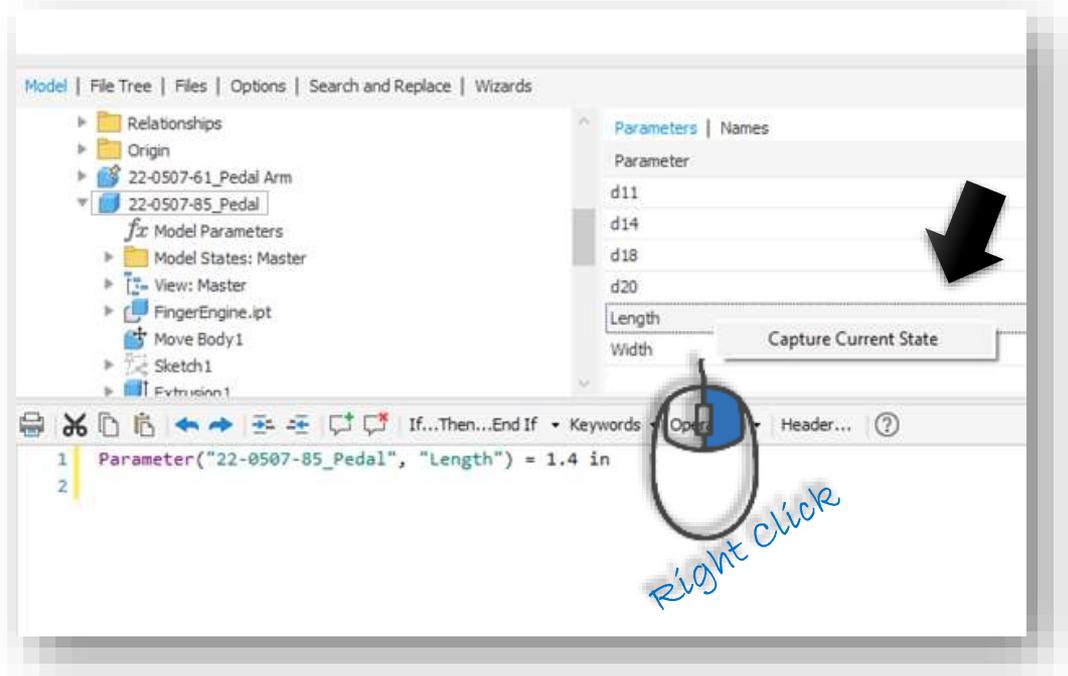


**Extract to be exact!**

Tip #2

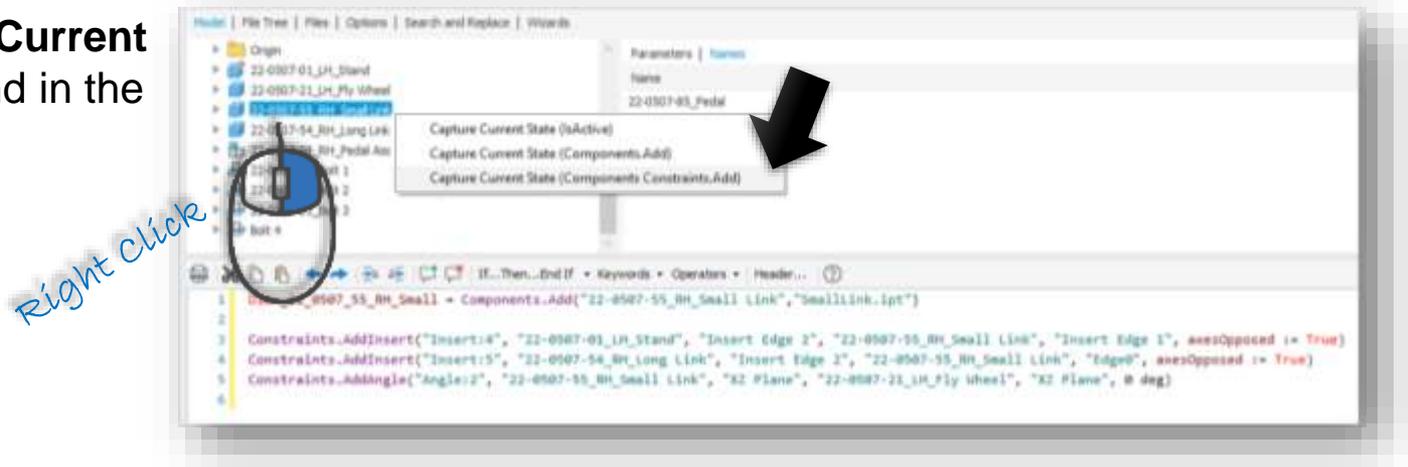
# Extracting a Parameter from a sub-component

- The iLogic Editor allows us to create code by extracting it rather than typing it.
- Use this to your advantage to :
  - Gain efficiency
  - Reduce typo errors
  - Avoid omissions
  - Eliminate syntax issues



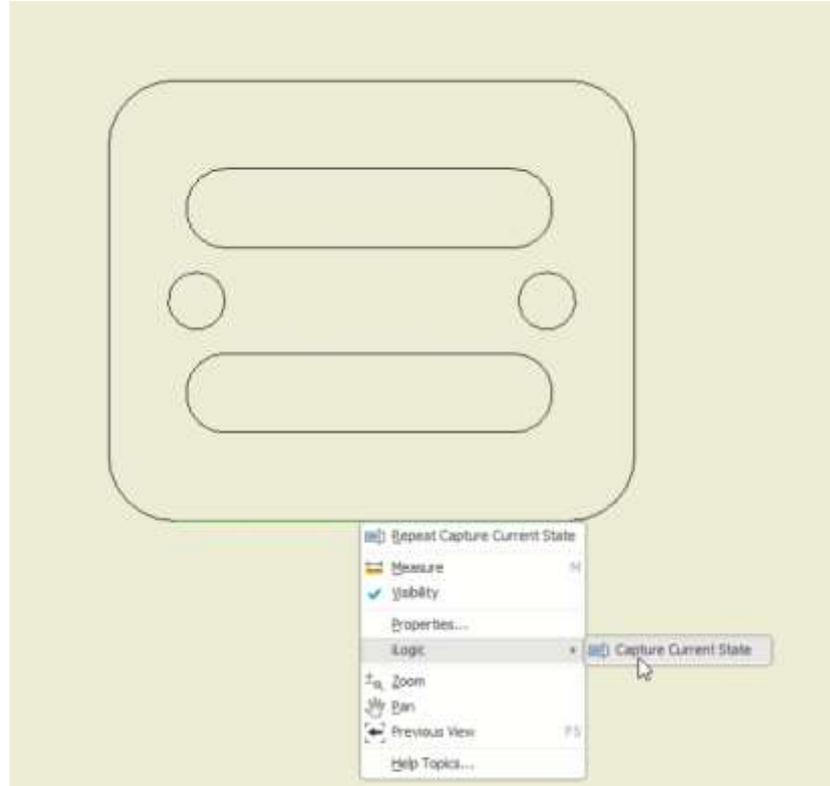
# Extracting the code to place and constrain a sub-component

- There are several different **Capture Current State** options found in the right click menu



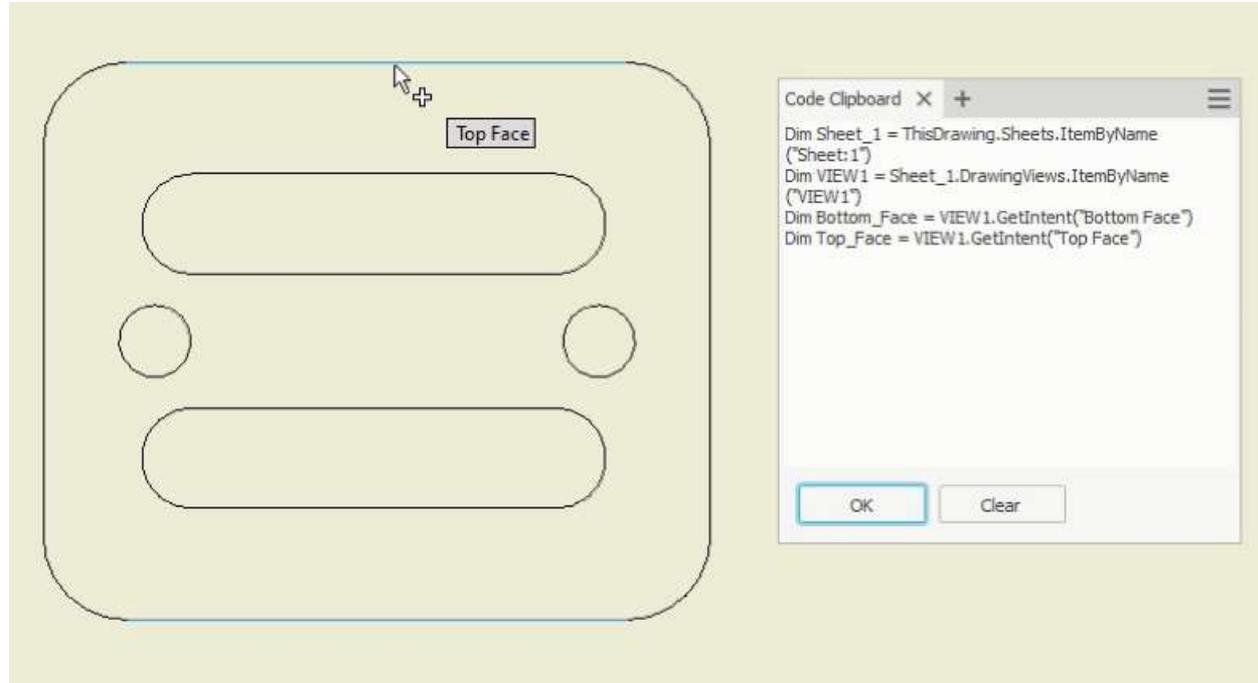
# Extract code from drawing edges

- In a drawing you can right click an edge to extract the code needed to place a dimension



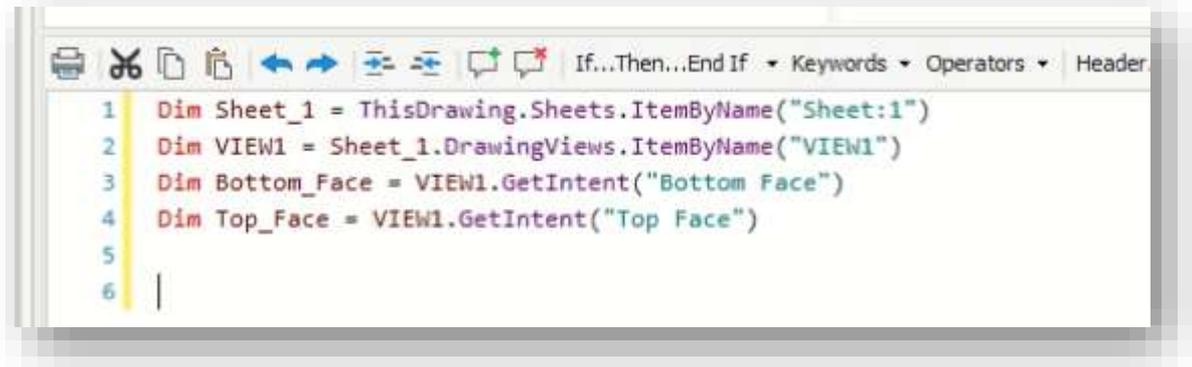
# Extract code from drawing edges

- If there are named geometry entities in a model, we can extract them by right clicking on the drawing edge associated with them
- If the edge does not have a named entity, iLogic will create it in the model for us



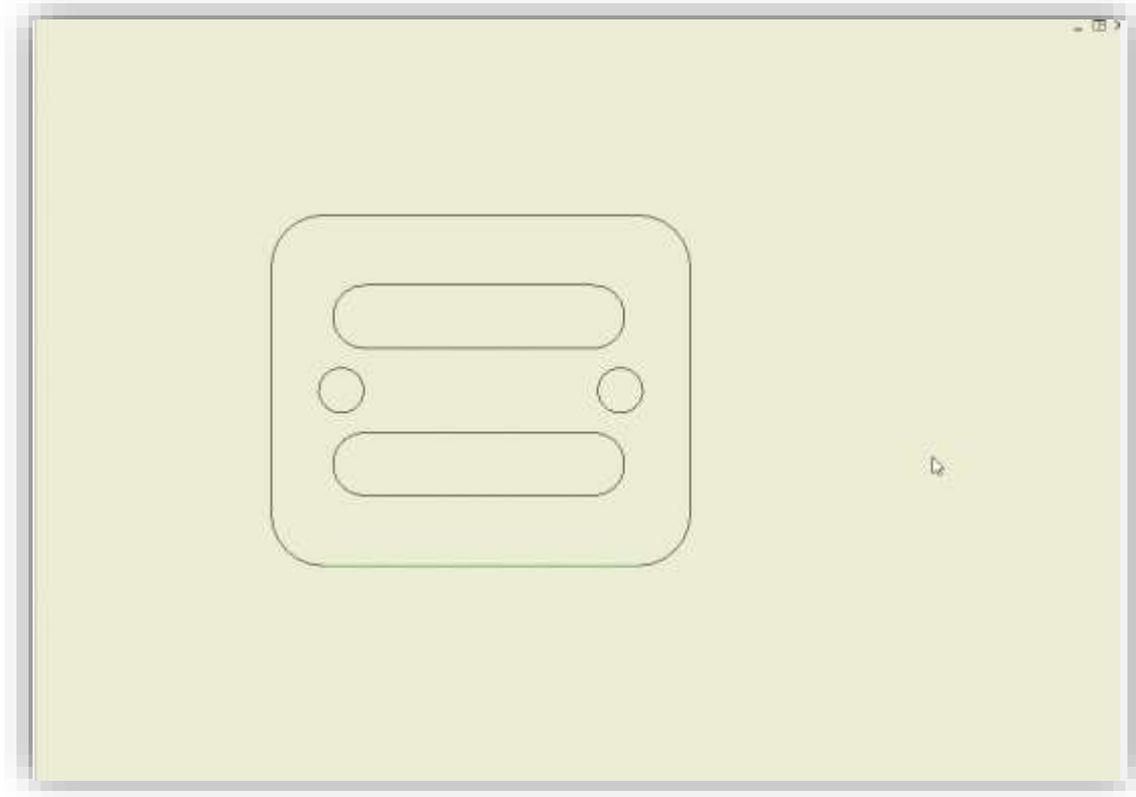
# Paste the extracted code into a rule

- In the iLogic Editor we can then paste in the code from the clipboard to our iLogic rule



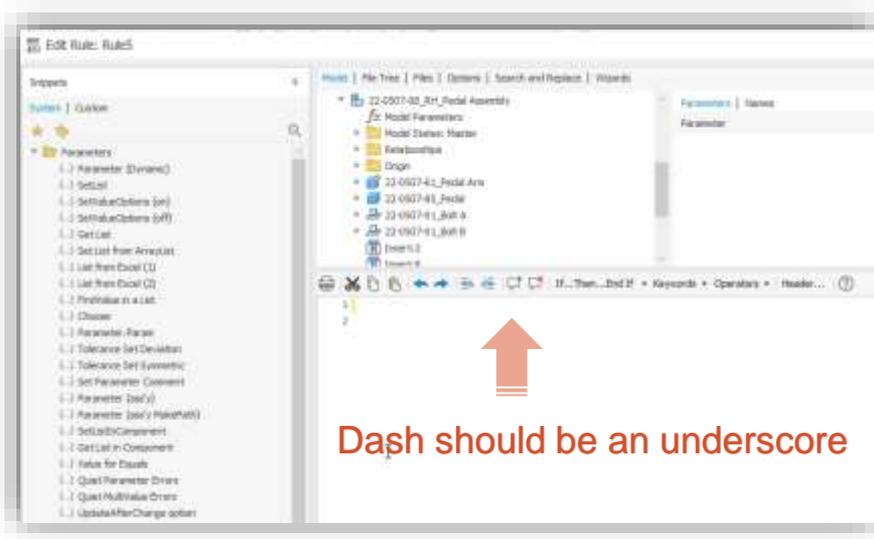
```
1 Dim Sheet_1 = ThisDrawing.Sheets.ItemByName("Sheet:1")
2 Dim VIEW1 = Sheet_1.DrawingViews.ItemByName("VIEW1")
3 Dim Bottom_Face = VIEW1.GetIntent("Bottom Face")
4 Dim Top_Face = VIEW1.GetIntent("Top Face")
5
6 |
```

# Extracting code to place dimensions

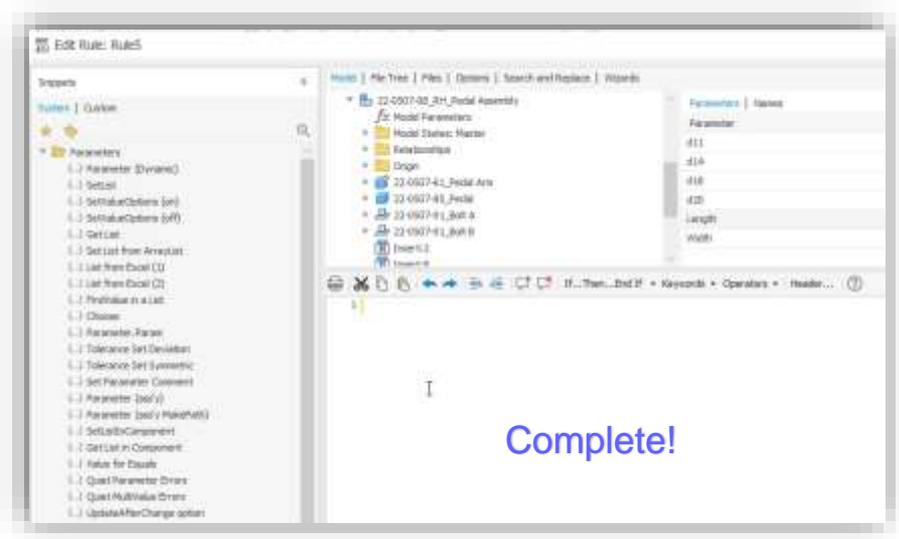


# Extracting Comparison

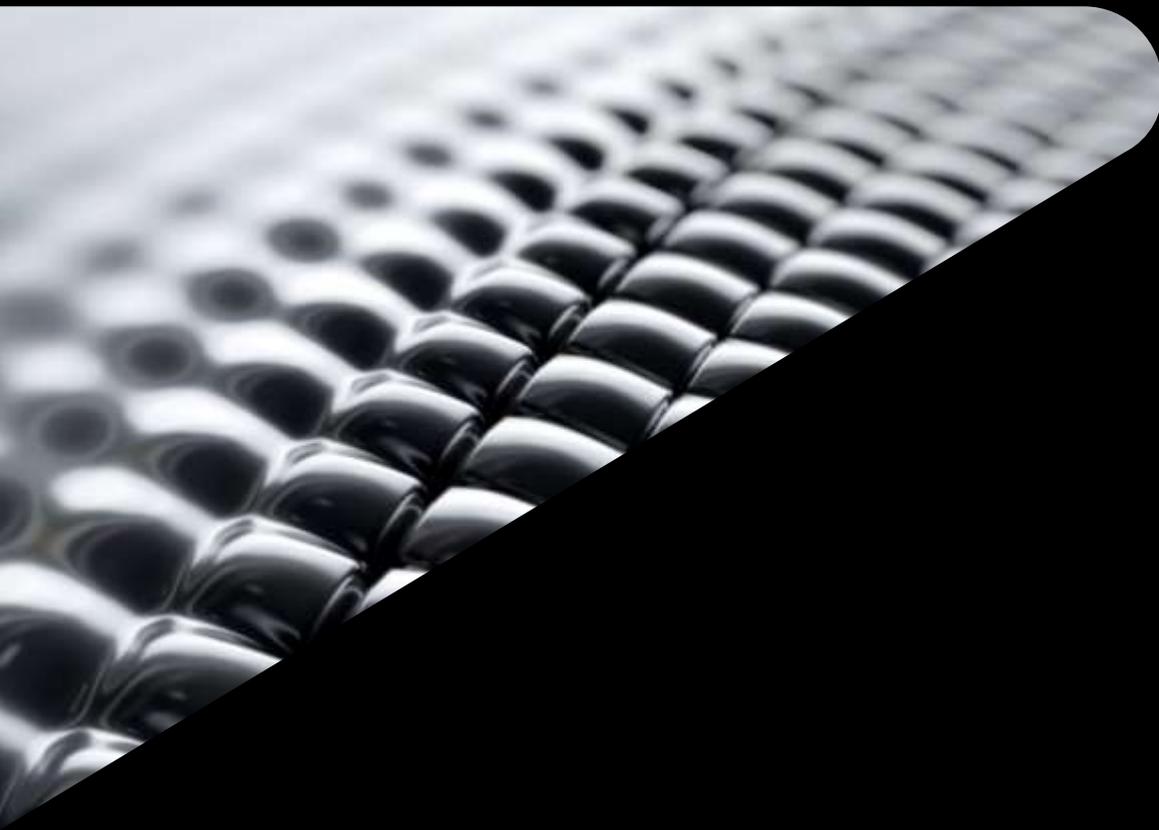
## Typing vs extracting



Typing the code (with typo)



Extracting the code



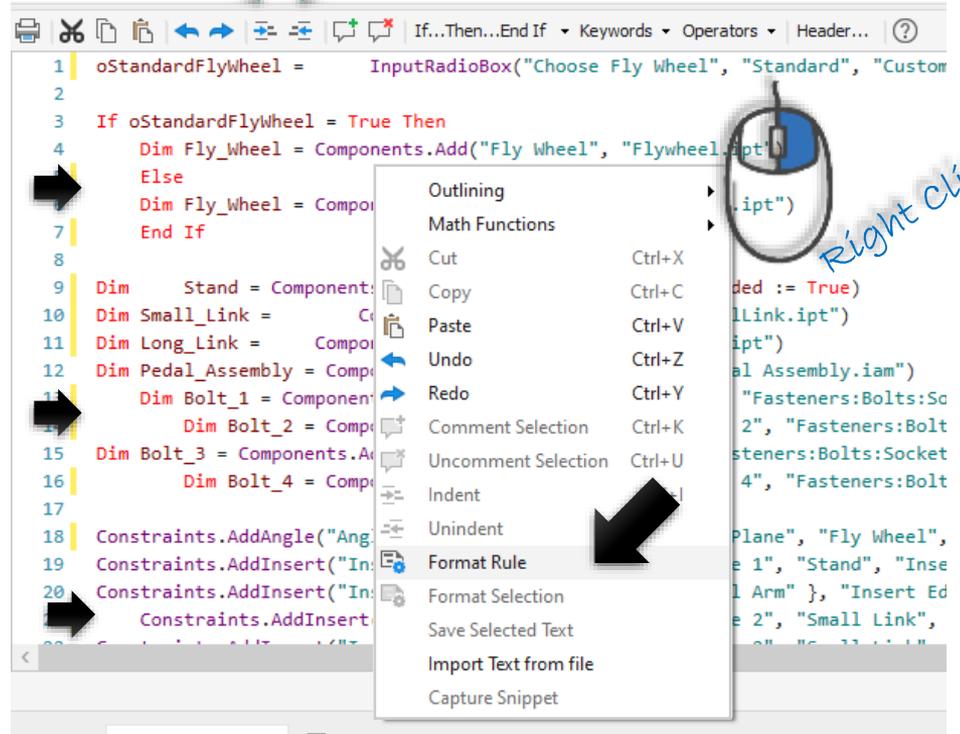
# Right Click: Format Rule

Tip #3

# Format Rule

- Rather than manually fixing poorly formatted code, you can right click and choose Format Rule
- Remove extra “White Space”
- Fix indents
- Improve your code’s readability
- Save time

## Indent buttons



# Format Rule

The screenshot shows the 'Edit Rule' dialog box in Autodesk Inventor. The title bar reads 'Edit Rule: Place and Constrain with Fly Wheel option'. The left sidebar shows the 'Imports' tree with 'Custom' selected. The main area displays a VBA script for a rule named 'eStandardFlyWheel'. The script uses an 'InputRadioBox' to allow the user to select between 'Standard' and 'Custom' fly wheel options. It then uses 'Components.Add' to place the selected fly wheel and other parts like 'Stand', 'Small link', 'Long link', and 'Pedal Assembly'. Finally, it uses 'Constraints.AddInsert' to define geometric constraints between the fly wheel and other components.

```
1 eStandardFlyWheel = InputRadioBox("Choose Fly wheel", "Standard", "Custom", True, "iLogic")
2
3 If eStandardFlyWheel = True Then
4   Dim Fly_wheel = Components.Add("Fly Wheel", "Flywheel.ipt")
5 Else
6   Dim Fly_wheel = Components.Add("Fly Wheel", "Flywheel2.ipt")
7 End If
8
9 Dim Stand = Components.Add("Stand", "Stand.ipt", grounded := True)
10 Dim Small_link = Components.Add("Small link", "Smalllink.ipt")
11 Dim Long_link = Components.Add("Long link", "LongLink.ipt")
12 Dim Pedal_Assembly = Components.Add("Pedal Assembly", "Pedal Assembly.iaw")
13 Dim Bolt_1 = Components.AddContentCenterPart("Bolt 1", "Fasteners\Bolts\Socket Head", "Hexagon Socket Head Cap Screw - Inch", "No. 6 - 32 UNF - 1")
14 Dim Bolt_2 = Components.AddContentCenterPart("Bolt 2", "Fasteners\Bolts\Socket Head", "Hexagon Socket Head Cap Screw - Inch", "No. 6 - 32 UNF - 1")
15 Dim Bolt_3 = Components.AddContentCenterPart("Bolt 3", "Fasteners\Bolts\Socket Head", "Hexagon Socket Head Cap Screw - Inch", "No. 6 - 32 UNF - 1")
16 Dim Bolt_4 = Components.AddContentCenterPart("Bolt 4", "Fasteners\Bolts\Socket Head", "Hexagon Socket Head Cap Screw - Inch", "No. 18 - 34 UNF - 1")
17
18 Constraints.AddAngle("Angle:1", "Small link", "XZ Plane", "Fly wheel", "XZ Plane", # deg)
19 Constraints.AddInsert("Insert:1", "Fly wheel", "Insert Edge 1", "Stand", "Insert Edge 1", axesOpposed := True)
20 Constraints.AddInsert("Insert:2", {"Pedal Assembly", "Pedal Arm"}, "Insert Edge 3", "Stand", "Insert Edge 3", axesOpposed := True)
21 Constraints.AddInsert("Insert:4", "Stand", "Insert Edge 2", "Small link", "Insert Edge 1", axesOpposed := True)
```

# Format Rule

- Be aware that there is a slight bug with the Format Rule tool, in how it handles these two else statements
  - **Else If** ( with space) formats unexpectedly
  - **Elseif** ( without space) formats as expected
- Both work, as far as the code running correctly, but the Format Rule tool “sees” them differently



```
8 If oSize < 3 Then
9   Parameter("Link Arm", "Length") = 7
10  Parameter("Link Arm", "Width") = 1
11  Else If oSize < 7 Then
12  Parameter("Link Arm", "Length") = 9
13  Parameter("Link Arm", "Width") = 3
14  ElseIf oSize < 9 Then
15  Parameter("Link Arm", "Length") = 11
16  Parameter("Link Arm", "Width") = 5
17  End If
```

```
8 If oSize < 3 Then
9   Parameter("Link Arm", "Length") = 7
10  Parameter("Link Arm", "Width") = 1
11  Else If oSize < 7 Then
12  Parameter("Link Arm", "Length") = 9
13  Parameter("Link Arm", "Width") = 3
14  ElseIf oSize < 9 Then
15  Parameter("Link Arm", "Length") = 11
16  Parameter("Link Arm", "Width") = 5
17  End If
```

A close-up, black and white photograph of a metal mesh or woven fabric, showing a grid of circular openings. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# Add Code Structure with Sub Procedures

Tip #4

# What is a Sub Procedure ?

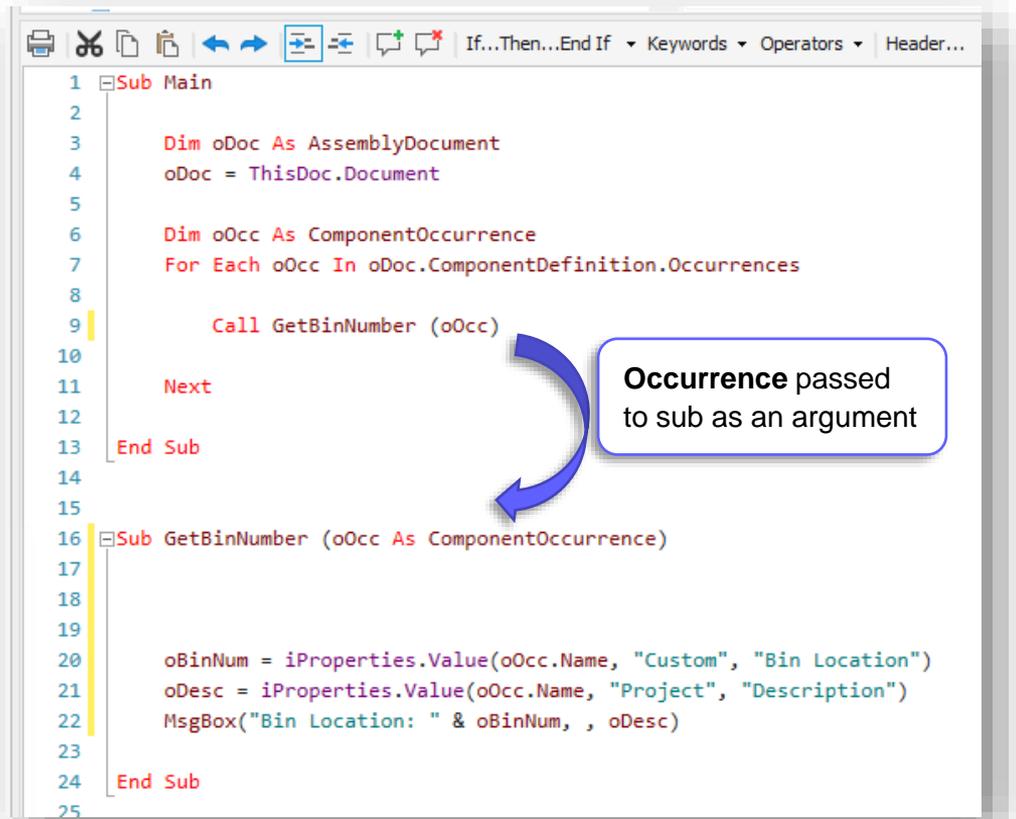
- A Sub Procedure is a collection of statements enclosed by the **Sub** and **End Sub** statements
- The Sub procedure performs a task and then returns control to the code from which it was called
- When using a Sub, we must include a **Sub Main**
- But it does not return a value to the calling code

```
1 Sub Main
2
3     Call SayHello
4
5 End Sub
6
7 Sub SayHello
8
9     MessageBox.Show("Hello World!", "iLogic")
10
11 End Sub
```

A blue curved arrow starts at the 'Call SayHello' statement on line 3 and points to the 'Sub SayHello' block starting at line 7. A second blue curved arrow starts at the 'End Sub' statement on line 11 and points back to the 'End Sub' statement on line 5, illustrating the return of control to the calling code.

# Passing information to the Sub

- Often, we pass information to a Sub procedure in the form of arguments
- The sub requires the argument's data type to be declared
  - technically the argument is called a parameter at the sub level
- Syntax for calling a Sub procedure:
  - Call SubName (Argument1, Argument2)
- The use of the **Call** keyword is optional

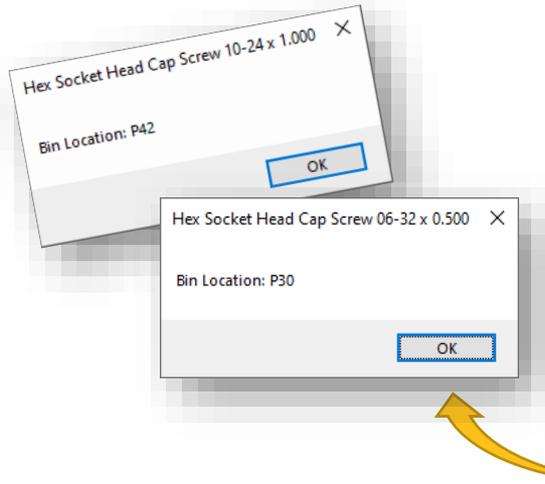


```
1 Sub Main
2
3     Dim oDoc As AssemblyDocument
4     oDoc = ThisDoc.Document
5
6     Dim oOcc As ComponentOccurrence
7     For Each oOcc In oDoc.ComponentDefinition.Occurrences
8
9         Call GetBinNumber (oOcc)
10
11     Next
12
13 End Sub
14
15
16 Sub GetBinNumber (oOcc As ComponentOccurrence)
17
18
19
20     oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
21     oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
22     MsgBox("Bin Location: " & oBinNum, , oDesc)
23
24 End Sub
25
```

**Occurrence passed to sub as an argument**

# Diagramming a Simple Sub Procedure

- Sub Name
- Argument/Parameter
- Sub Statement Code



```
1 Sub Main
2
3   Dim oDoc As AssemblyDocument
4   oDoc = ThisDoc.Document
5
6   Dim oOcc As ComponentOccurrence
7   For Each oOcc In oDoc.ComponentDefinition.Occurrences
8
9       Call GetBinNumber (oOcc)
10
11   Next
12
13 End Sub
14
15
16 Sub GetBinNumber (oOcc As ComponentOccurrence)
17
18
19
20   oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
21   oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
22   MsgBox("Bin Location: " & oBinNum, , oDesc)
23
24 End Sub
25
```

# Why use Sub Procedures ?

- Subdividing large amount of code into smaller subs make your code more:
  - Readable
  - Maintainable
- Procedures are useful for performing repeated or shared tasks, such as:
  - Calculations
  - Formatting
  - Data operations
- You can use procedures as building blocks for your iLogic rules.
- Allows us to reuse, rather than copy/paste the same code over and over

```
1 Sub Main
2
3     Dim oDrawDoc = ThisDrawing.Document
4     Dim oModel = ThisDrawing.ModelDocument
5
6     Call ProcessAssembly(oModel)
7     Call Publish_PDF(oDrawDoc)
8     Call DXF_Out(oDrawDoc)
9     Call SetApprovalStamp(oDrawDoc)
10    Call ProcessAssembly(oModel)
11 End Sub
12
13 Sub ProcessAssembly(oMDoc As AssemblyDocument) ...
14
15
16
17 Sub Publish_PDF(oDraw As DrawingDocument) ...
18
19
20
21 Sub DXF_Out(oDraw As DrawingDocument) ...
22
23
24
25
26 Sub SetApprovalStamp(oDraw As DrawingDocument) ...
```



# Use Function Procedures

Tip #5

# What is a Function Procedure ?

- A Function procedure is a collection of statements enclosed by the **Function** and **End Function** statements
- Like a Sub, a Function performs a task and then returns control to the code from which it was called
- Unlike a Sub, a Function returns a value to the calling code
- When using a Function, we must include a **Sub Main**

```
1 Sub Main
2
3     oThickness = Parameter("Thickness")
4     oHoleSize = Parameter("Diameter")
5
6     oOffset = CalculateOffset(oThickness, oHoleSize)
7     MessageBox.Show("Offset: " & oOffset, "iLogic")
8
9 End Sub
10
11 Function CalculateOffset (oThick As Double , oHole As Double)
12
13     If oThick < 3 Then
14         oShimSize = 0.32
15     ElseIf oThick < 5 Then
16         oShimSize = 0.41
17     Else
18         oShimSize = 0.533
19     End If
20
21     oOffset = oThick + oHole - oShimSize*2
22     Return oOffset
23 End Function
```

# Passing information to the Function

- We pass information into a Function the same as we do a Sub
- The Function requires the argument's data type to be declared
  - technically the argument is called a parameter at the function level
- Syntax for calling a Function procedure:
  - Capturing Variable = FunctionName (Argument1, Argument2)

```
1 Sub Main
2
3     oThickness = Parameter("Thickness")
4     oHoleSize = Parameter("Diameter")
5
6     oOffset = CalculateOffset(oThickness, oHoleSize)
7     MessageBox.Show("Offset: " & oOffset, "iLogic")
8
9 End Sub
10
11 Function CalculateOffset (oThick As Double , oHole As Double)
12
13     If oThick < 3 Then
14         oShimSize = 0.32
15     ElseIf oThick < 5 Then
16         oShimSize = 0.41
17     Else
18         oShimSize = 0.533
19     End If
20
21     oOffset = oThick + oHole - oShimSize*2
22     Return oOffset
23 End Function
```

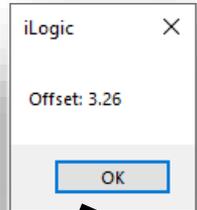
Parameter values  
passed to function as  
arguments

Calculated value  
passed back from the  
function to the calling  
code

# Diagramming a Simple Function Procedure

- Function Name
- Arguments/Parameters
- Function Statement Code
- Return Statement

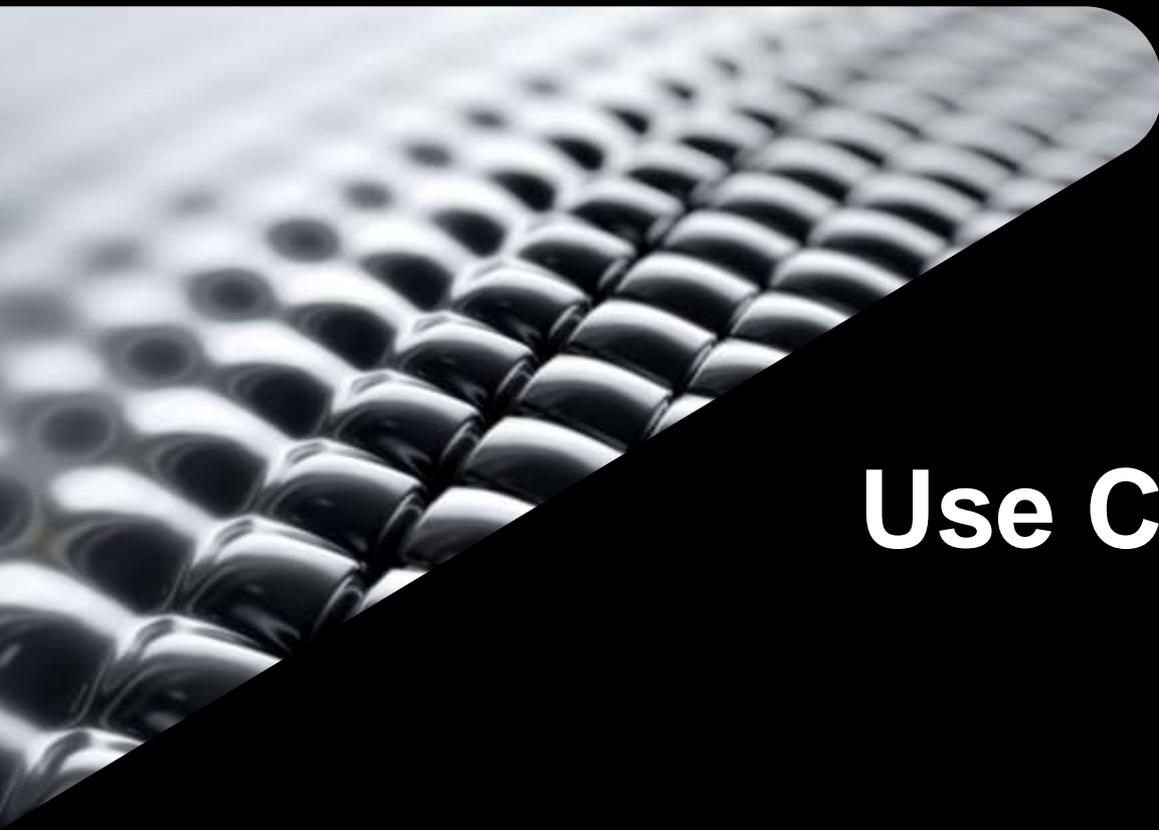
```
1 Sub Main
2
3   oThickness = Parameter("Thickness")
4   oHoleSize = Parameter("Diameter")
5
6   oOffset = CalculateOffset(oThickness, oHoleSize)
7   MessageBox.Show("Offset: " & oOffset, "iLogic")
8
9
10 End Sub
11
12 Function CalculateOffset(oThick As Double, oHole As Double)
13
14   If oThick < 3 Then
15     oShimSize = 0.32
16   ElseIf oThick < 5 Then
17     oShimSize = 0.41
18   Else
19     oShimSize = 0.533
20   End If
21
22   oOffset = oThick + oHole - oShimSize * 2
23
24   Return oOffset
25 End Function
```



# Why use Function Procedures ?

- Generally, we use functions to do some task or calculation, and return one or more values to the calling code.
- Often a Function is called multiple times within the same code but is passed different values each time.
- Allows us to reuse, rather than copy/paste the same code over and over

```
1 Sub Main
2
3     oThickness = Parameter("Thickness")
4     oHoleSize = Parameter("Diameter")
5
6     oOffset = CalculateOffset(oThickness, oHoleSize)
7
8     MessageBox.Show("Offset: " & oOffset, "iLogic")
9
10 End Sub
11
12 Function CalculateOffset(oThick As Double, oHole As Double)
13
14     If oThick < 3 Then
15         oShimSize = 0.32
16     ElseIf oThick < 5 Then
17         oShimSize = 0.41
18     Else
19         oShimSize = 0.533
20     End If
21
22     oOffset = oThick + oHole - oShimSize * 2
23
24     Return oOffset
25 End Function
```



# Use Class ThisRule

Tip #6

# What is an iLogic rule Class ?

- If you add a **Sub Main** procedure to your iLogic rule, the rule immediately takes the form of a standard VB.net class.
- A class is just a blueprint for a data type.
- In our case the data type is an iLogic Rule.

```
Sub Main
    Dim oDoc As AssemblyDocument
    oDoc = ThisDoc.Document

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDoc.ComponentDefinition.Occurrences

        oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
        oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
        MsgBox("Bin Location: " & oBinNum, , oDesc)

    Next
End Sub
```

# What is an iLogic rule Class ?

- By Default, the **Class** and **End Class** statements are not visible.
- We can add them as needed

```
Class ThisRule
Sub Main
    Dim oDoc As AssemblyDocument
    oDoc = ThisDoc.Document

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDoc.ComponentDefinition.Occurrences

        oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
        oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
        MsgBox("Bin Location: " & oBinNum, , oDesc)

    Next
End Sub
End Class
```

# What is an iLogic rule Class ?

- One of the most common uses of this within an iLogic rule, is to allow us to use a **shared** variable to pass information between procedures without the need to do so with arguments.

```
Class ThisRule
Sub Main
    Dim oDoc As AssemblyDocument
    oDoc = ThisDoc.Document

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDoc.ComponentDefinition.Occurrences

        oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
        oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
        MsgBox("Bin Location: " & oBinNum, , oDesc)

    Next
End Sub
End Class
```

# Using “Class ThisRule”

```
1 Class ThisRule
2
3     Shared oOcc As ComponentOccurrence
4     Shared oBinNum As String
5
6 Sub Main
7
8     Dim oDoc As AssemblyDocument
9     oDoc = ThisDoc.Document
10
11    For Each oOcc In oDoc.ComponentDefinition.Occurrences
12
13        'call the sub routine to get bin number
14        Call GetBinNumber
15
16        oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
17
18        MsgBox("Bin Location: " & oBinNum,, oDesc)
19    Next
20 End Sub
21
22
23 Sub GetBinNumber
24
25     oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
26
27 End Sub
28
29 End Class
```

Value is held at the rule level

Occurrence is held at the rule level

```
Sub Main
    Dim oDoc As AssemblyDocument
    oDoc = ThisDoc.Document

    Dim oOcc As ComponentOccurrence
    For Each oOcc In oDoc.ComponentDefinition.Occurrences

        'call the sub routine to get bin number
        Call GetBinNumber(oOcc)

    Next
End Sub

Sub GetBinNumber(oOcc As ComponentOccurrence)

    oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
    oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
    MsgBox("Bin Location: " & oBinNum,, oDesc)

End Sub
```

Occurrence passed to sub as an argument, and the iProperty value is held in the sub

# Example

- Here a variable called oDocName is captured at the beginning of the Main sub
- And then used over and over by the Sub and Function procedures

## Be Aware:

- Shared variable values at the class level persist in rule for until the file is closed

```
1 Class ThisRule
2
3 Shared oDocName As String
4
5 Sub Main
6
7     oDocName = ThisDoc.FileName
8
9     Call GetPartNumber
10    Call GetRelatedComps
11    Call CalcOffset
12
13    Call GetPartNumber
14    Call GetRelatedComps
15    Call CalcOffset
16
17    Call GetPartNumber
18    Call GetRelatedComps
19    Call CalcOffset
20
21 End Sub
22
23 Sub GetPartNumber...
26
27 Sub GetRelatedComps...
30
31 Function CalcOffset...
34
35 End Class
```

# Note that you don't need to call it ThisRule

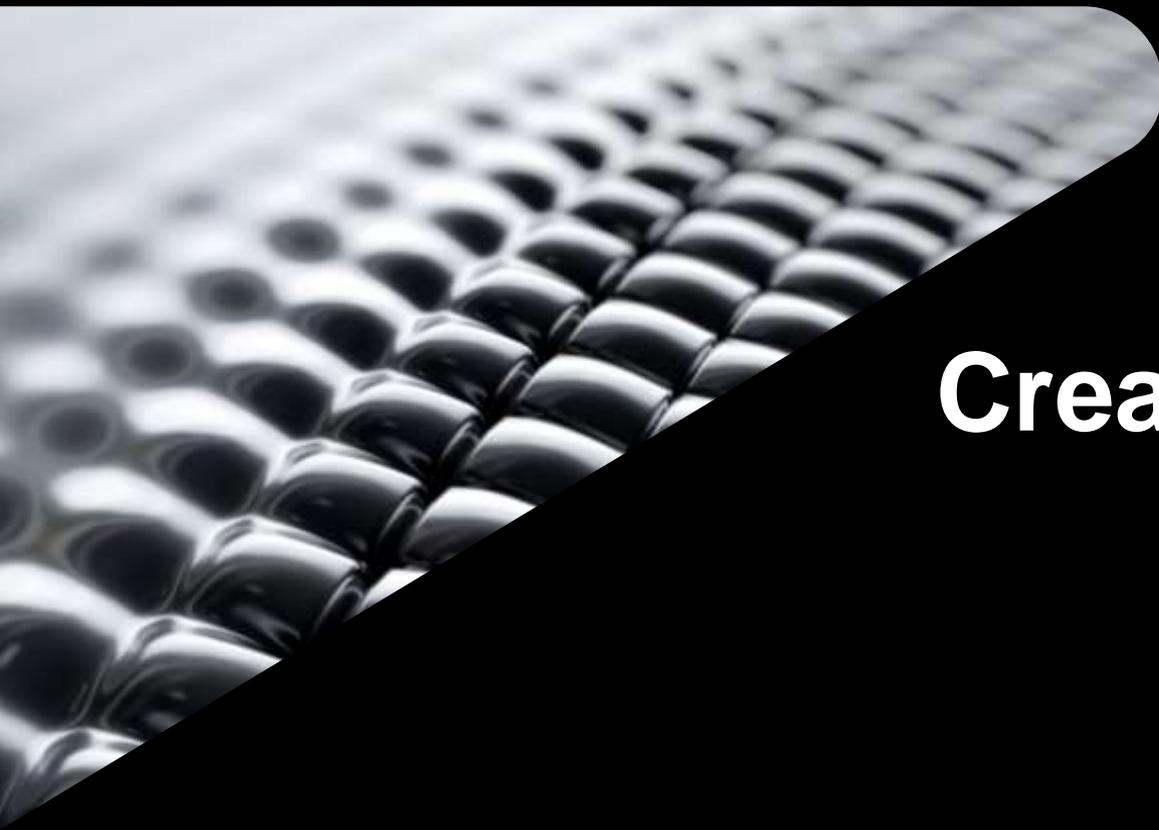
- We typically just call the class ThisRule
- But it can be named anything you like
- Both examples work the same:



```
1 Class ThisRule
2
3     Dim oOcc As ComponentOccurrence
4     Dim oBinNum As String
5
6 Sub Main
7
8     Dim oDoc As AssemblyDocument
9     oDoc = ThisDoc.Document
10
```



```
1 Class Carl_The_ilogic_Rule
2
3     Dim oOcc As ComponentOccurrence
4     Dim oBinNum As String
5
6 Sub Main
7
8     Dim oDoc As AssemblyDocument
9     oDoc = ThisDoc.Document
10
```



# Create collapsible code groups

Tip #7

# Create collapsible code groups

- The syntax is simply, apostrophe & an opening square bracket

' [

- Followed by, apostrophe & a closing square bracket:

']

```
7  [ Get iProps
8      oPN = iProperties.Value(oOcc.Name, "Proj
9      oRev = iProperties.Value(oOcc.Name, "Pro
10     oDesc = iProperties.Value(oOcc.Name, "Pr
11     oTitle = iProperties.Value(oOcc.Name, "S
12     oBinNum = iProperties.Value(oOcc.Name, '
13     ']
```

# The same rule with outlining collapsed

```
1
2 Dim oDoc As AssemblyDocument
3 oDoc = ThisDoc.Document
4
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     '[ Get iProps
8     oPN = iProperties.Value(oOcc.Name, "Project", "Part Number")
9     oRev = iProperties.Value(oOcc.Name, "Project", "Revision Number")
10    oDesc = iProperties.Value(oOcc.Name, "Project", "Description")
11    oTitle = iProperties.Value(oOcc.Name, "Summary", "Title")
12    oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")
13    ' ]
14
15    '[ Write to logger
16    Logger.Info("Part Number: " & oPN)
17    Logger.Info("Revision Number: " & oRev)
18    Logger.Info("Description: " & oDesc)
19    Logger.Info("oTitle: " & oTitle)
20    Logger.Info("Bin Location: " & oBinNum)
21    Logger.Info("")
22    ' ]
23 Next
```

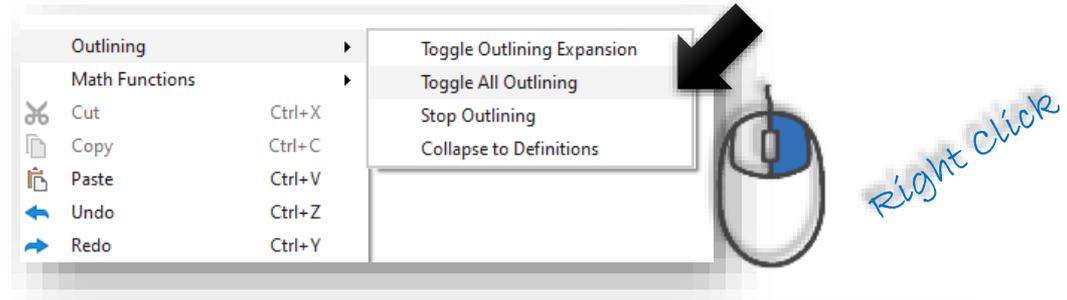
```
1
2 Dim oDoc As AssemblyDocument
3 oDoc = ThisDoc.Document
4
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     [ + ] Get iProps
14
15     [ + ] Write to logger
23 Next
```

You can collapse or expand the groups individually by clicking the

+ or -

# Use the Right Click Outlining Options

- These code groups can be collapsed using the right click Outlining options



```
1
2 Dim oDoc As AssemblyDocument
3 oDoc = ThisDoc.Document
4
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7   ⊕ Get iProps
14
15   ⊕ Write to logger
23 Next
24
```

# Note the difference in outlining using Sub Main()

Sub Main ()



Sub Main

```
1 Sub Main()  
2  
3 Dim oDoc As AssemblyDocument  
4 oDoc = ThisDoc.Document  
5  
6 For Each oOcc In oDoc.ComponentDefinition.Occurrences  
7  
8 + Get iProps  
15  
16 + Write to logger  
24 Next  
25  
26 End Sub
```

With parenthesis, no outlining for  
Main sub

```
1 Sub Main  
2  
3 Dim oDoc As AssemblyDocument  
4 oDoc = ThisDoc.Document  
5  
6 For Each oOcc In oDoc.ComponentDefinition.Occurrences  
7  
8 + Get iProps  
15  
16 + Write to logger  
24 Next  
25  
26 End Sub
```

Without parenthesis, outlining included for  
Main sub

A background image showing a close-up, perspective view of a woven mesh or fabric texture, rendered in grayscale. The mesh is composed of many small, rounded, interconnected elements, creating a grid-like pattern that recedes into the distance.

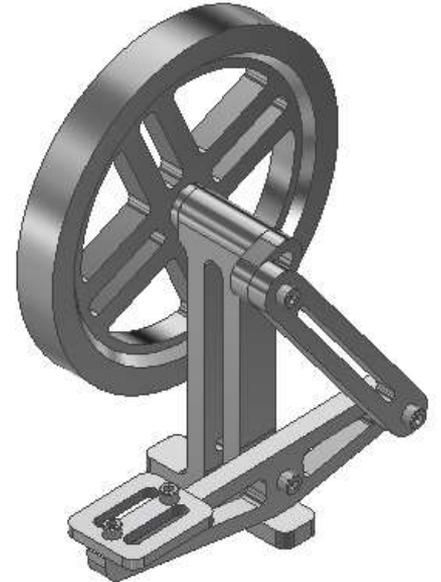
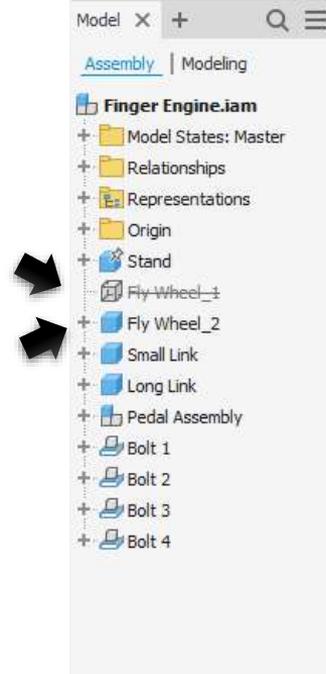
# iLogic Place and Constrain vs Suppress/Unsuppress

Tip #8

# Configuring Active and Inactive Components

## TRADITIONAL ILOGIC APPROACH

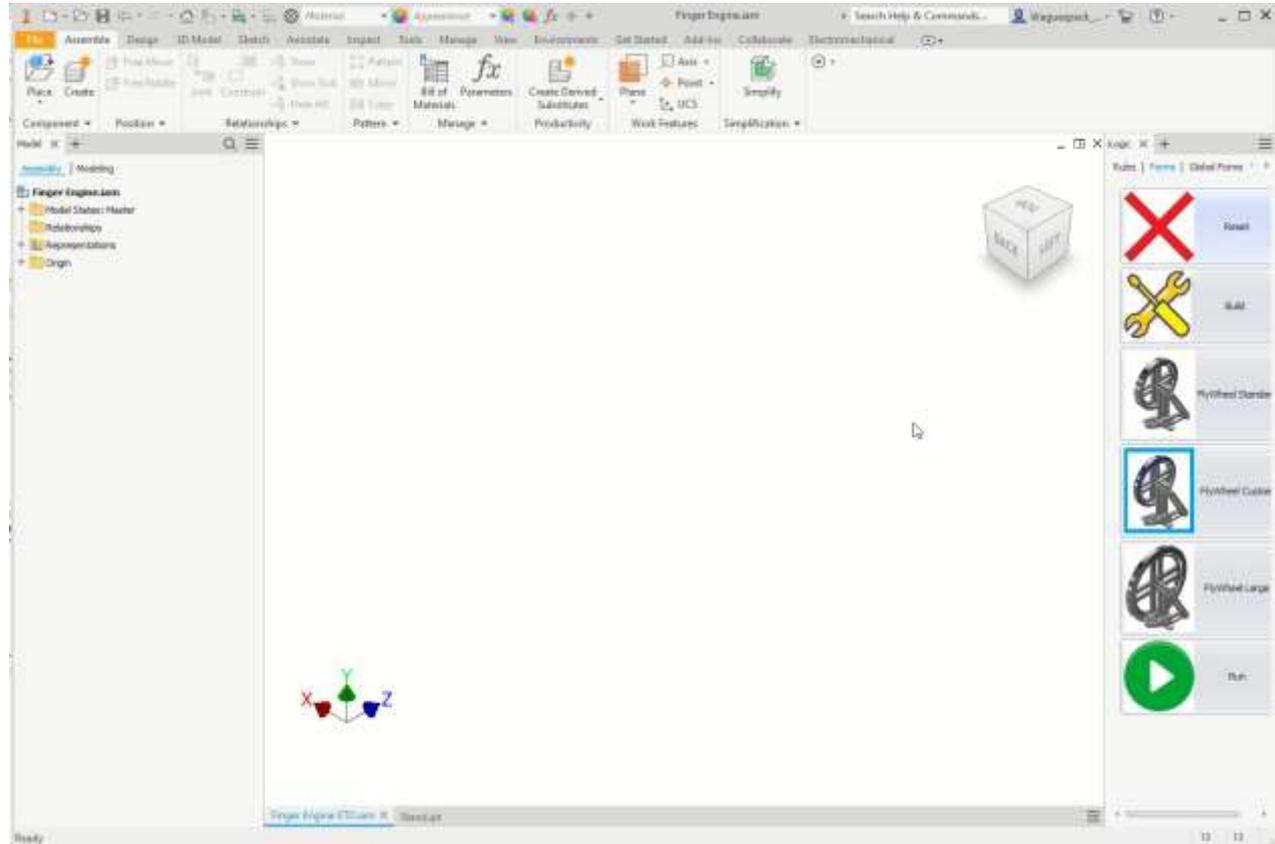
- In the past our primary mode of operation withing in iLogic was to create configurations by placing all of the components in an assembly, and then making some active and some inactive in order to configure
- This created some extra overhead, as far as dealing with Level of Details, Bill of Materials, Drawings, etc.



# Place and Constrain

## NEW ILOGIC APPROACH

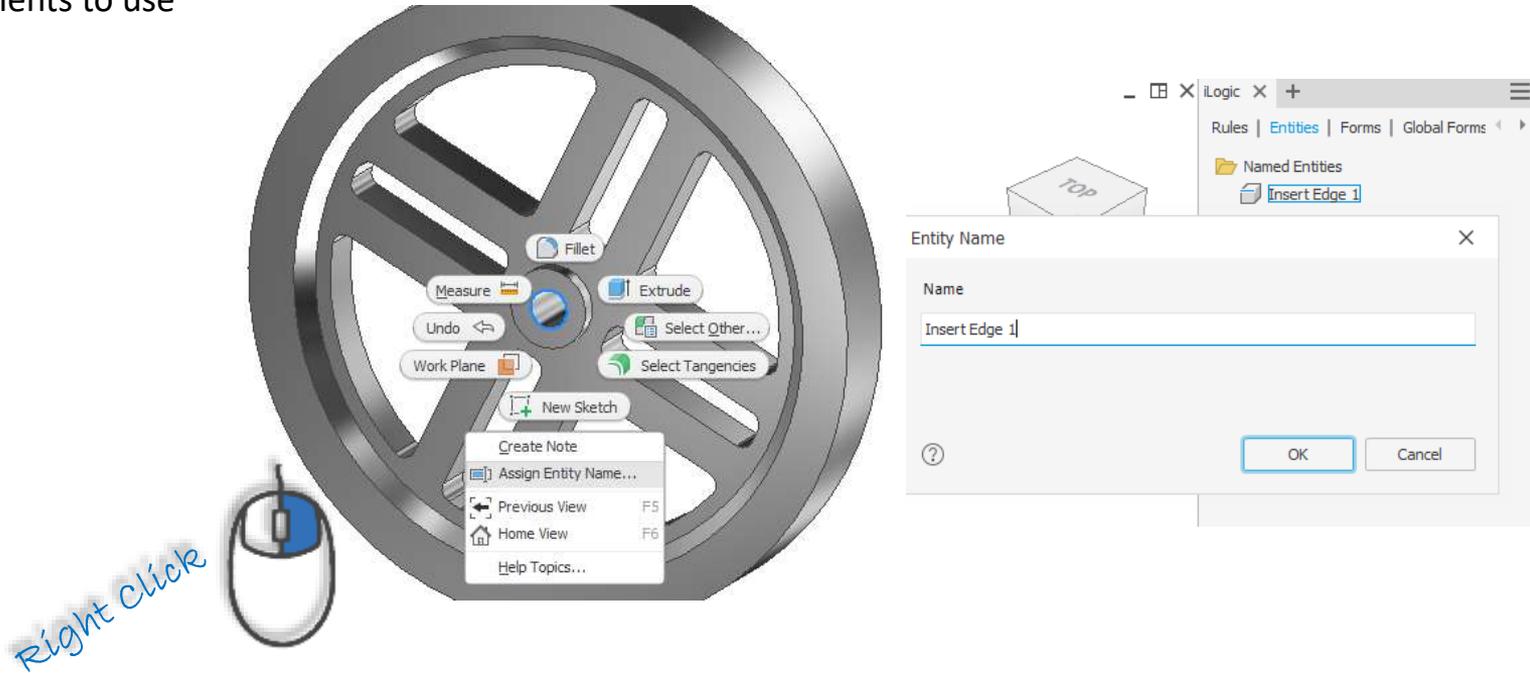
- Inventor 2019 introduced the ability to add components and constraints quickly and easily, by extracting the code from an assembly.
- This allows us to start with an empty template assembly and place only the components needed.
- Or swap out components as needed
- The advantage is there are no need to manage suppressed or invisible components for drawings, Bill of Materials, etc.



# Place and Constrain

## NEW ILOGIC APPROACH

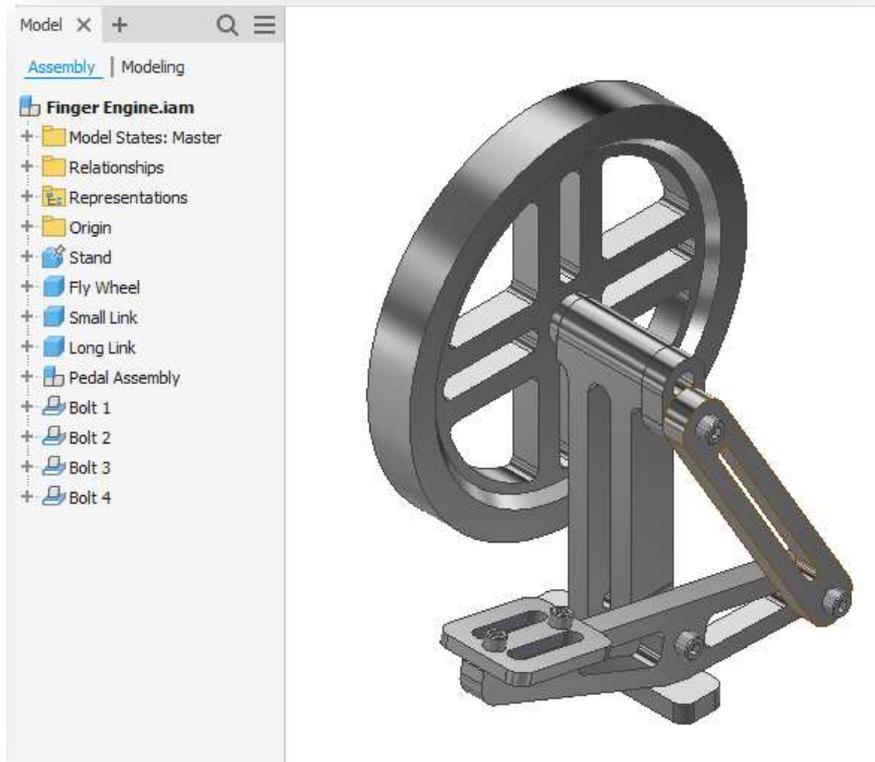
1. Create Named Geometry Entities in the components to use for constraints



# Place and Constrain

## NEW ILOGIC APPROACH

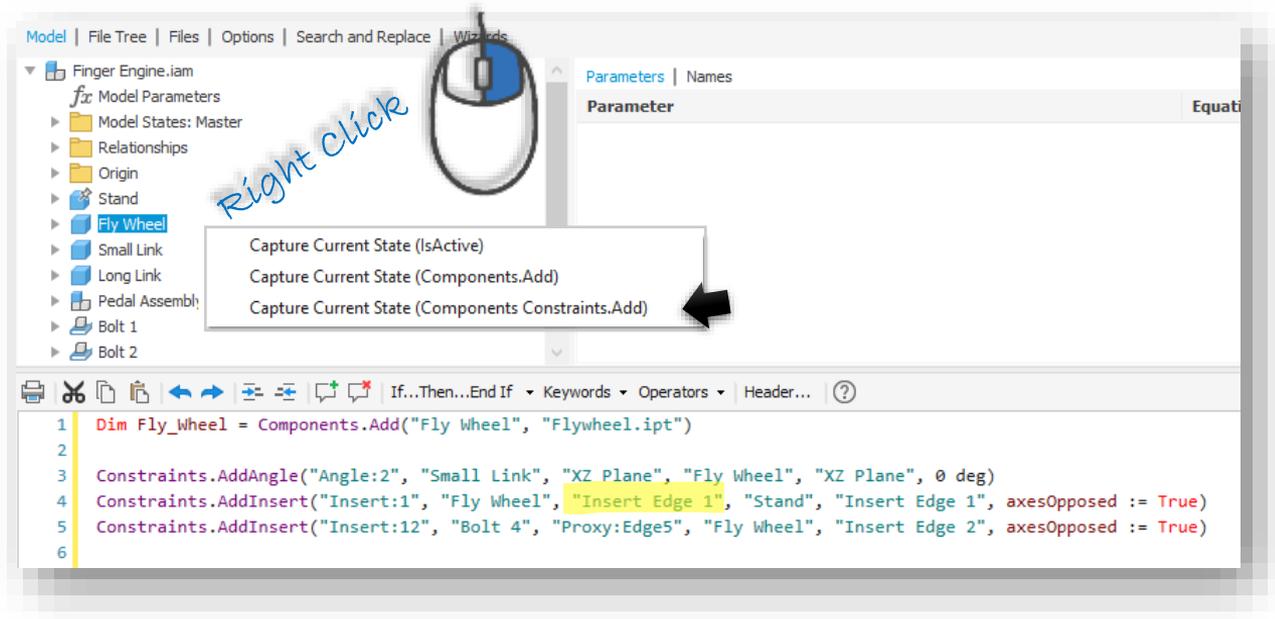
1. Create Named Geometry Entities in the components to use for constraints
2. Create and constrain your assembly



# NEW ILOGIC APPROACH

## Extract the Code

1. Create Named Geometry Entities in the components to use for constraints
2. Create and constrain your assembly
3. Extract the code from your fully assembled model

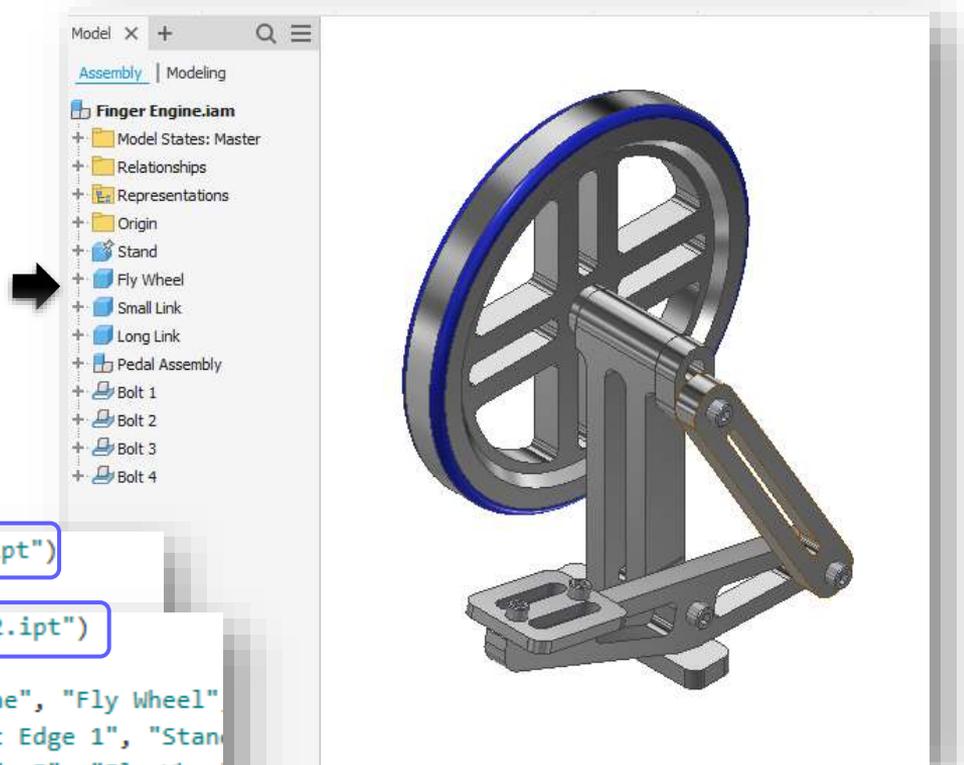


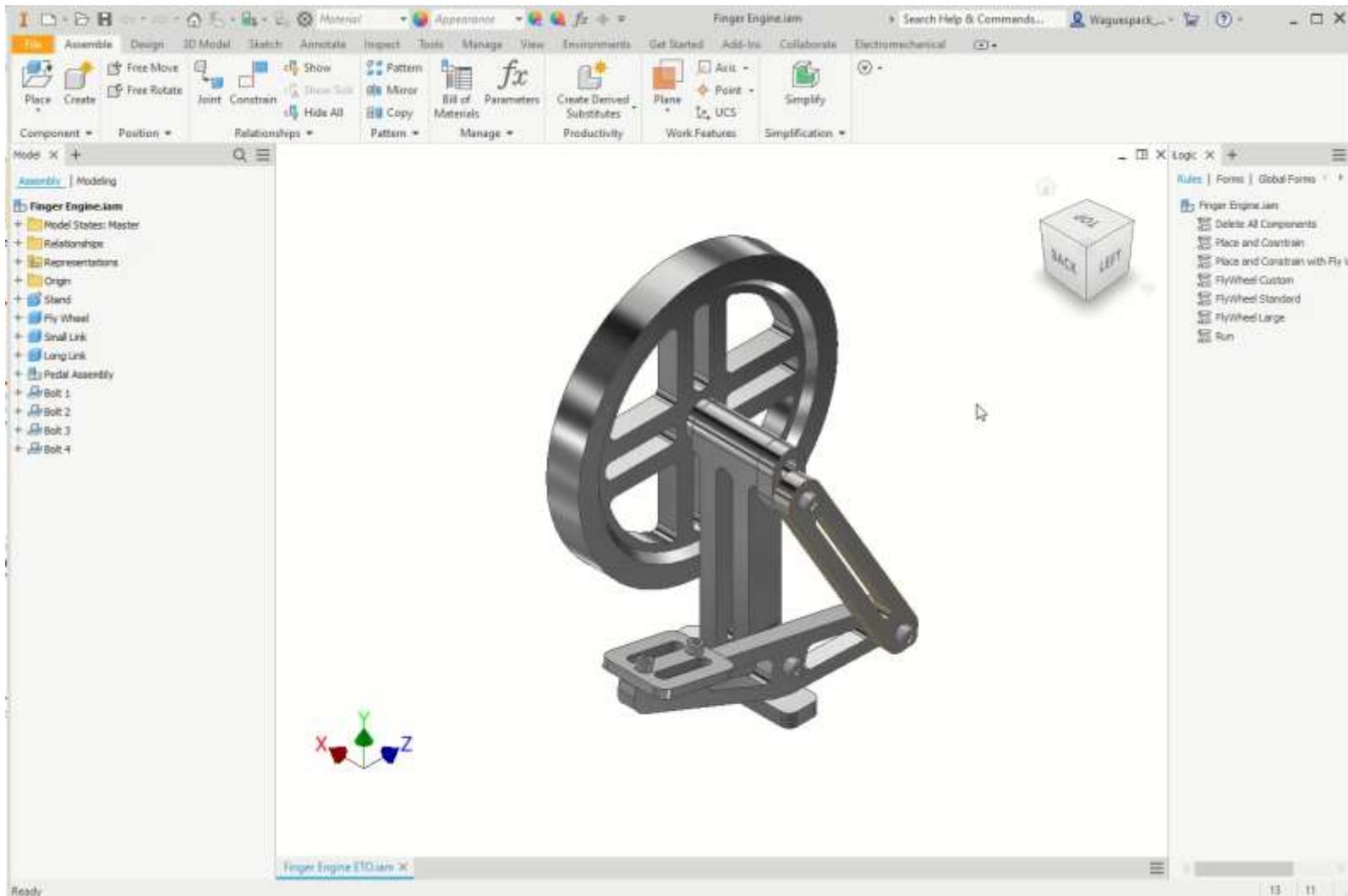
# Configure the rule

## NEW ILOGIC APPROACH

1. Create Named Geometry Entities in the components to use for constraints
2. Create and constrain your assembly
3. Extract the code from your fully assembled model
4. Configure rules to swap out component files as needed.

```
1 Dim Fly_Wheel = Components.Add("Fly Wheel", "Flywheel.ipt")
2
3 Dim Fly_Wheel = Components.Add("Fly Wheel", "Flywheel2.ipt")
4
5 Constraints.AddAngle("Angle:2", "Small Link", "XZ Plane", "Fly Wheel")
6 Constraints.AddInsert("Insert:1", "Fly Wheel", "Insert Edge 1", "Stand")
7 Constraints.AddInsert("Insert:12", "Bolt 4", "Proxy:Edge5", "Fly Wheel")
```







# Plan your code

Tip #9



# How to plan our iLogic rules

- Write out some of it, such as just a single size of the configuration to **get things working**
- Often this involves a lot of **copy/paste**, which creates repetitive lines
- Stop and look for a **pattern**
- **Recognize** this as an opportunity

```
1 oTrigger = Height
2 If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then
3     Parameter("Size") = 4
4     Parameter("Count") = 3
5
6     Component.Visible("NG4 AssemblyLH:1") = True
7     oCompOcc = Component.InventorComponent("NG4 AssemblyLH:1")
8     oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
9     Component.Visible("NG4 AssemblyRH:1") = True
10    oCompOcc = Component.InventorComponent("NG4 AssemblyRH:1")
11    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
12    Constraint.IsActive("NG4 _Mate") = True
13
14    Component.Visible("NG5 AssemblyLH:1") = False
15    oCompOcc = Component.InventorComponent("NG5 AssemblyLH:1")
16    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
17    Component.Visible("NG5 AssemblyRH:1") = False
18    oCompOcc = Component.InventorComponent("NG5 AssemblyRH:1")
19    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
20    Constraint.IsActive("NG5 _Mate") = False
21
22    Component.Visible("NG6 AssemblyLH:1") = False
23    oCompOcc = Component.InventorComponent("NG6 AssemblyLH:1")
```

# Evaluate and outline what you have

```
1 oTrigger = Height
2 If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then
3     Parameter("Size") = 4
4     Parameter("Count") = 3
5
6     Component.Visible("NG4 AssemblyLH:1") = True
7     oCompOcc = Component.InventorComponent("NG4 AssemblyLH:1")
8     oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
9     Component.Visible("NG4 AssemblyRH:1") = True
10    oCompOcc = Component.InventorComponent("NG4 AssemblyRH:1")
11    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
12    Constraint.IsActive("NG4 _Mate") = True
13
14    Component.Visible("NG5 AssemblyLH:1") = False
15    oCompOcc = Component.InventorComponent("NG5 AssemblyLH:1")
16    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
17    Component.Visible("NG5 AssemblyRH:1") = False
18    oCompOcc = Component.InventorComponent("NG5 AssemblyRH:1")
19    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
20    Constraint.IsActive("NG5 _Mate") = False
21
22    Component.Visible("NG6 AssemblyLH:1") = False
23    oCompOcc = Component.InventorComponent("NG6 AssemblyLH:1")
```

20 instances of RH & 20 instances of LH assemblies (40 assemblies)

17 Height ranges (4 thru 20) ... 1 thru 3 are static

If Height >= (some number) and Height <= (some other number) Then

Size = 4

Count = 3

LH size 4 and RH size 4 Assemblies visible w/ default BOM structure,  
all others are invisible and reference

Else If Height >= (some number) and Height <= (some other number) Then

Size = 5

Count = 4

LH size 4 and RH size 4 Assemblies visible w/ default BOM structure,  
all others are invisible and reference

Else if...

140  
LIMIT

$$140 \times 17 = 2380$$

# Use your outline to write “pseudo” code

- Plan out your code structure
- Create Sub procedures and Functions to reuse code where we’re doing the same task for different components

Trigger = Height

~ Call Sub to set Size

~ {Loop i = 4 to 20,

~ Set Count = Size -1

~ BooleanVariable = i <= Parameter("Size")

~ Call Sub to set Visibility

~ Call Sub to set BOM structure

Next]

{Sub to set Size

If Height >= (X) and Height <= (Y) Then

Set Size

Else....

End Sub]

{Sub to set Visibility

~ Set visibility

~ Set constraint active

End Sub]

{Sub to set BOM structure

~ if component is visible, then default BOM,

Else reference BOM

End Sub]

```

1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set size
6   Call SetSize(oHeight)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     oBoolean = i <= Parameter("Size")
12
13     oObjectName = "NO" & i & " Assembly@1"
14     Call Set_Visible(oObjectName, oBoolean) 'step into subroutine to set visibility
15
16     oObjectName = "NO" & i & " Assembly@1"
17     Call Set_Visible(oObjectName, oBoolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
21
22 Sub Set_Visible(oObjectName As String, oBoolean As Boolean)
23
24   Component.InventorComponent(oObjectName).Visible = oBoolean
25   Constraint.IsActive("NO" & i & " Mate") = oBoolean
26
27   'step into subroutine to set BOP structure
28   Call BOPStructure(oObjectName)
29
30 End Sub
31
32 Sub BOPStructure(oObjectName As String)
33
34   If Component.Visible(oObjectName) = True Then
35     Component.InventorComponent(oObjectName).BOPStructure = _
36     BOPStructureEnum.kDefaultBOPStructure
37   ElseIf Component.Visible(oObjectName) = False Then
38     Component.InventorComponent(oObjectName).BOPStructure = _
39     BOPStructureEnum.kReferenceBOPStructure
40   End If
41
42 End Sub
43
44 Sub SetSize(oHeight As Integer)
45
46   If Parameter("Height") <= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
63   End If
64
65   ReleaseParametersOutput()
66   InventorVb.Document.GetData()
67 End Sub

```





```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     oBoolean = i <= Parameter("Size")
12
13     oOccName = "NG" & i & " AssemblyLH:1"
14     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
15
16     oOccName = "NG" & i & " AssemblyRH:1"
17     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
```



```
36 Component.InventorComponent(oOccName).BOPDoc
37 BOPStructureEnum.DefaultBOPStructure
38 ElseIf Component.Visible(oOccName) = False Then
39   Component.InventorComponent(oOccName).BOPDoc
40   BOPStructureEnum.AssembliesBOPStructure
41 End If
42 End Sub
43
44 Sub SetSize(oHeight As Integer)
45
46   If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
63 End If
64
65 RuleParametersOutput()
66 InventorVb.DocumentUpdate()
67 End Sub
```

```
Size") - 1
'step into subroutine to set visibility
'step into subroutine to set visibility
'step into subroutine to set visibility
Sub SetSize(oHeight As Integer)
If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
End If
RuleParametersOutput()
InventorVb.DocumentUpdate()
End Sub
```

```
Sub SetSize(oHeight As Integer)
If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
End If
RuleParametersOutput()
InventorVb.DocumentUpdate()
End Sub
```



```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     oBoolean = i <= Parameter("Size")
12
13     oOccName = "NG" & i & " AssemblyLH:1"
14     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
15
16     oOccName = "NG" & i & " AssemblyRH:1"
17     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
```

```
21
22
23 Sub Set_Visible(oOccName As String, oBoolean As Boolean)
24
25   Component.InventorComponent(oOccName).Visible = oBoolean
26   Constraint.IsActive("NG" & i & "_Mate") = oBoolean
27
28   'step into subroutine to set BOM Structure
29   Call BOMStructure(oOccName)
30 End Sub
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
```

```
Size) - 1
...
'step into subroutine to set visibility
...
'step into subroutine to set visibility
```

```
43
44 Sub SetSize(oHeight As Integer)
45
46   If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
63   End If
64
65   RuleParametersOutput()
66   InventorVb.DocumentUpdate()
67 End Sub
```

```
53   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
54   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
55   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
56   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
57   End If
58
59   RuleParametersOutput()
60   InventorVb.DocumentUpdate()
61 End Sub
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     oBoolean = i <= Parameter("Size")
12
13     oOccName = "NG" & i & " AssemblyLH:1"
14     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
15
16     oOccName = "NG" & i & " AssemblyRH:1"
17     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
```

```
21
22
23 Sub Set_Visible(oOccName As String, oBoolean As Boolean)
24
25   Component.InventorComponent(oOccName).Visible = oBoolean
26   Constraint.IsActive("NG" & i & "_Mate") = oBoolean
27
28   'step into subroutine to set BOM Structure
29   Call BOMStructure(oOccName)
30 End Sub
```



```
Size) - 1
'step into subroutine to set visibility
'step into subroutine to set visibility
```

```
43
44 Sub SetSize(oHeight As Integer)
45
46   If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
63   End If
64
65   RuleParametersOutput()
66   InventorVb.DocumentUpdate()
67 End Sub
```

```
53   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
54   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
55   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
56   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
57   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
58   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
59   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
60   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
61   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
62   End If
63
64   RuleParametersOutput()
65   InventorVb.DocumentUpdate()
66 End Sub
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
```

```
1 Sub Main
2
3   oTrigger = Height
4
5   'step into subroutine to set Size
6   Call SetSize(oHeight)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     oBoolean = i <= Parameter("Size")
12
13     oOccName = "NG" & i & " AssemblyLH:1"
14     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
15
16     oOccName = "NG" & i & " AssemblyRH:1"
17     Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
```

```
21
22
23 Sub Set_Visible(oOccName As String, oBoolean As Boolean)
24
25   Component.InventorComponent(oOccName).Visible = oBoolean
26   Constraint.IsActive("NG" & i & "_Mate") = oBoolean
27
28   'step into subroutine to set BOM Structure
29   Call BOMStructure(oOccName)
30 End Sub
```



```
32 Sub BOMStructure(oOccName As String)
33
34   If Component.Visible(oOccName) = True Then
35     Component.InventorComponent(oOccName).BOMStructure = _
36     BOMStructureEnum.kDefaultBOMStructure
37   ElseIf Component.Visible(oOccName) = False Then
38     Component.InventorComponent(oOccName).BOMStructure = _
39     BOMStructureEnum.kReferenceBOMStructure
40   End If
41
42 End Sub
```

```
43
44 Sub SetSize(oHeight As Integer)
45
46   If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") > 32 And Parameter("Height") <= 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") > 40 And Parameter("Height") <= 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") > 48 And Parameter("Height") <= 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") > 56 And Parameter("Height") <= 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") > 64 And Parameter("Height") <= 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") > 72 And Parameter("Height") <= 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") > 80 And Parameter("Height") <= 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") > 88 And Parameter("Height") <= 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") > 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
63   End If
64
65   RuleParametersOutput()
66   InventorVb.DocumentUpdate()
67 End Sub
```

```
68
69
70   If Parameter("Height") >= 96 And Parameter("Height") <= 104 Then : Parameter("Size") = 13
71   ElseIf Parameter("Height") > 104 And Parameter("Height") <= 112 Then : Parameter("Size") = 14
72   ElseIf Parameter("Height") > 112 And Parameter("Height") <= 120 Then : Parameter("Size") = 15
73   ElseIf Parameter("Height") > 120 And Parameter("Height") <= 128 Then : Parameter("Size") = 16
74   ElseIf Parameter("Height") > 128 And Parameter("Height") <= 136 Then : Parameter("Size") = 17
75   ElseIf Parameter("Height") > 136 And Parameter("Height") <= 144 Then : Parameter("Size") = 18
76   ElseIf Parameter("Height") > 144 And Parameter("Height") <= 152 Then : Parameter("Size") = 19
77   ElseIf Parameter("Height") > 152 And Parameter("Height") <= 160 Then : Parameter("Size") = 20
78   End If
79
80   RuleParametersOutput()
81   InventorVb.DocumentUpdate()
82 End Sub
```

20 instances of RH & 20 instances of LH assemblies (40 assemblies)  
17 Height ranges (4 thru 20) ... 1 thru 3 are static

If Height >= (some number) and Height <= (some other n)

Size = 4

Count = 3

LH size 4 and RH size 4 Assemblies visible w/  
all others are invisible and reference

Else If Height >= (some number) and Height <=

Size = 5

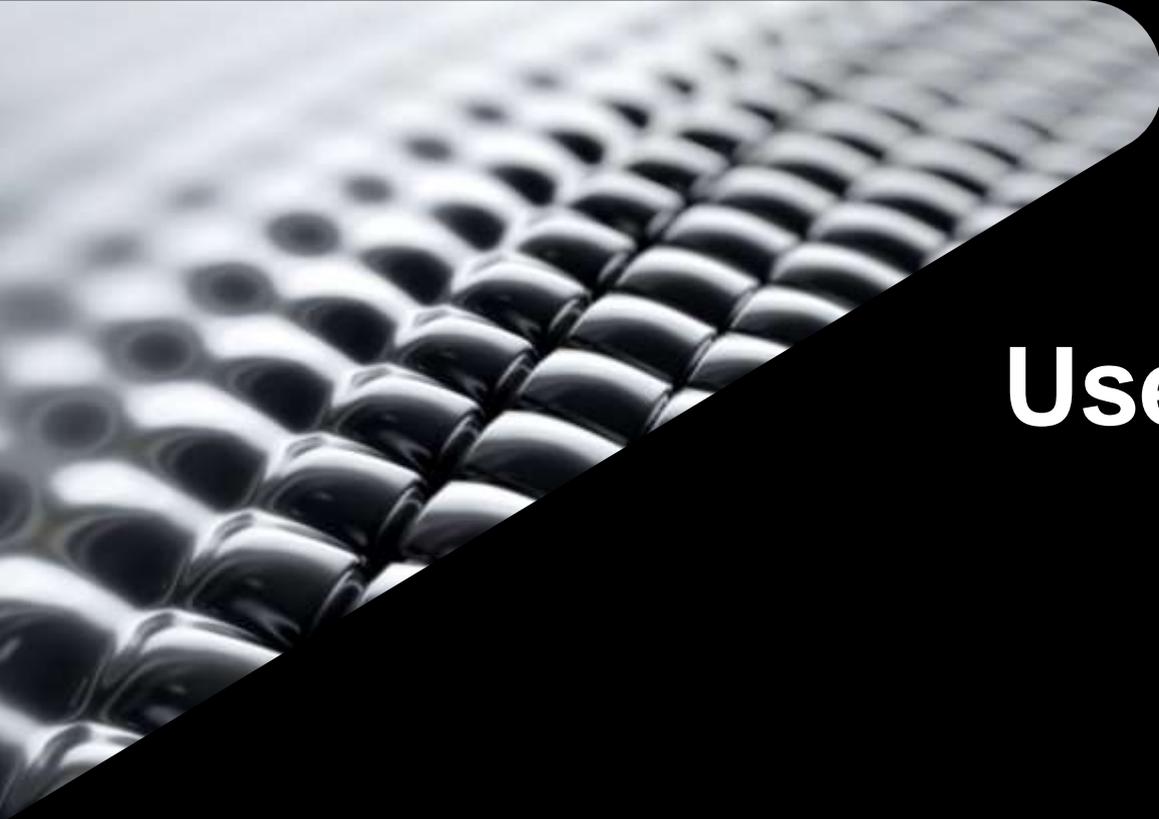
Count = 4

LH size 4 and RH size 4 Assemblies  
all others are invisible and

Else if...

Trigger = Height  
~ Call Sub to set Size  
~ {Loop i = 4 to 20,  
~ Set Count = Size - 1  
~ BooleanVariable = i <= Parameter("Size")  
~ Call Sub to set Visibility  
~ Call Sub to set BOM structure  
Next}  
[Sub to set Size  
If Height >= (X) and Height <= (Y) Then  
Set Size  
Else...  
End Sub]  
[Sub to set Visibility  
~ Set visibility  
~ Set constraint active  
End Sub]  
[Sub to set BOM structure  
~ if component is visible, then default BOM,  
Else reference BOM  
End Sub]

```
1 Sub Main
2
3   sTrigger = Height
4
5   'step into subroutine to set size
6   Call SetSize(Height)
7
8   For i = 4 To 20
9     Parameter("Count") = Parameter("Size") - 1
10
11     Boolean = i <= Parameter("Size")
12
13     oOcName = "HG" & i & " Assembly1"
14     Call Set_Visible(oOcName, Boolean) 'step into subroutine to set visibility
15
16     oOcName = "HG" & i & " AssemblyB1"
17     Call Set_Visible(oOcName, Boolean) 'step into subroutine to set visibility
18   Next
19
20 End Sub
21
22 Sub Set_Visible(oOcName As String, Boolean As Boolean)
23
24   Component.InventorComponent(oOcName).Visible = Boolean
25   Constraint.IsActive("HG" & i & " state") = Boolean
26
27   'step into subroutine to set BOM structure
28   Call BOMStructure(oOcName)
29
30 End Sub
31
32 Sub BOMStructure(oOcName As String)
33
34   If Component.Visible(oOcName) = True Then
35     Component.InventorComponent(oOcName).BOMStructure = _
36     BOMStructureEnum.DefaultBOMStructure
37   ElseIf Component.Visible(oOcName) = False Then
38     Component.InventorComponent(oOcName).BOMStructure = _
39     BOMStructureEnum.ReferenceBOMStructure
40   End If
41
42 End Sub
43
44 Sub SetSize(Height As Integer)
45
46   If Parameter("Height") = 24 And Parameter("Height") = 31 Then : Parameter("Size") = 4
47   ElseIf Parameter("Height") = 31 And Parameter("Height") = 40 Then : Parameter("Size") = 5
48   ElseIf Parameter("Height") = 40 And Parameter("Height") = 48 Then : Parameter("Size") = 6
49   ElseIf Parameter("Height") = 48 And Parameter("Height") = 56 Then : Parameter("Size") = 7
50   ElseIf Parameter("Height") = 56 And Parameter("Height") = 64 Then : Parameter("Size") = 8
51   ElseIf Parameter("Height") = 64 And Parameter("Height") = 72 Then : Parameter("Size") = 9
52   ElseIf Parameter("Height") = 72 And Parameter("Height") = 80 Then : Parameter("Size") = 10
53   ElseIf Parameter("Height") = 80 And Parameter("Height") = 88 Then : Parameter("Size") = 11
54   ElseIf Parameter("Height") = 88 And Parameter("Height") = 96 Then : Parameter("Size") = 12
55   ElseIf Parameter("Height") = 96 And Parameter("Height") = 104 Then : Parameter("Size") = 13
56   ElseIf Parameter("Height") = 104 And Parameter("Height") = 111 Then : Parameter("Size") = 14
57   ElseIf Parameter("Height") = 111 And Parameter("Height") = 118 Then : Parameter("Size") = 15
58   ElseIf Parameter("Height") = 118 And Parameter("Height") = 128 Then : Parameter("Size") = 16
59   ElseIf Parameter("Height") = 128 And Parameter("Height") = 136 Then : Parameter("Size") = 17
60   ElseIf Parameter("Height") = 136 And Parameter("Height") = 144 Then : Parameter("Size") = 18
61   ElseIf Parameter("Height") = 144 And Parameter("Height") = 151 Then : Parameter("Size") = 19
62   ElseIf Parameter("Height") = 151 And Parameter("Height") = 160 Then : Parameter("Size") = 20
63   End If
64
65   Call ParameterOutput()
66   InventorVB.Document.Update()
67 End Sub
```

A close-up, black and white photograph of a woven mesh texture, possibly a metal screen or fabric, with a diagonal line of perspective. The mesh is composed of small, rounded, interconnected elements that create a grid-like pattern. The lighting is dramatic, highlighting the texture and depth of the weave.

# Use Loops when possible

Tip #10

# Use Loops to eliminate repetitious lines

## Unoptimized rule

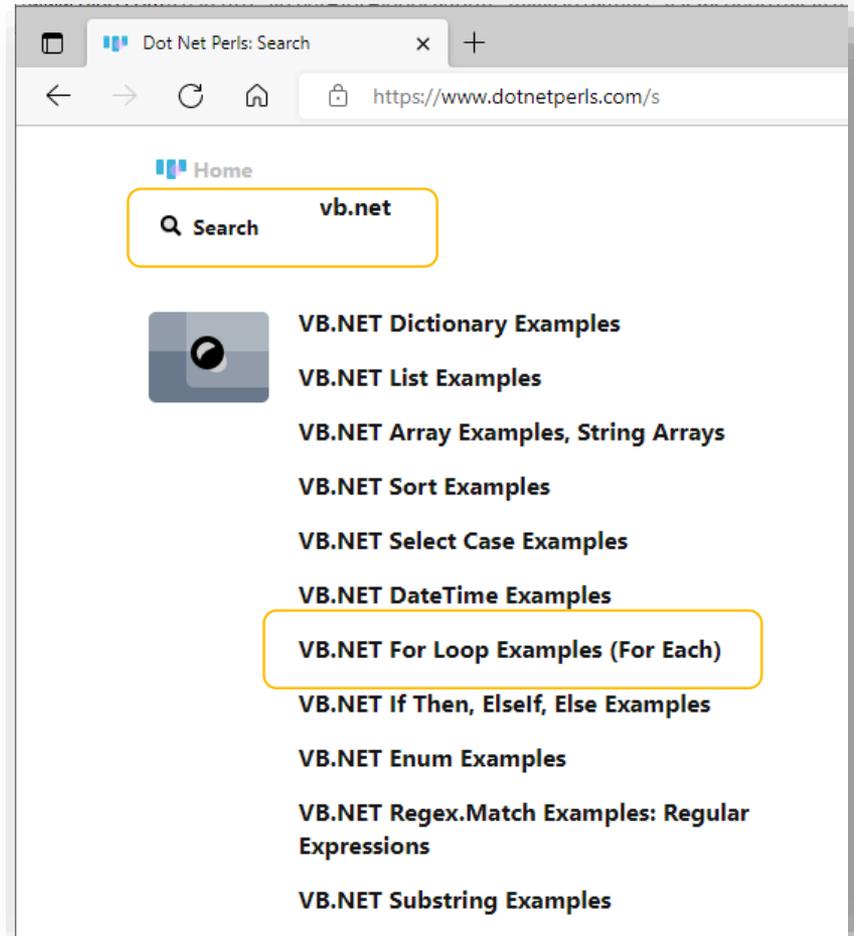
```
1 If Parameter("IsRightHand") True Then
2     Component.IsActive("RH_Brace") = True
3     Component.IsActive("RH_TopBracket") = True
4     Component.IsActive("RH_BottomBracket") = True
5     Component.IsActive("RH_Lid") = True
6     Component.IsActive("RH_Shim") = True
7
8     Component.IsActive("LH_Brace") = False
9     Component.IsActive("LH_TopBracket") = False
10    Component.IsActive("LH_BottomBracket") = False
11    Component.IsActive("LH_Lid") = False
12    Component.IsActive("LH_Shim") = False
13
14 Else
15     Component.IsActive("RH_Brace") = False
16     Component.IsActive("RH_TopBracket") = False
17     Component.IsActive("RH_BottomBracket") = False
18     Component.IsActive("RH_Lid") = False
19     Component.IsActive("RH_Shim") = False
20
21     Component.IsActive("LH_Brace") = True
22     Component.IsActive("LH_TopBracket") = True
23     Component.IsActive("LH_BottomBracket") = True
24     Component.IsActive("LH_Lid") = True
25     Component.IsActive("LH_Shim") = True
26 End If
```

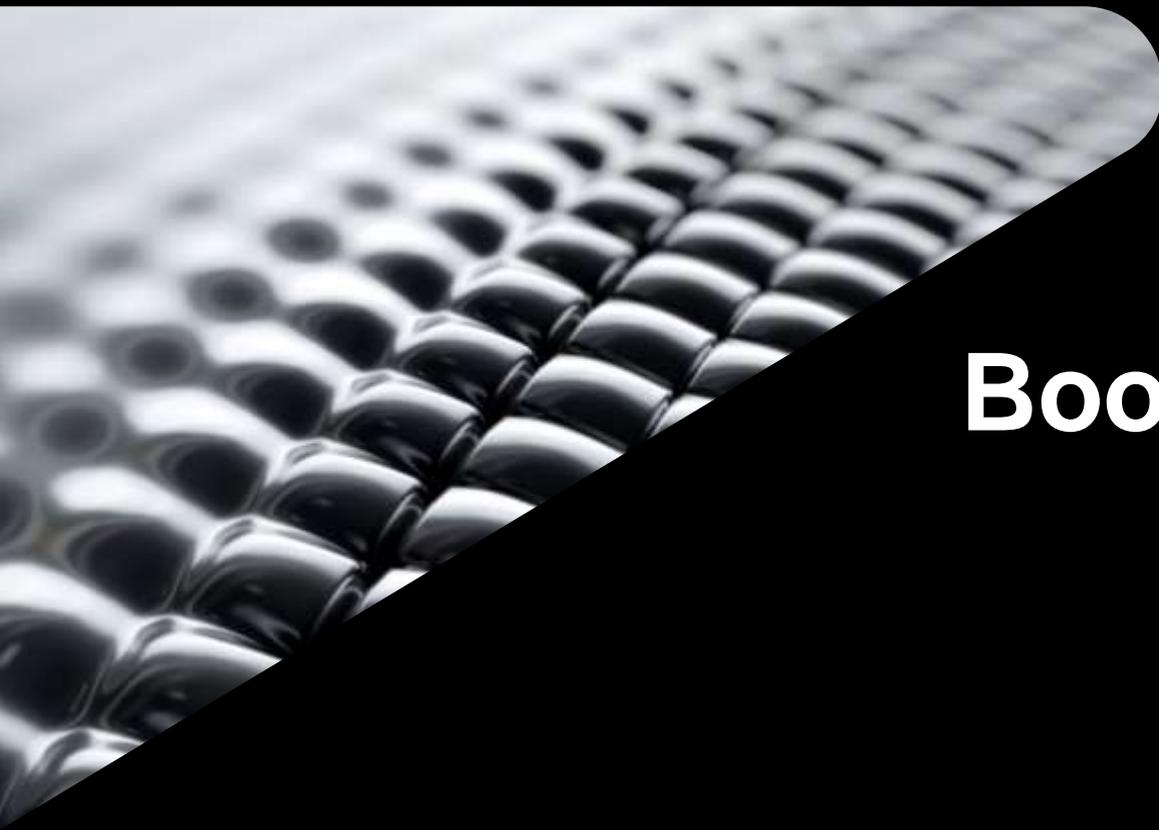
## Optimized rule using a For Each loop

```
1 Dim oDoc As AssemblyDocument
2 oDoc = ThisDoc.Document
3
4 Dim oOcc As ComponentOccurrence
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     If Left(oOcc.Name, 3) = "RH_" Then
8         Component.IsActive(oOcc.Name) = Parameter("IsRightHand")
9     ElseIf Left(oOcc.Name, 3) = "LH_" Then
10        Component.IsActive(oOcc.Name) = Not Parameter("IsRightHand")
11    End If
12 Next
```

# New to loops?

- Find examples online
- iLogic uses VB.net
- search online for something such as **“VB.net Loops”**
- Or **“VB.net For Each”**
- [www.dotnetperls.com](https://www.dotnetperls.com) is a site that has a lot of examples



A close-up, black and white photograph of a metal mesh or woven fabric, showing a grid of circular openings. The mesh is slightly out of focus in the background, creating a sense of depth. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# Booleans and the **Not** operator

Tip #11

# Boolean = True/False

 **Bool·e·an**  
*/ˈboʊləən/*

*adjective*

denoting a system of algebraic notation used to represent logical propositions, especially in computing and electronics.

*noun* **COMPUTING**

a binary variable, having two possible values called "true" and "false."

**Parameters**

	Parameter Name	Consumed by	Unit/Type	Equation	No
	Model Parameters				
	User Parameters				
	IsCustom		True/False	True	

# Skip the If/Then statements and use the **Not** operator

Parameters				
Parameter Name	Consumed by	Unit/Type	Equation	Ne
Model Parameters				
User Parameters				
IsCustom		True/False	True	

```
1 If Parameter("IsCustom") = True Then
2
3     Component.IsActive("Bolt 1") = True
4     Component.IsActive("Bolt 2") = True
5     Component.IsActive("Bolt 3") = False
6     Component.IsActive("Bolt 4") = False
7
8 Else
9
10    Component.IsActive("Bolt 1") = False
11    Component.IsActive("Bolt 2") = False
12    Component.IsActive("Bolt 3") = True
13    Component.IsActive("Bolt 4") = True
14
15 End If
```

```
1 Component.IsActive("Bolt 1") = Parameter("IsCustom")
2 Component.IsActive("Bolt 2") = Parameter("IsCustom")
3 Component.IsActive("Bolt 3") = Not Parameter("IsCustom")
4 Component.IsActive("Bolt 4") = Not Parameter("IsCustom")
```

Optimized rule using the Not Operator

Objective:

# Finding your way around the Inventor API

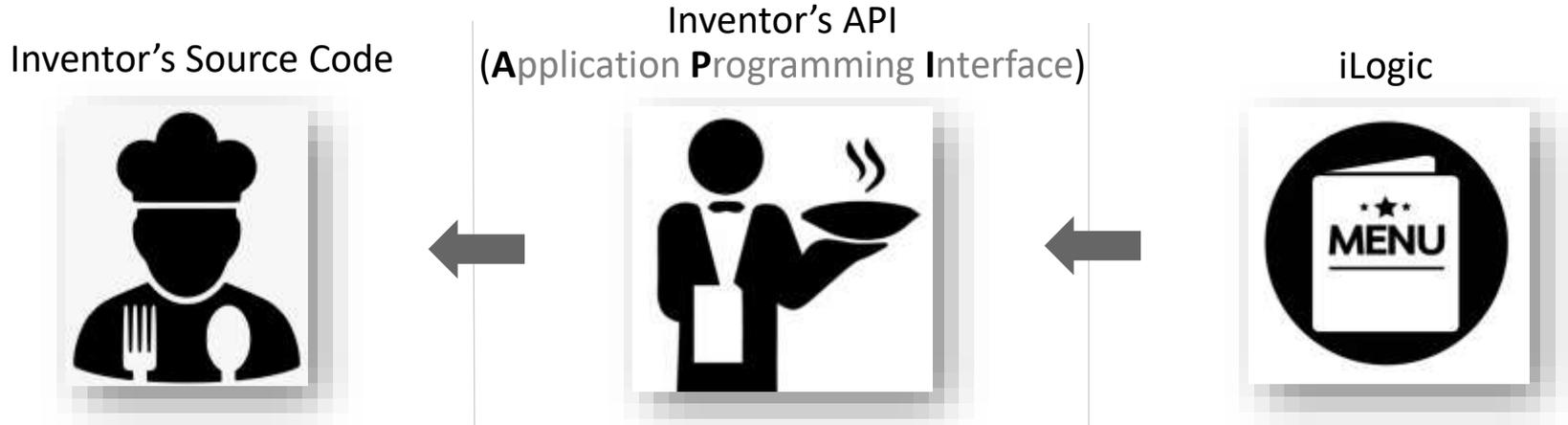
**API: Application Programming Interface**



# Understand the API

Tip #12

# An Analogy



- The chef knows how to create any dish, just as the Autodesk source code knows how to do anything in Inventor
  - Has access to all the recipes, ingredients, culinary tool and techniques
- We don't get to see the recipes, ingredients, culinary tool and techniques, and likewise we don't get to see Autodesk's source code
- We speak to the chef through the waiter, and we speak to Inventor's source code using the API
- iLogic is like the menu from which we read; in that it gives us a simplified subset of functions from which to choose

## API version

```
Dim oDoc As PartDocument
oDoc = ThisApplication.ActiveDocument

Dim oCustomPropertySet As PropertySet
oCustomPropertySet = oDoc.PropertySets.Item _
    ("Inventor User Defined Properties")

Try
    'try to set the iproperty
    oProp = oCustomPropertySet.Item("Rack Number")
Catch
    'catch error when iproperty doesn't exist and create it
    oProp = oCustomPropertySet.Add("Rack Number")
End Try

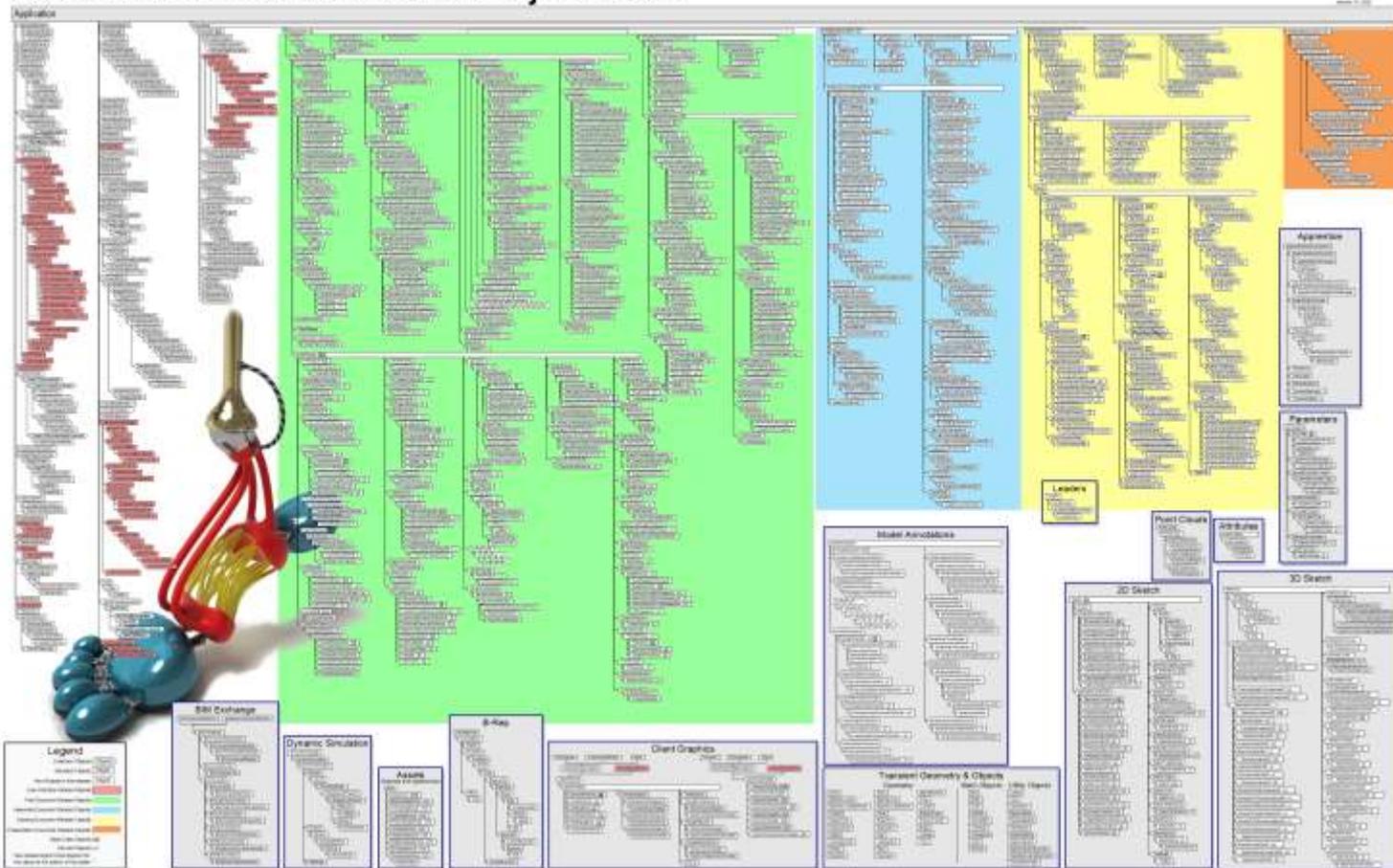
oProp.Value = "NG-4578-01"
```

Both do the  
exact same thing

## iLogic version

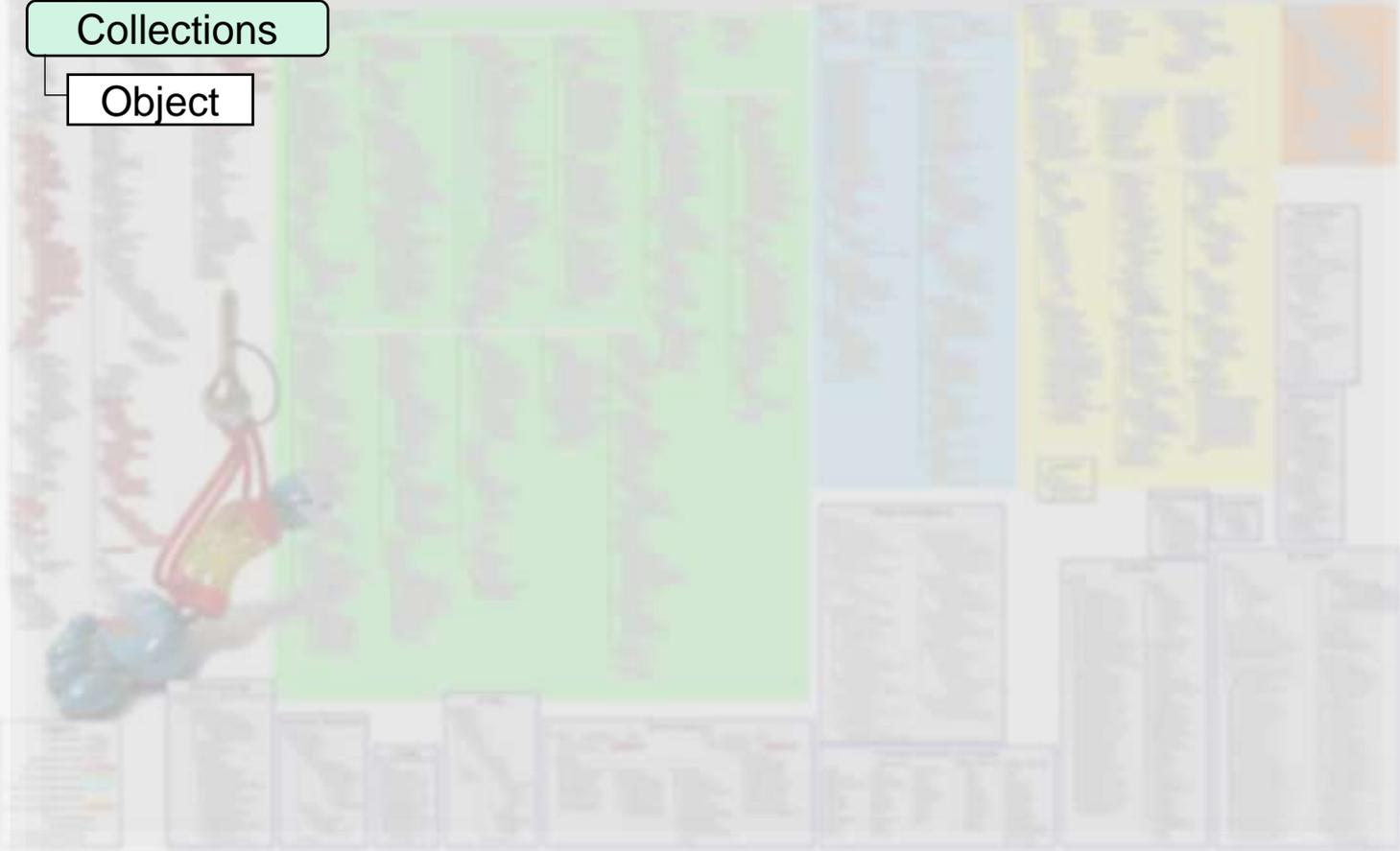
```
iProperties.Value("Custom", "Rack Number") = "NG-4578-01"
```

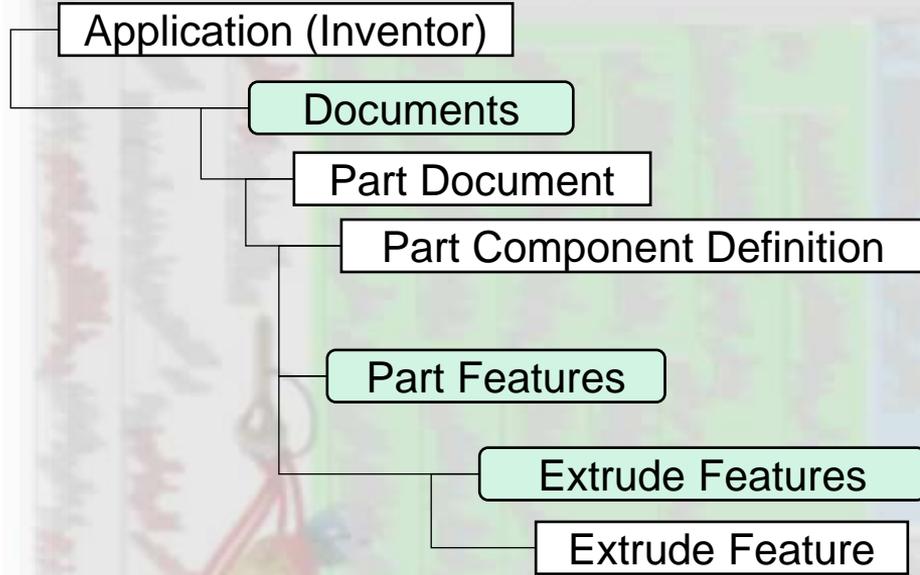
# AUTODESK® INVENTOR® 2023 API Object Model



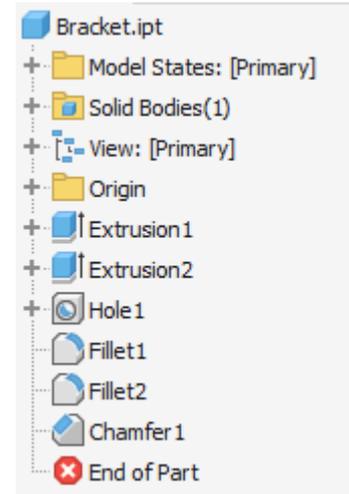
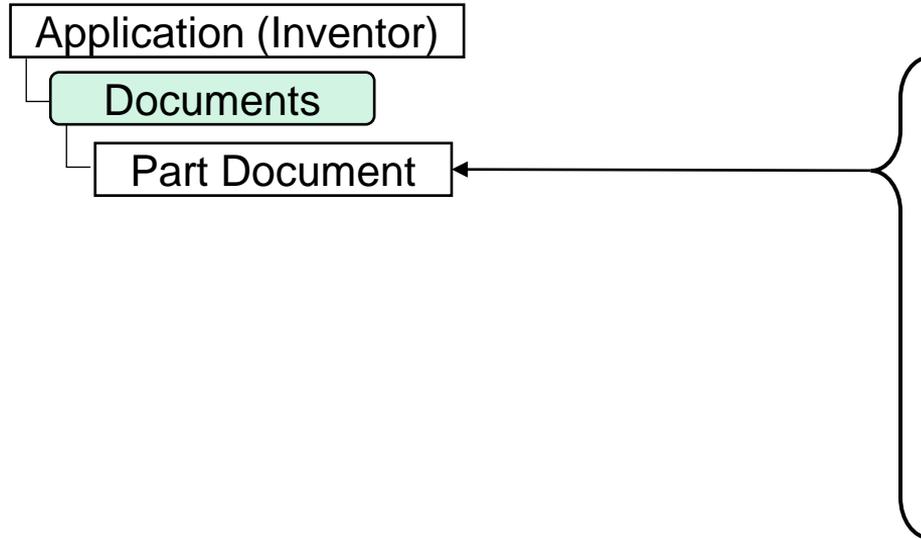
Collections

Object

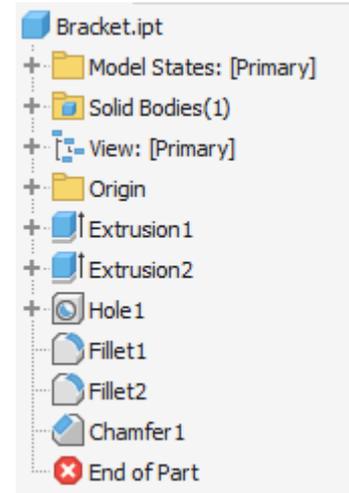
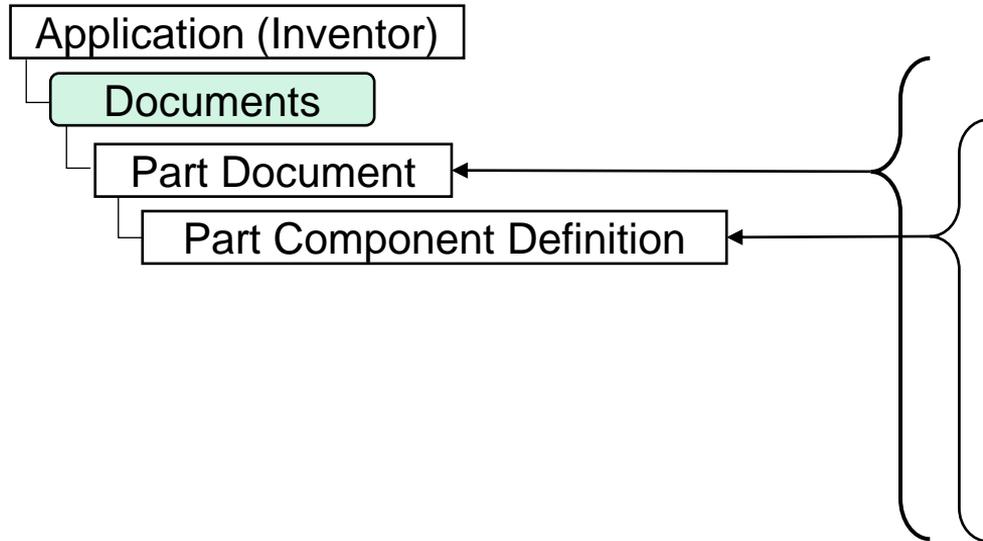




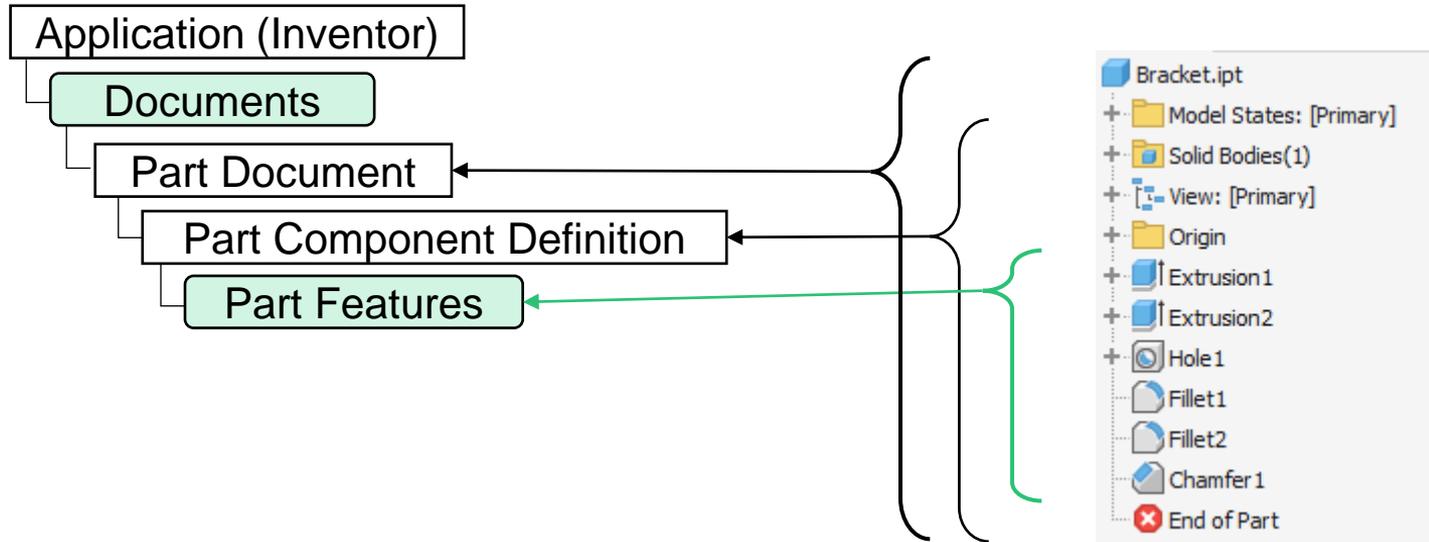
# API Object Model of a part



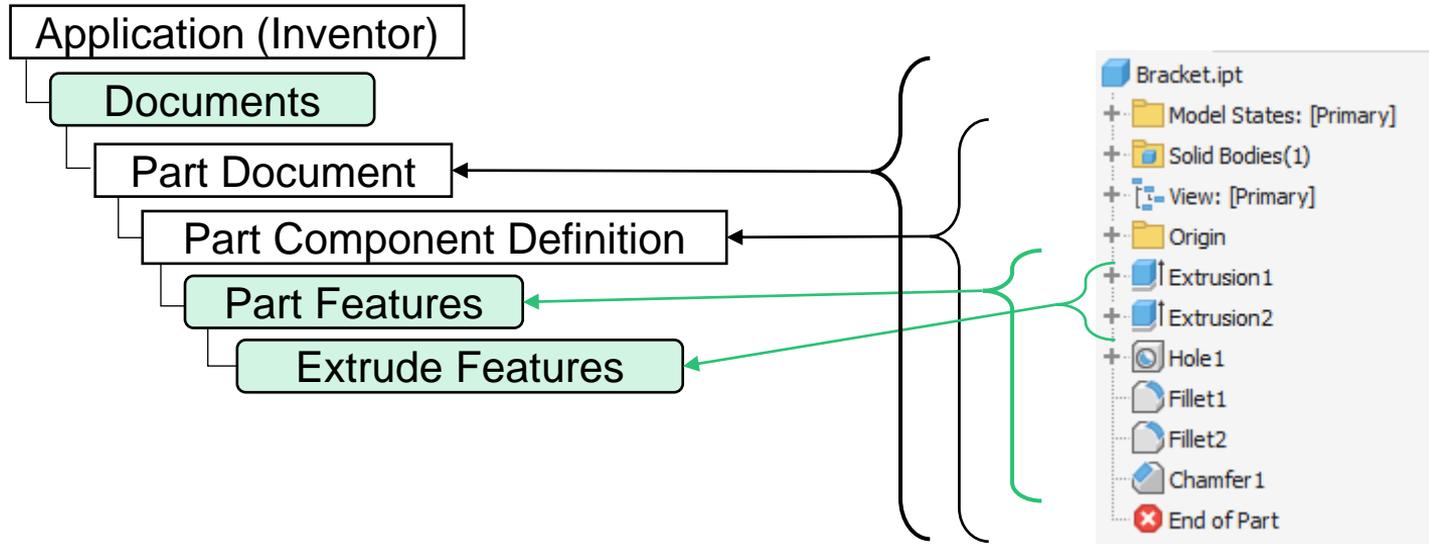
# API Object Model of a part



# API Object Model of a part

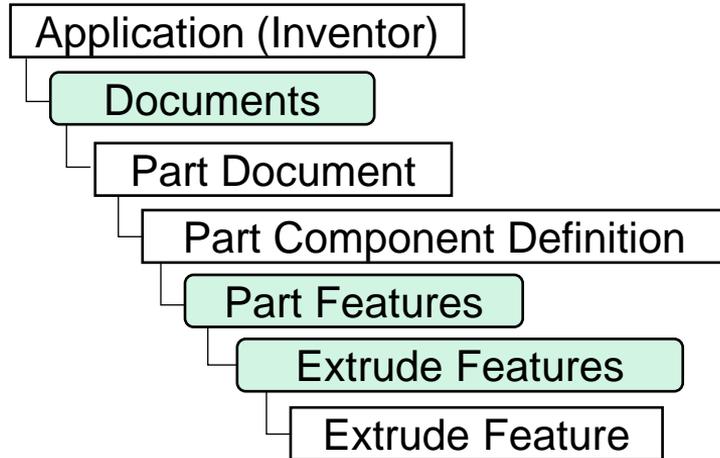


# API Object Model of a part



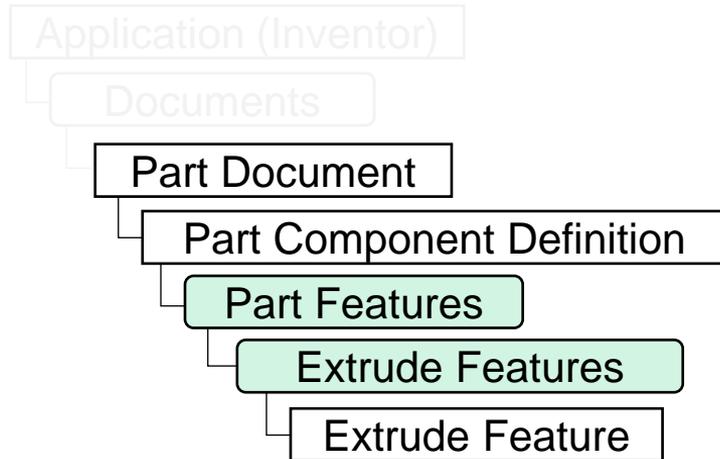


# Code Comparison



```
1 Dim oApp As Inventor.Application
2 oApp = ThisApplication
3
4 Dim oDocs As Documents
5 oDocs = oApp.Documents
6
7 Dim oDoc As PartDocument
8 oDoc = oDocs.VisibleDocuments.Item(1)
9
10 Dim oDefinition As ComponentDefinition
11 oDefinition = oDoc.ComponentDefinition
12
13 Dim oFeatures As PartFeatures
14 oFeatures = oDefinition.Features
15
16 Dim oExtrusions As ExtrudeFeatures
17 oExtrusions = oFeatures.ExtrudeFeatures
18
19 Dim oExtrude1 As ExtrudeFeature
20 oExtrude1 = oExtrusions.Item("Extrusion1")
```

# Using iLogic's **ThisDoc.Document** to get the document the rule is triggered from



```
1 Dim oDoc As PartDocument
2 oDoc = ThisDoc.Document
3
4 Dim oDefinition As ComponentDefinition
5 oDefinition = oDoc.ComponentDefinition
6
7 Dim oFeatures As PartFeatures
8 oFeatures = oDefinition.Features
9
10 Dim oExtrusions As ExtrudeFeatures
11 oExtrusions = oFeatures.ExtrudeFeatures
12
13 Dim oExtrude1 As ExtrudeFeature
14 oExtrude1 = oExtrusions.Item("Extrusion1")
```

# Using a single line to get the named Extrusion

```
oExtrude1 = ThisDoc.Document.ComponentDefinition.Features.ExtrudeFeatures.Item("Extrusion1")
```

Part Document

Part Component Definition

Part Document

Part Features

Definition

Part Features

Extrude Features

Extrude Feature

Extrude Features

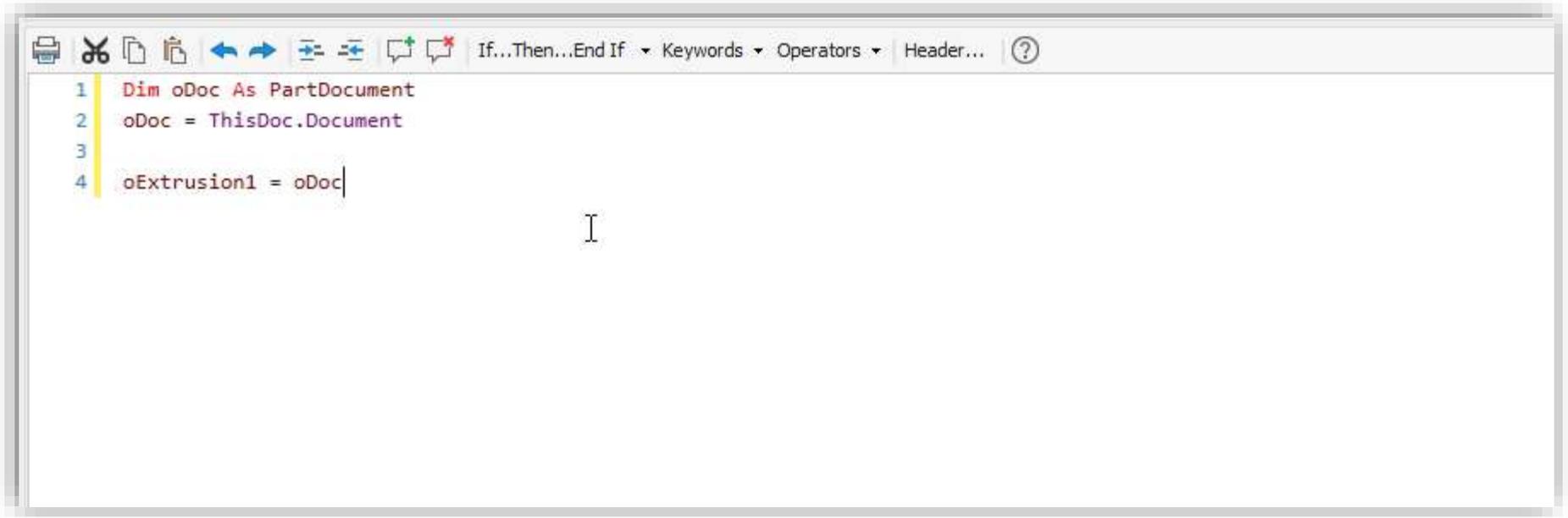
Extrude Feature

A close-up, black and white photograph of a metal mesh or woven fabric, showing a grid of circular openings. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# IntelliSense and Declared Variables

Tip #13

# Using IntelliSense to navigate the API



The screenshot shows a code editor window with a toolbar at the top containing icons for print, cut, copy, paste, undo, redo, and comments. The code is as follows:

```
1 Dim oDoc As PartDocument
2 oDoc = ThisDoc.Document
3
4 oExtrusion1 = oDoc|
```

The cursor is positioned at the end of the fourth line, and a vertical line is visible at the end of the text 'oDoc|', indicating that IntelliSense is active and ready to provide suggestions for the variable 'oDoc'.

# Why can't I find the API object I was expecting?



Document

Part Component Definition

Part Features

Extrude Features

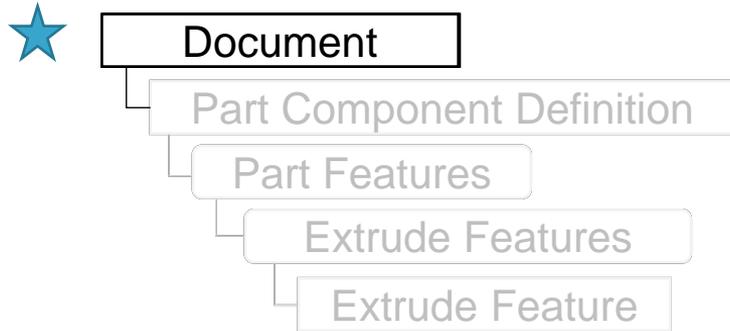
Extrude Feature



```
1 oDoc = ThisDoc.Document
2
3 oExtrusion1 = |
```

ComponentDefinition ?

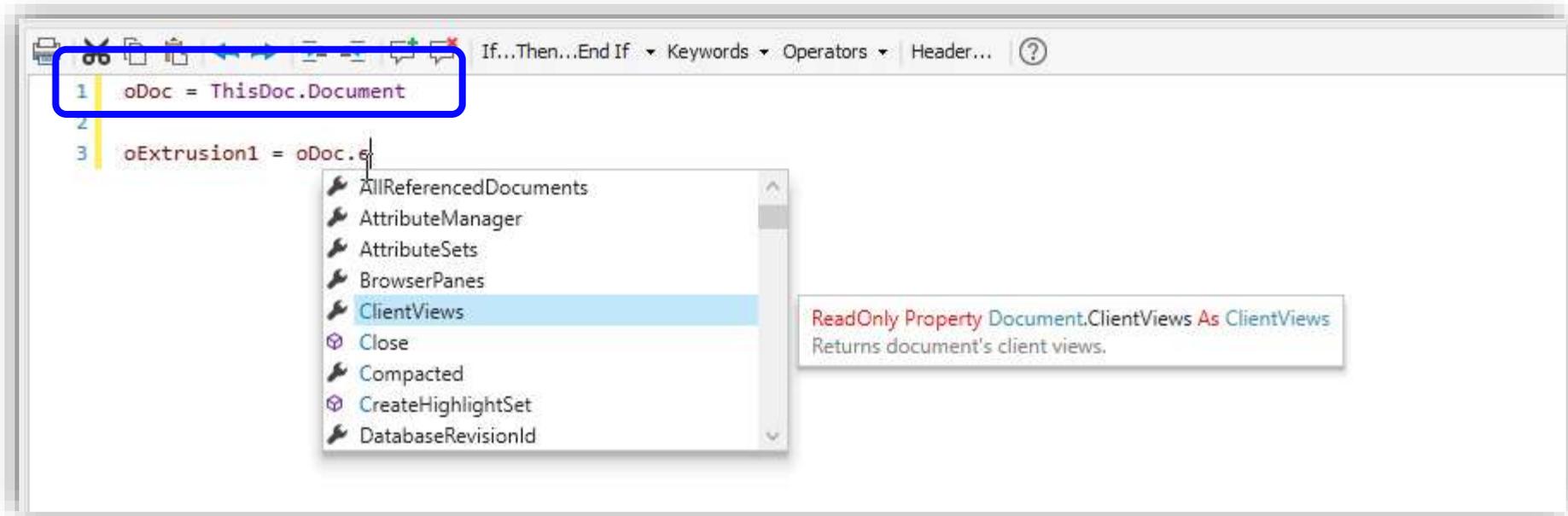
# Solution: Declare the variable



```
oDoc = ThisDoc.Document
```

A blue star icon is positioned to the left of the code line.

# Declaring the Variable



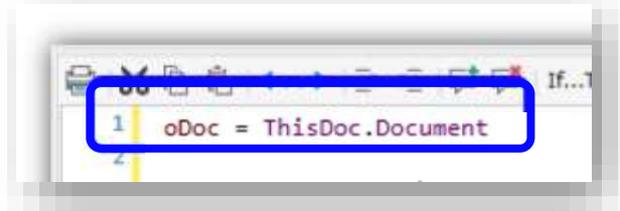
The screenshot shows a code editor window with a toolbar at the top. The first line of code, `oDoc = ThisDoc.Document`, is highlighted with a blue box. Below it, the second line is blank, and the third line is `oExtrusion1 = oDoc.e`. A dropdown menu is open from the end of the third line, listing various properties of the `Document` object. The `ClientViews` property is selected and highlighted in blue. To the right of the dropdown, a tooltip box displays the text: `ReadOnly Property Document.ClientViews As ClientViews` followed by the description "Returns document's client views." The editor's status bar at the top right shows "If...Then...End If", "Keywords", "Operators", and "Header...".

```
1 oDoc = ThisDoc.Document
2
3 oExtrusion1 = oDoc.e
```

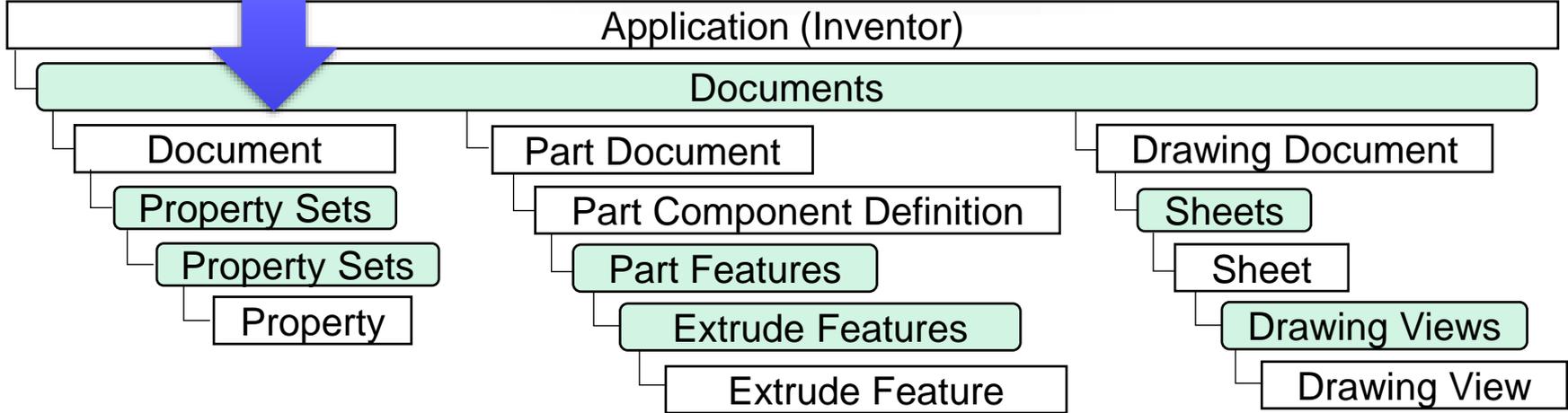
- AllReferencedDocuments
- AttributeManager
- AttributeSets
- BrowserPanes
- ClientViews
- Close
- Compacted
- CreateHighlightSet
- DatabaseRevisionId

ReadOnly Property Document.ClientViews As ClientViews  
Returns document's client views.

# What happens when we declare the variable?

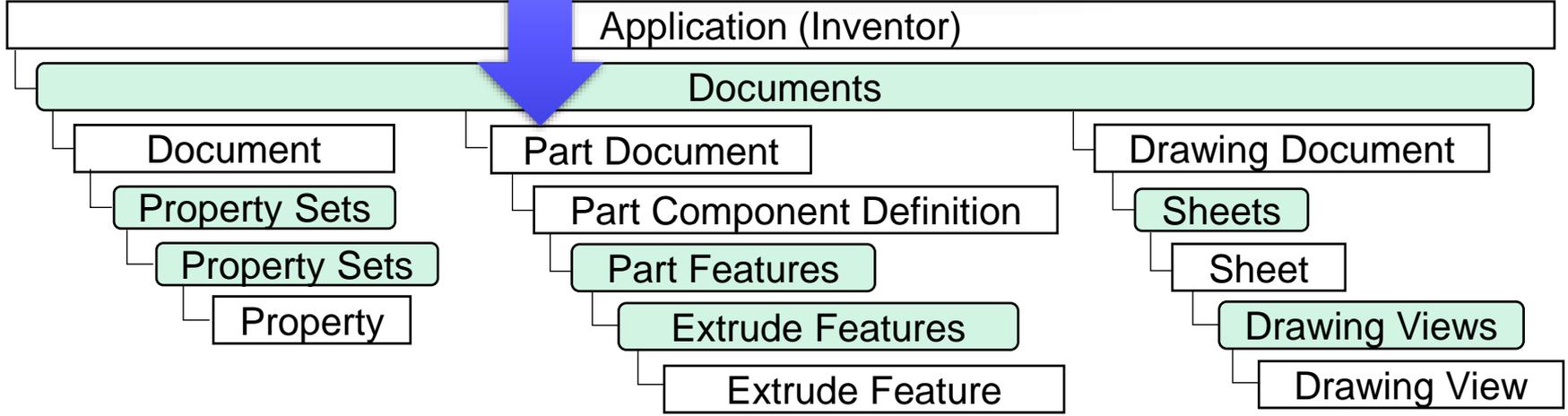


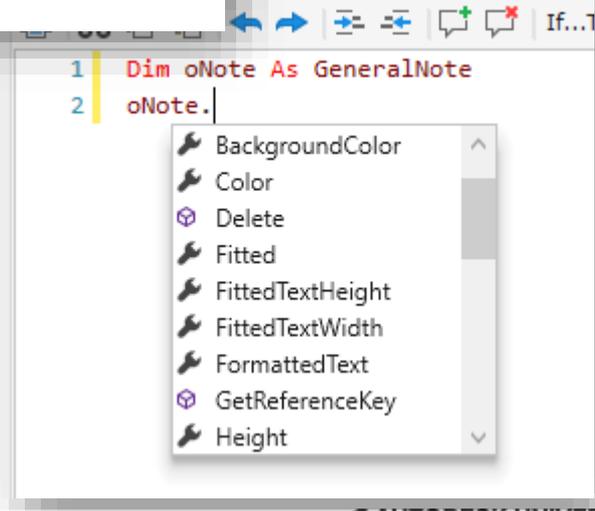
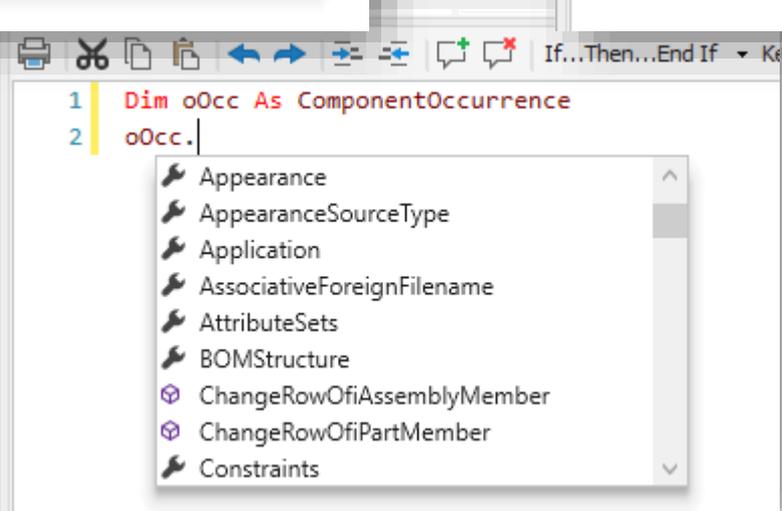
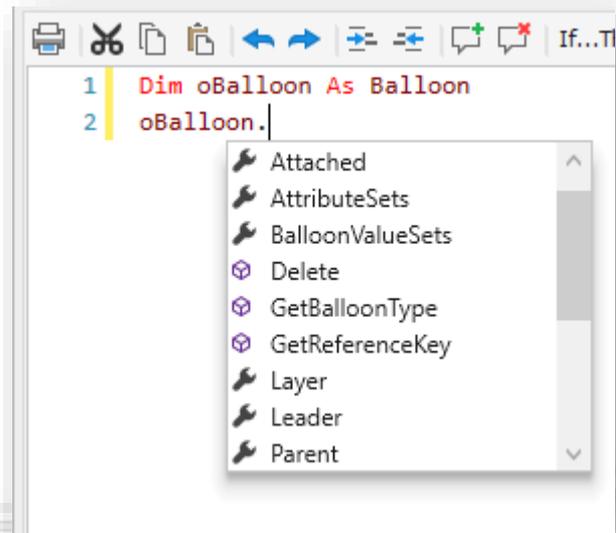
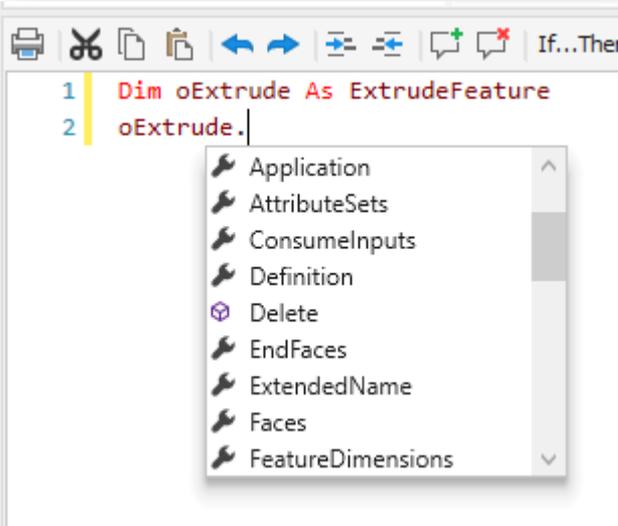
```
1 oDoc = ThisDoc.Document
2
```



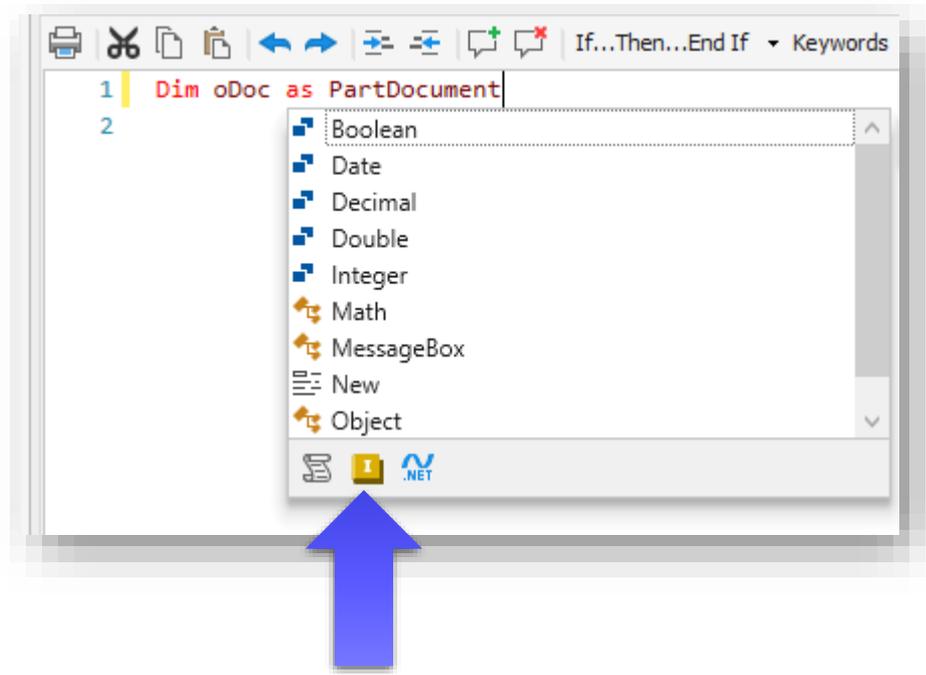
# What happens when we declare the variable?

```
1 Dim oDoc As PartDocument
2 oDoc = ThisDoc.Document
```





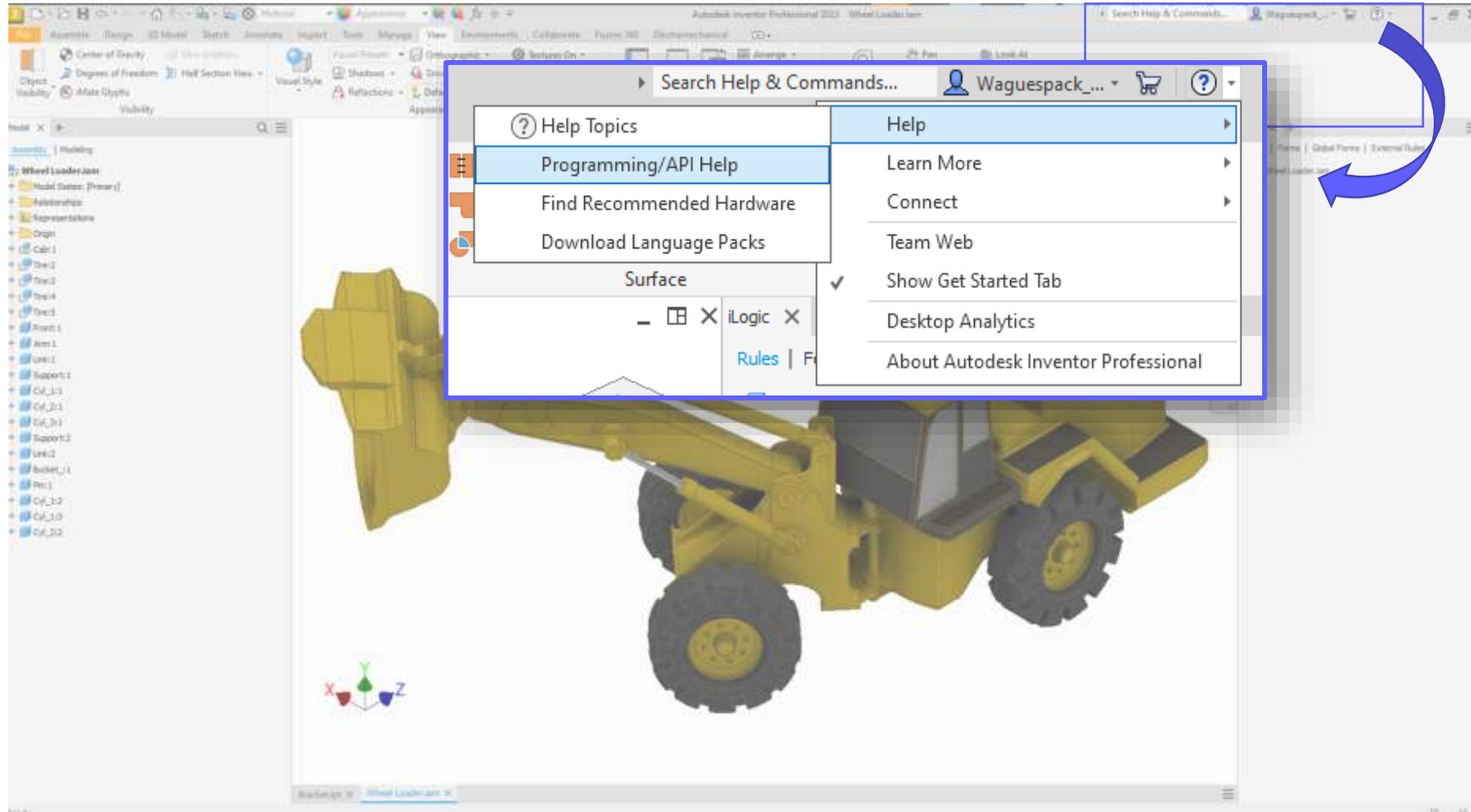
# iLogic Editor IntelliSense Filters



A background image showing a close-up, perspective view of a metal mesh or woven fabric, with the mesh receding into the distance. The image is partially obscured by a black diagonal shape that contains the text.

# Access API and Programming Help Files

Tip #14

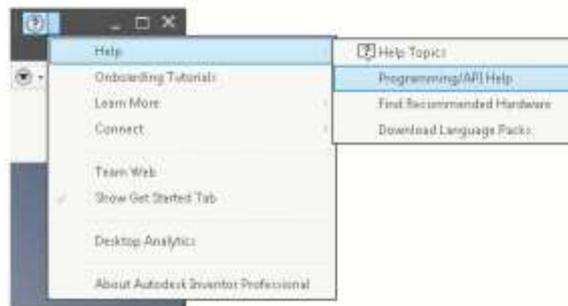


## Introduction to Using Inventor's Programming Interface

There are several resources provided to help you use Inventor's Application Programming Interface (API). These resources are all part of Inventor's Software Development Kit (SDK). The various elements of the SDK and some additional external resources are described below.

### API Help

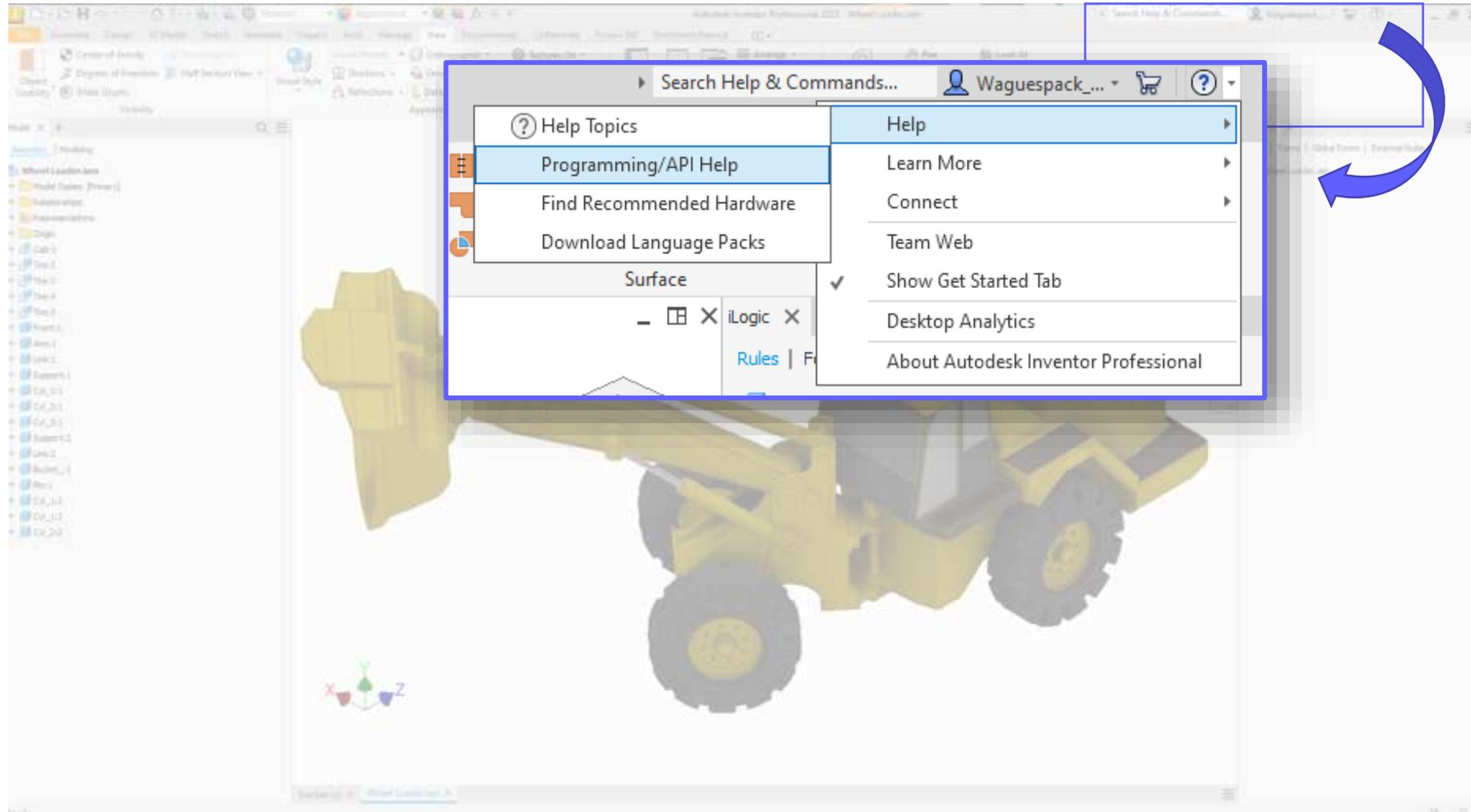
The API Help is installed with Inventor and is accessed from the Help menu as shown below.



The help content consists of several parts:

- Introduction to the API, which is what you're reading now.
- What's new in this release of Inventor. This lists the changes that have been made in Inventor that may require some changes to any existing programs and lists the new objects, method, properties, and events that have been added for this release.
- User's manual which provides overview topics of much of the API.
- Reference manual. The reference manual provides detailed information about every object, method, property, and event. If that topic is demonstrated in a sample, there is a link to the sample in that topic.
- Sample programs. This is a categorized list of the sample programs. These are the same samples that are also accessed through links in the reference manual topics. They are primarily VBA programs, with a few C# programs. The API is the same regardless of what language is used. It's the syntax that changes, so a program in any language can serve as an example of how to use the API.

### SDK Folder



Search Help & Commands... Waguespack\_... ?

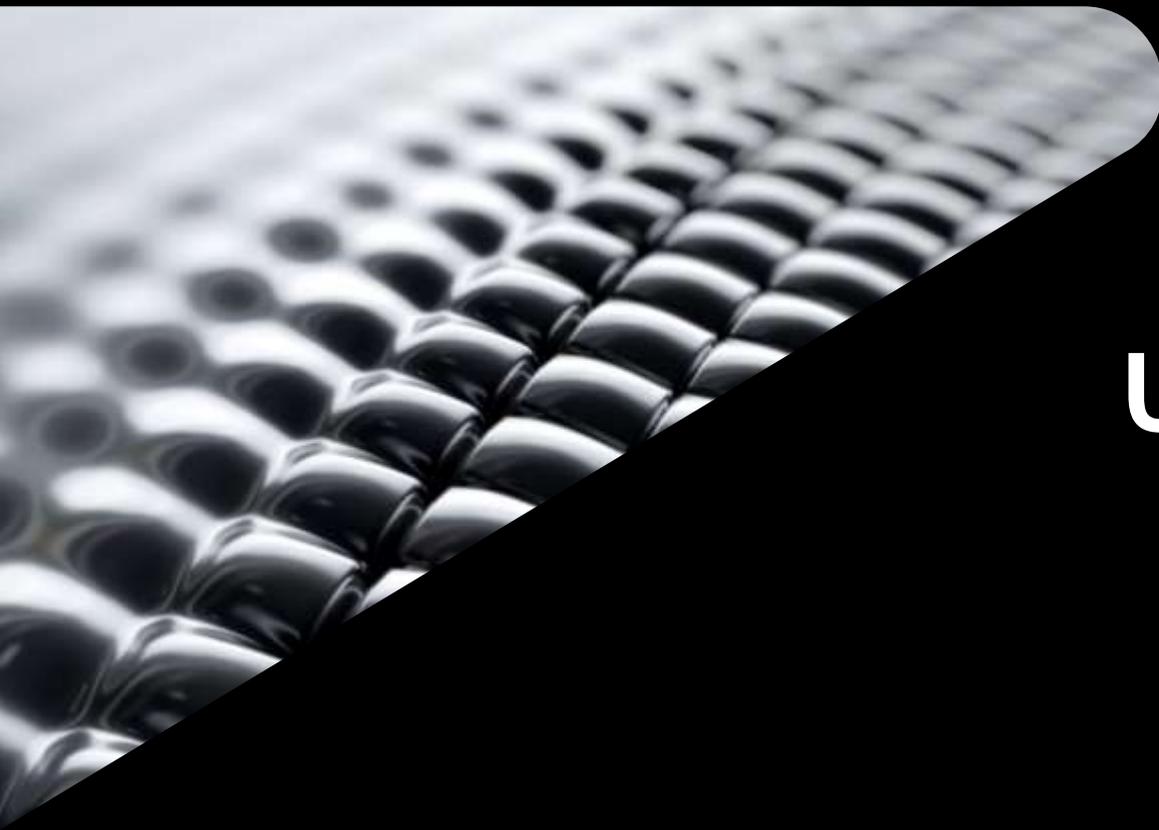
- Help Topics
- Programming/API Help
- Find Recommended Hardware
- Download Language Packs

Surface

- Help
- Learn More
- Connect
- Team Web
- Show Get Started Tab
- Desktop Analytics
- About Autodesk Inventor Professional

Objective:

**Explore development and error handling techniques**

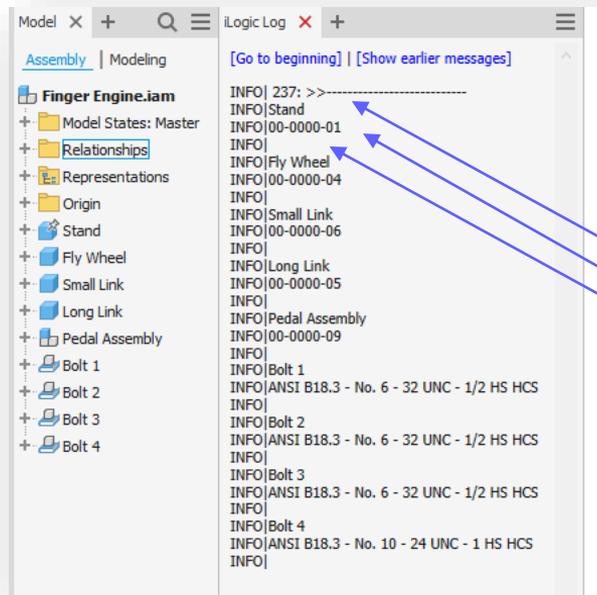


# Use the iLogic Logger

Tip #15

# What is the iLogic Logger?

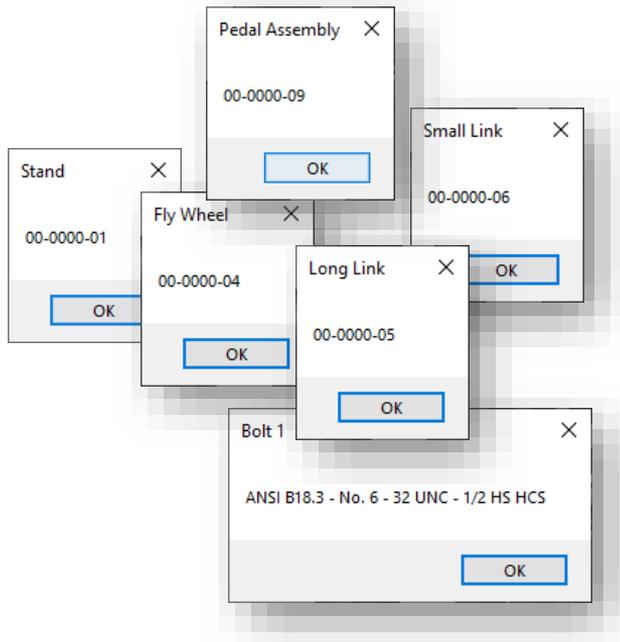
- The iLogic Logger is a tool that we can use to view or report information from our iLogic rules
- This could be model information, errors, etc
- In this example a rule writes out the occurrence name and part number to the logger



```
1 Dim oDoc As AssemblyDocument
2 oDoc = ThisDoc.Document
3
4 Dim oOcc As ComponentOccurrence
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     oPn = iProperties.Value(oOcc.Name, "Project", "Part Number")
8
9     Logger.Info(oOcc.Name)
10    Logger.Info(oPn)
11    Logger.Info("")
12
13 Next
```

# Works similar to how we someone use message boxes?

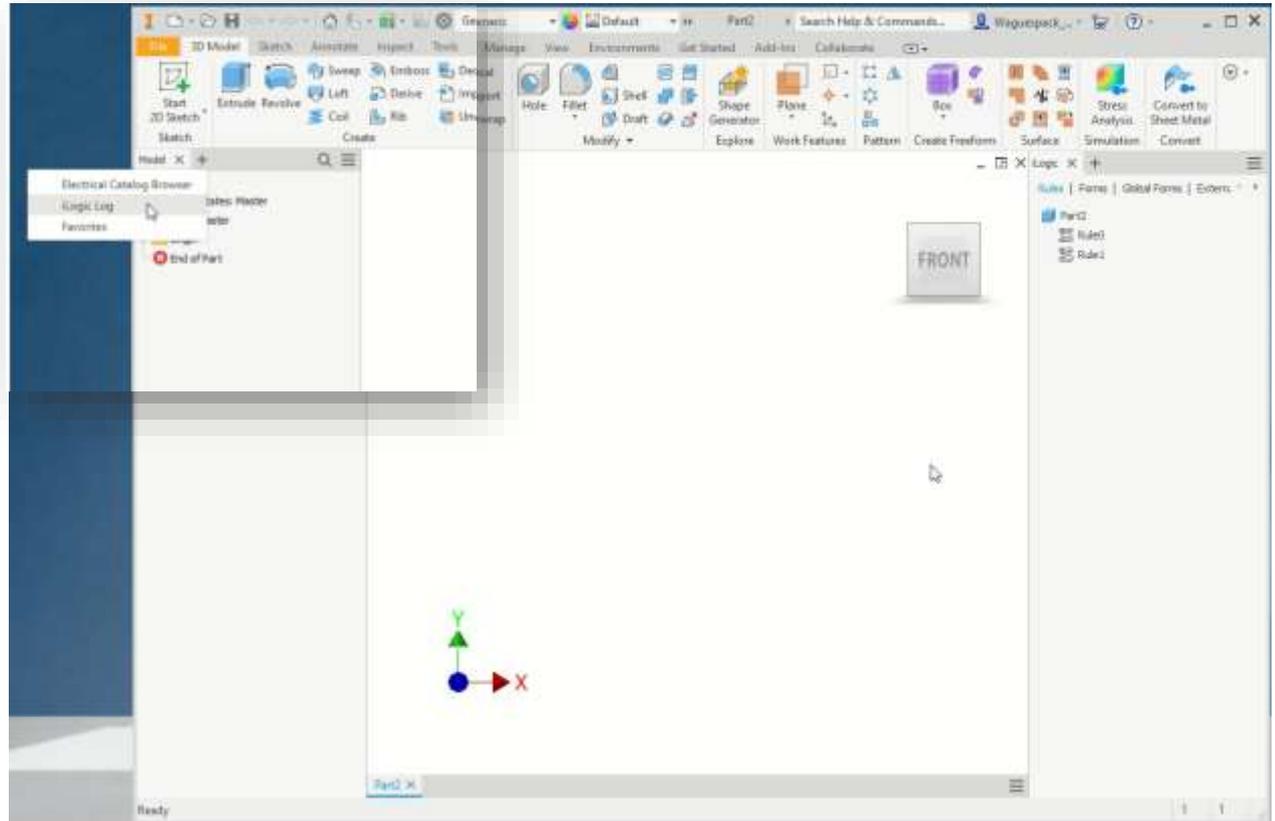
- The iLogic Logger is often a more efficient, less disruptive way to report and view information, that traditional message box sleuthing



```
1 Dim oDoc As AssemblyDocument
2 oDoc = ThisDoc.Document
3
4 Dim oOcc As ComponentOccurrence
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     oPn = iProperties.Value(oOcc.Name, "Project", "Part Number")
8     MessageBox.Show(oPn, oOcc.Name)
9
10 Next
```

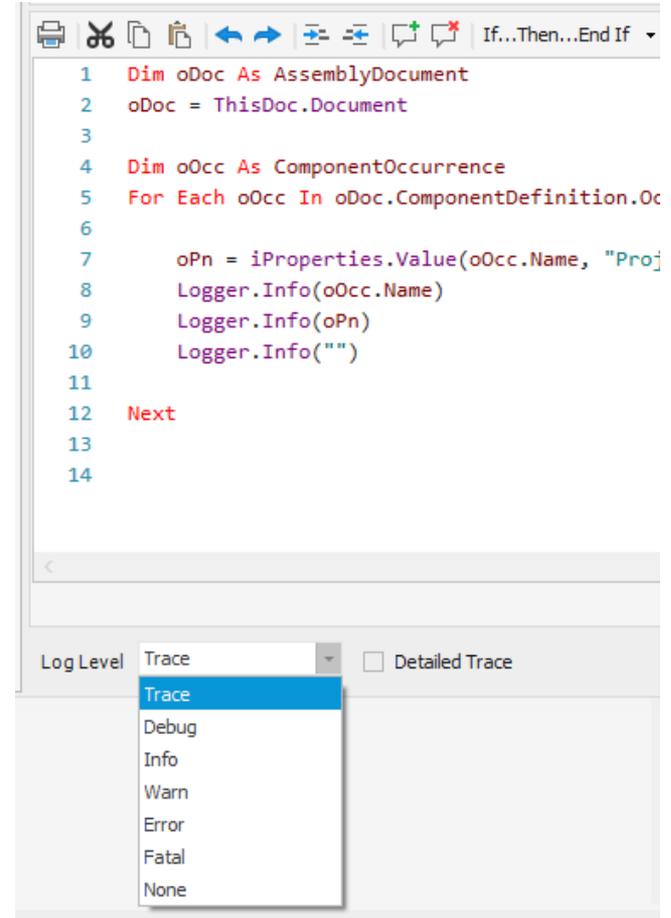
# Accessing the iLogic Logger

- We access the iLogic logger using the + next the standard Model browser
- The Logger can be docked to a location and position of your liking



# Logger Levels

- The logger offers multiple levels of logging accessed via the Log Level drop down in the iLogic code editor

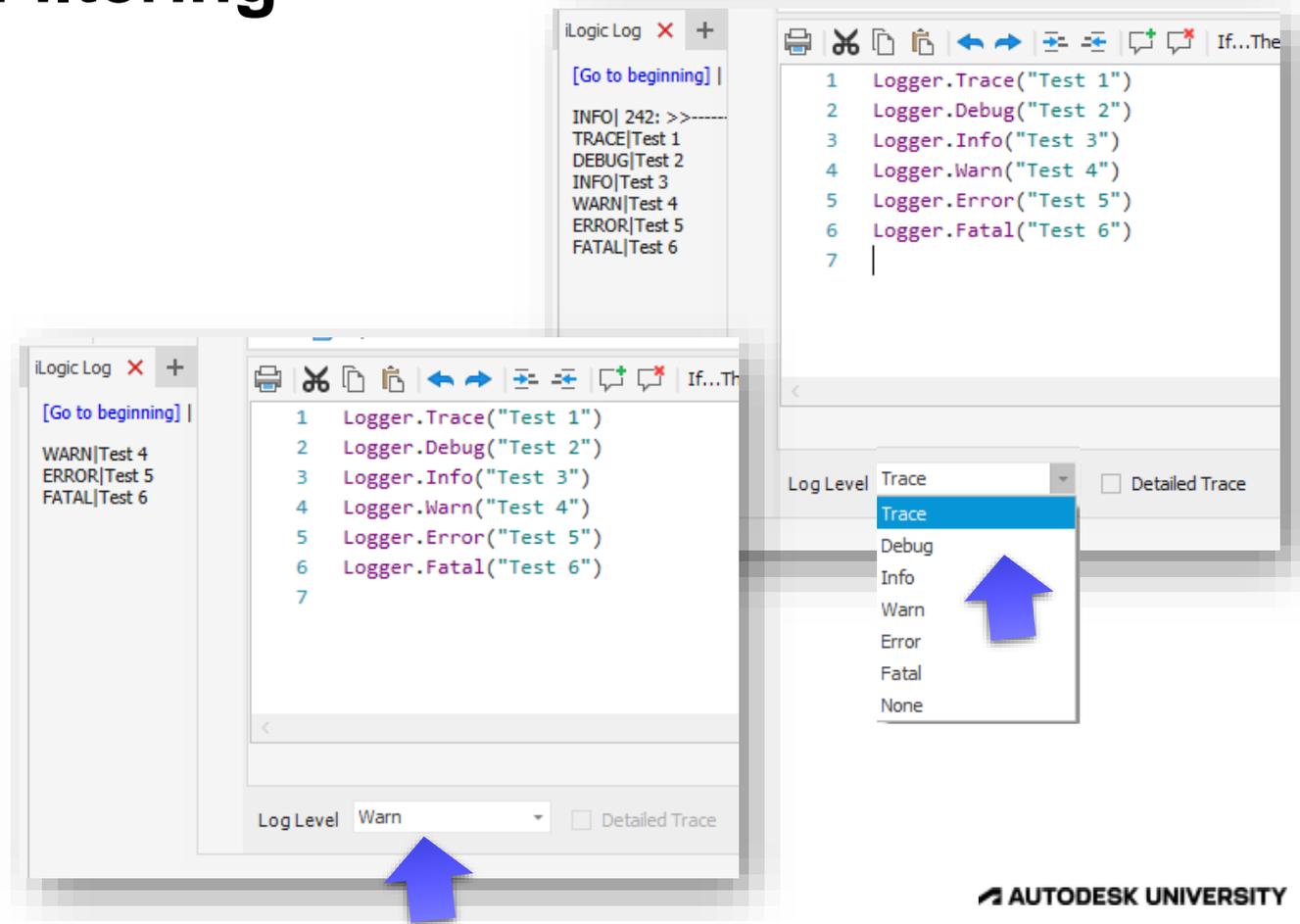


The screenshot shows the iLogic code editor with a VBA script. The script defines a variable `oDoc` of type `AssemblyDocument` and assigns it `ThisDoc.Document`. It then enters a `For Each` loop over `oDoc.ComponentDefinition.Occurrences`. Inside the loop, it retrieves a property value for the occurrence name, "Proj", and logs it using `Logger.Info` three times. The loop ends with `Next`. Below the code editor, the `Log Level` dropdown menu is open, showing options: `Trace` (selected), `Debug`, `Info`, `Warn`, `Error`, `Fatal`, and `None`. There is also a `Detailed Trace` checkbox which is currently unchecked.

```
1 Dim oDoc As AssemblyDocument
2 oDoc = ThisDoc.Document
3
4 Dim oOcc As ComponentOccurrence
5 For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7     oPn = iProperties.Value(oOcc.Name, "Proj")
8     Logger.Info(oOcc.Name)
9     Logger.Info(oPn)
10    Logger.Info("")
11
12 Next
13
14
```

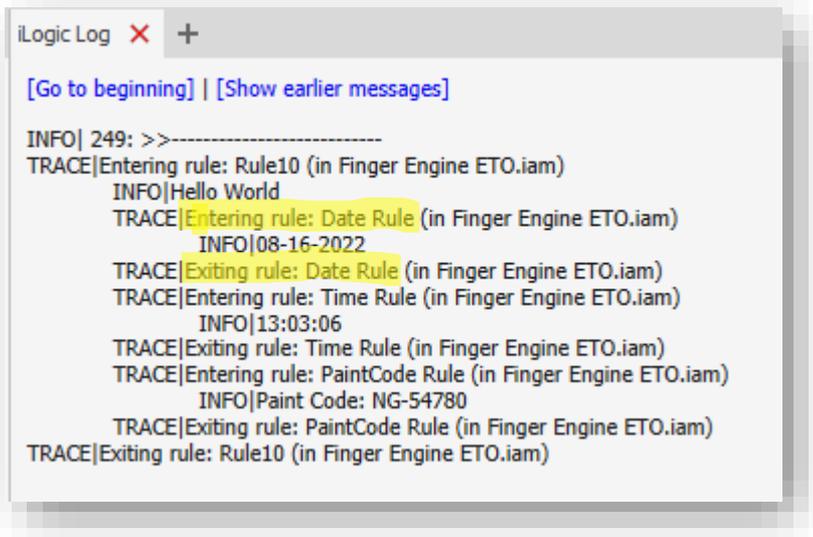
# Logger Level Filtering

- The levels are cascading in nature, meaning that if you set the Log Level to the highest level (Trace) you will see all other levels.
- If you were to set it to a lower level (such as **Warn**) you would only see levels at or below that level



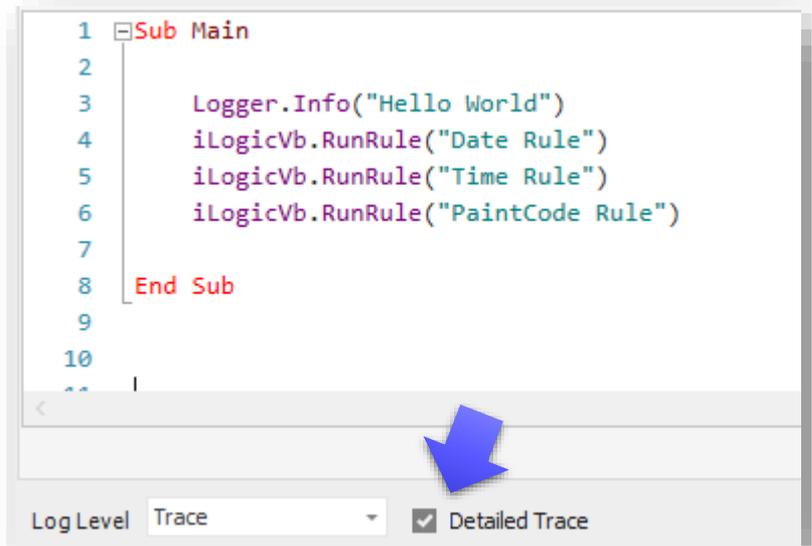
# Detailed Trace

- If we turn on Detailed Trace we monitor when the automation steps into an iLogic Rule
- This can be particularly helpful when we have changes to parameters that are triggering rules excessively or unintentionally



The screenshot shows the iLogic Log window with the following content:

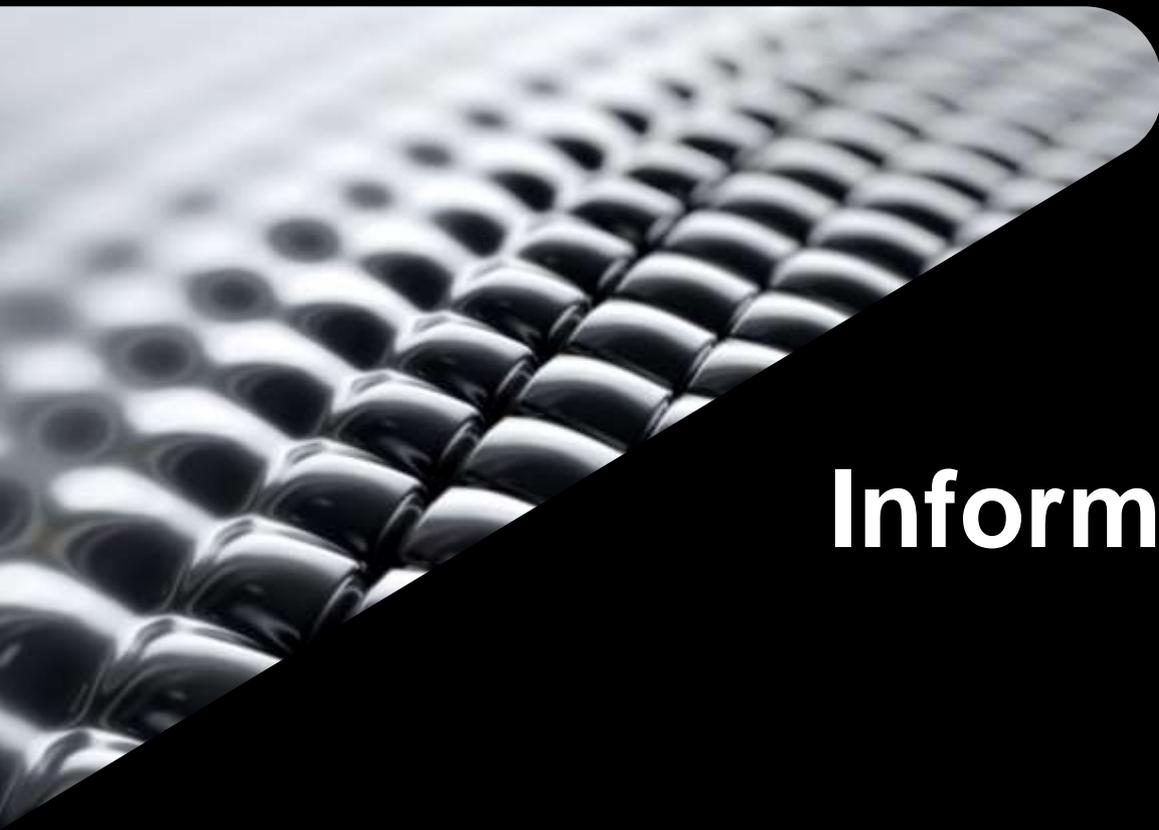
```
iLogic Log X +
[Go to beginning] | [Show earlier messages]
INFO| 249: >>-----
TRACE|Entering rule: Rule10 (in Finger Engine ETO.iam)
INFO|Hello World
TRACE|Entering rule: Date Rule (in Finger Engine ETO.iam)
INFO|08-16-2022
TRACE|Exiting rule: Date Rule (in Finger Engine ETO.iam)
TRACE|Entering rule: Time Rule (in Finger Engine ETO.iam)
INFO|13:03:06
TRACE|Exiting rule: Time Rule (in Finger Engine ETO.iam)
TRACE|Entering rule: PaintCode Rule (in Finger Engine ETO.iam)
INFO|Paint Code: NG-54780
TRACE|Exiting rule: PaintCode Rule (in Finger Engine ETO.iam)
TRACE|Exiting rule: Rule10 (in Finger Engine ETO.iam)
```



The screenshot shows the iLogic code editor with the following code:

```
1 Sub Main
2
3     Logger.Info("Hello World")
4     iLogicVb.RunRule("Date Rule")
5     iLogicVb.RunRule("Time Rule")
6     iLogicVb.RunRule("PaintCode Rule")
7
8 End Sub
9
10
11
```

At the bottom of the editor, the Log Level is set to "Trace" and the "Detailed Trace" checkbox is checked. A blue arrow points to this checkbox.

A close-up, black and white photograph of a metal mesh or grate, showing the repeating pattern of the mesh. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# Write Logger Information to a File

Tip #16

# External Logging

- It's often advantageous to have the Logger information written out to an external file
- We can do this by writing out a \*.txt file
- Example provided

```
1 Class Thisrule
2   Dim oStartMarker As String
3   Dim oStartTime As DateTime
4   Dim oElapsedTime As TimeSpan
5   Dim oTimeout As Integer
6
7   Sub Main
8     Call StartLogging 'start logger/timer
9     Dim oDoc As AssemblyDocument
10    oDoc = ThisDoc.Document
11    Dim oOcc As ComponentOccurrence
12    For Each oOcc In oDoc.ComponentDefinition.Occurrences
13      oPn = iProperties.Value(oOcc.Name, "Project", "Part Number")
14      Logger.Info(oOcc.Name)
15      Logger.Info(oPn)
16      Logger.Info("")
17    Next
18    'Finalize
19    Call EndLogging
20  End Sub
21
22  Sub StartLogging
23    oStartMarker = "iLogic log"
24    Logger.Info(oStartMarker)
25    Logger.Info(DateTime.Now)
26    oStartTime = Now ' Start the timer from this point
27  End Sub
28
29  Sub EndLogging
30    ' Stop the timer
31    oElapsedTime = Now().Subtract(oStartTime)
32    oTime = oElapsedTime.TotalSeconds
33    oTime = Round(oTime, 2)
34    oLine1 = "iLogic Complete! "
35    oLine2 = "Total Run Time : " & oTime & " seconds "
36    oTimeout = 3.5 ' seconds
37    Logger.Info(oLine1)
38    Logger.Info(oLine2)
39    'write out the log file
40    Dim oLogFileName = "C:\Temp\iLogic log.txt"
41    iLogicVB.Automation.LogControl.SaveLogAs(oLogFileName)
42    Dim oLogText = System.IO.File.ReadAllText(oLogFileName)
43    Dim oMarkerIndex = oLogText.LastIndexOf(oStartMarker)
44    ' Replace the file contents with the portion from the last marker
45    System.IO.File.WriteAllText(oLogFileName, oLogText.Substring(oMarkerIndex + 1))
46    ThisDoc.Launch(oLogFileName)
47  End Sub
48 End Class
```

Logic log.txt - Notepad  
File Edit Format View Help  
iLogic Log  
INFO|8/16/2022 1:15:50 PM  
INFO|Stand  
INFO|00-0000-01  
INFO|  
INFO|Fly Wheel  
INFO|00-0000-04  
INFO|  
INFO|Small Link  
INFO|00-0000-00

Logic log.txt - Notepad  
File Edit Format View Help  
iLogic Log  
INFO|8/16/2022 1:27:42 PM  
INFO|Stand  
INFO|00-0000-01  
INFO|  
INFO|Fly Wheel  
INFO|00-0000-04  
INFO|  
INFO|Small Link  
INFO|00-0000-00  
INFO|  
INFO|Long Link  
INFO|00-0000-05  
INFO|  
INFO|Pedal Assembly  
INFO|00-0000-09  
INFO|  
INFO|Bolt 1  
INFO|ANSI B18.3 - No. 6 - 32 UNC - 1/2 HS HCS  
INFO|  
INFO|Bolt 2  
INFO|ANSI B18.3 - No. 6 - 32 UNC - 1/2 HS HCS  
INFO|  
INFO|Bolt 3  
INFO|ANSI B18.3 - No. 6 - 32 UNC - 1/2 HS HCS  
INFO|  
INFO|Bolt 4  
INFO|ANSI B18.3 - No. 10 - 24 UNC - 1 HS HCS  
INFO|  
INFO|iLogic Complete!  
INFO|Total Run Time : 0.22 seconds

# External Logging



- Tip provided by **Mike Deck** of Autodesk

  [MjDeck](#) als Antwort auf: [Curtis\\_Waguespack](#) 09-24-2019 08:38 PM

✓ Hi [@Curtis\\_Waguespack](#) ,  
There's no straightforward way to do it. We could add that in a future release.  
But here's a hack. Write a marker, and then delete all the text before the last marker.

```
Dim startMarker = "----- Custom iLogic Log marker -----"  
Logger.Info(startMarker)  
  
' < do some stuff here >  
Logger.Info("some info about the stuff that happened starting at " & DateTime.Now)  
  
' < do more some stuff here >  
Logger.Info("some more info about the stuff that just happened...")  
  
'write out the log file  
Dim logFileName = "C:\TEMP\iLogic Log Example.txt"  
iLogicVb.Automation.LogControl.SaveLogAs(logFileName)  
  
Dim logText = System.IO.File.ReadAllText(logFileName)  
Dim markerIndex = logText.LastIndexOf(startMarker)  
' Replace the file contents with the portion from the last marker  
System.IO.File.WriteAllText(logFileName, logText.Substring(markerIndex))
```

 **AUTODESK**  
Mike Deck  
Software Developer  
Autodesk, Inc.



# Inventor iLogic and VB.net Forum

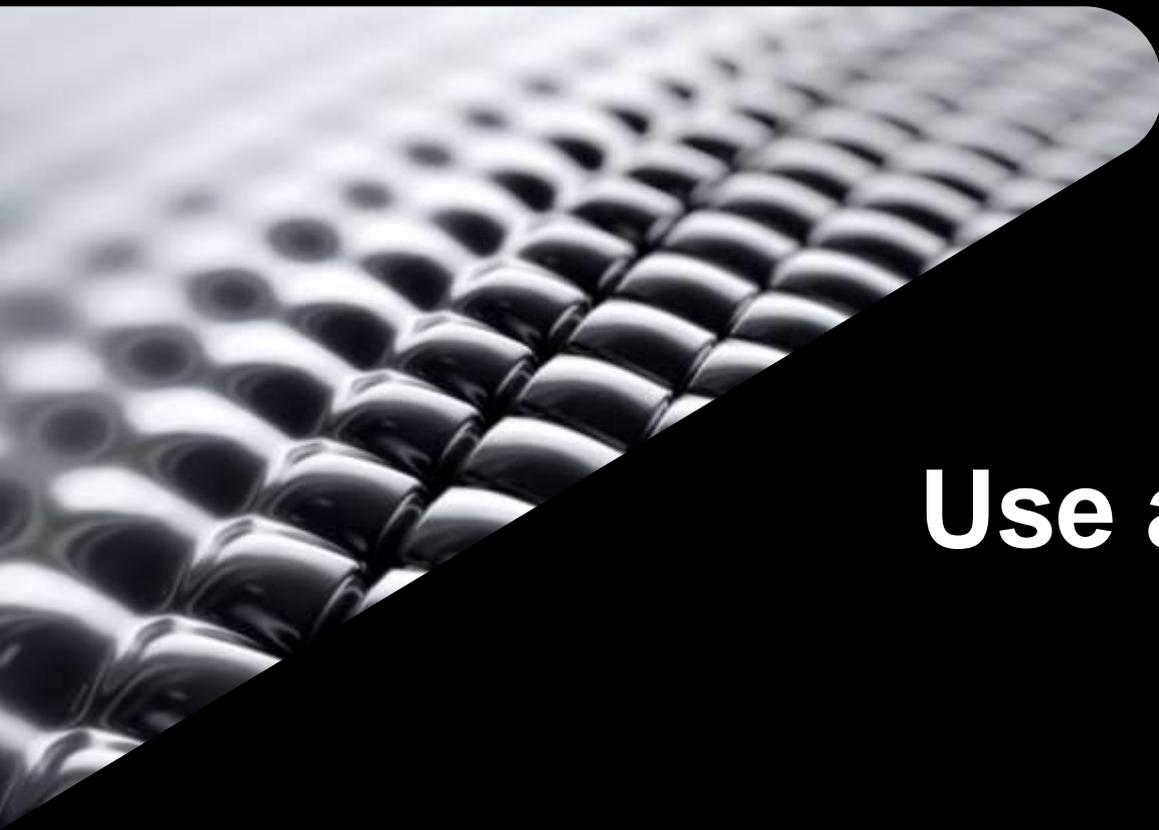
Tip #17

# Visit the forum

- Search for solutions
- Ask questions
- Provide answers
- <https://forums.autodesk.com/t5/inventor-ilogic-and-vb-net-forum/bd-p/120>

The screenshot shows the Autodesk Knowledge Network forum page for 'Inventor iLogic and VB.net Forum'. The page features a search bar at the top with the text 'Dieses Board' and 'Suchen'. Below the search bar, there is a navigation bar with 'IN FOREN VERÖFFENTLICHEN' and a link to 'Zurück zur Kategorie Inventor'. The main content area displays a list of forum posts with columns for 'ANTWORTEN' (Answers) and 'AUFRUFE' (Views). The posts include titles like 'Autodesk University 2022 - Let Us Know if You'll Be There!', 'Accepted Solutions - What They Are, How to Find Them, and How to Accept a Solution', and 'Important for developers working with Level Of Details'. The page also includes a pagination control showing '1 2 3 ... 1033 Nächste'.

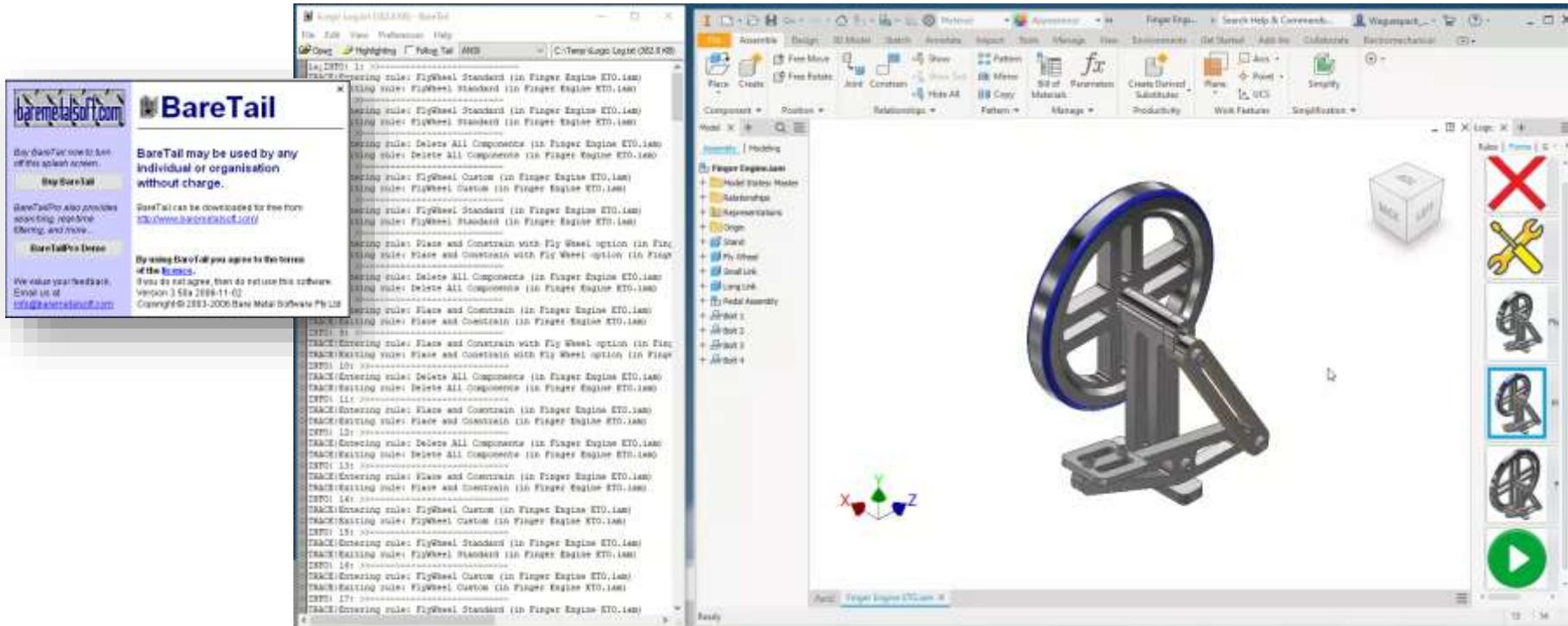
Post Title	Author	Answers	Views
Autodesk University 2022 - Let Us Know if You'll Be There!	CGBenner	0	9
Accepted Solutions - What They Are, How to Find Them, and How to Accept a Solution	CGBenner	0	182
Important for developers working with Level Of Details	adam.nagy	5	2467
New default for iLogic Excel functions in Inventor 2021	MJDeck	4	3125
Check out the self-paced guide "My First Plug-in"	wayne.brill	53	20207
Autodesk University 2022 - Let Us Know if You'll Be There!	CGBenner	0	9
Mirrored Sheet Metal Parts	abarlaan	4	28
If Statement for if a string is a number.	chainesLH3G	4	38



# Use a Log Monitor

Tip #18

# Use a log monitor such as BareTail to watch the iLogic log file



- Watch log outputs real time



# Use Try Catch Statements with the iLogic Logger to catch Exceptions

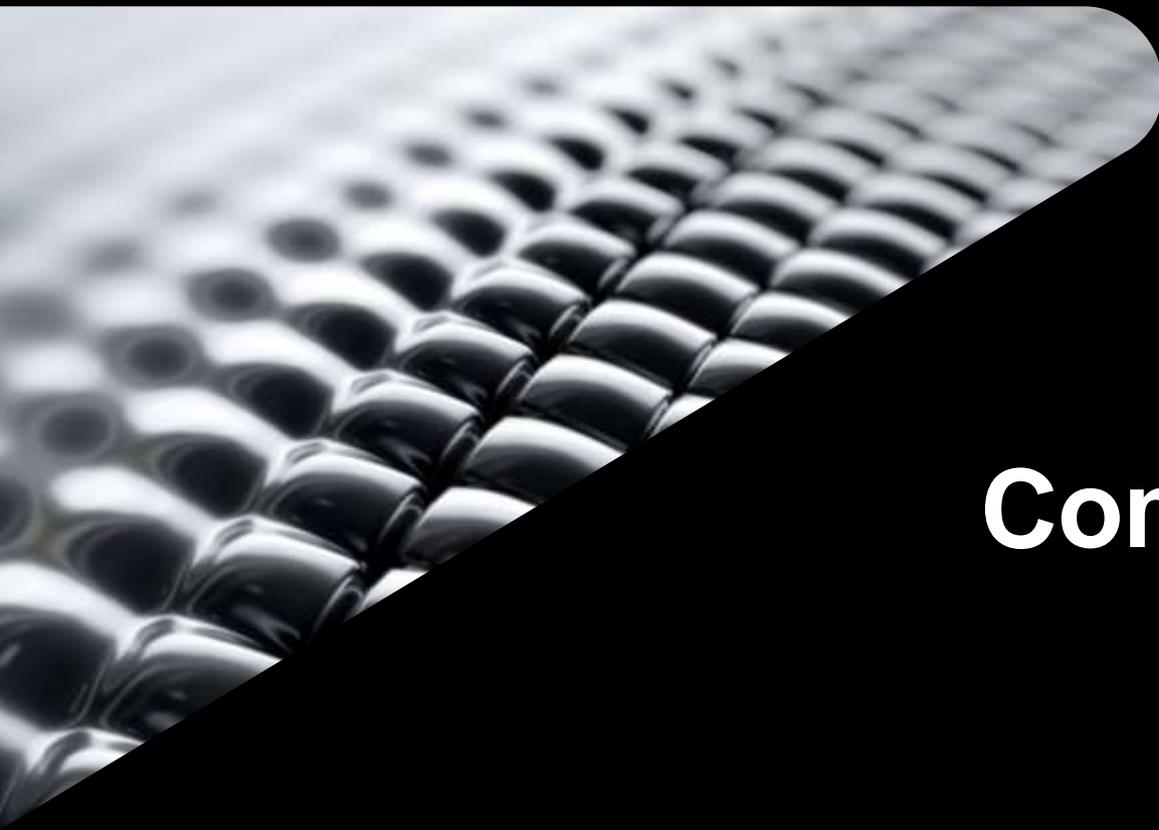
Tip #19

# Catch Errors/Exceptions and log the error line

- A Try Catch statement allows the rule to continue running
- Here a Parameter named Foo does not exist, and produces an error
- The Try Catch statement catches the Exception and passes it to an error handler which passes the **Error message** and the **Error line** to the iLogic Logger

```
1 Sub main
2
3     Try
4         Parameter("Foo") = 4
5     Catch ex As Exception
6         Call HandleErrors(ex)
7     End Try
8
9 End Sub
10
11 Sub HandleErrors(ex As Exception)
12
13     oStackTrace = ex.StackTrace
14     iPos = InStr(oStackTrace, iLogicVb.RuleName)
15     oErrorRuleLine = Mid(oStackTrace, iPos, Len(oStackTrace) - iPos + 1)
16     Logger.Error("Error at: " & oErrorRuleLine)
17     Logger.Info(ex.Message)
18
19 End Sub
```

```
iLogic Log X +
[Go to beginning] | [Show earlier messages]
INFO| 367: >>-----
TRACE|Entering rule: Load Calculations (in Part2)
ERROR|Error at: Load Calculations.vb:line 6
INFO|Parameter: Could not find a parameter named: "Foo"
TRACE|Exiting rule: Load Calculations (in Part2)
```

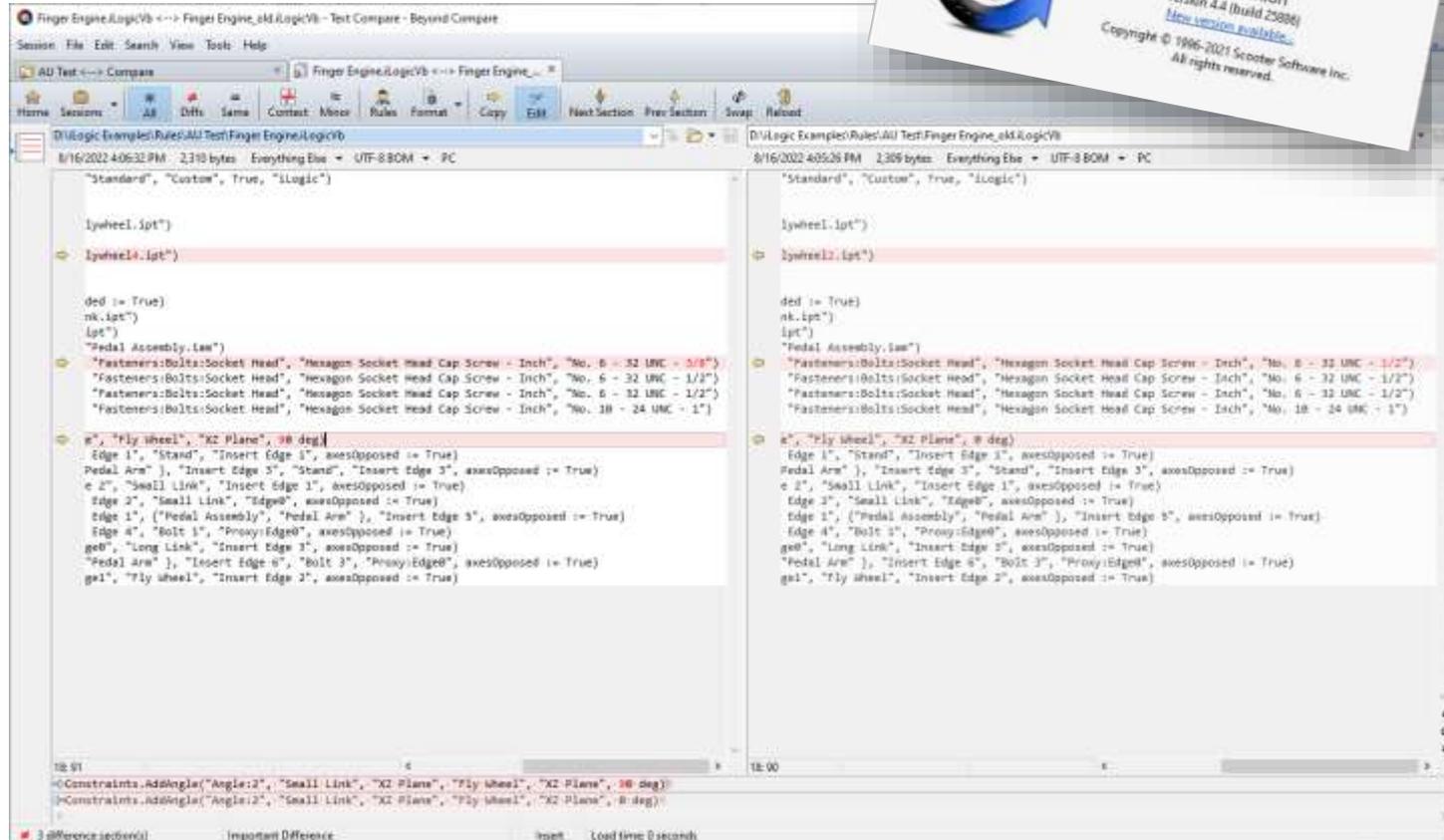


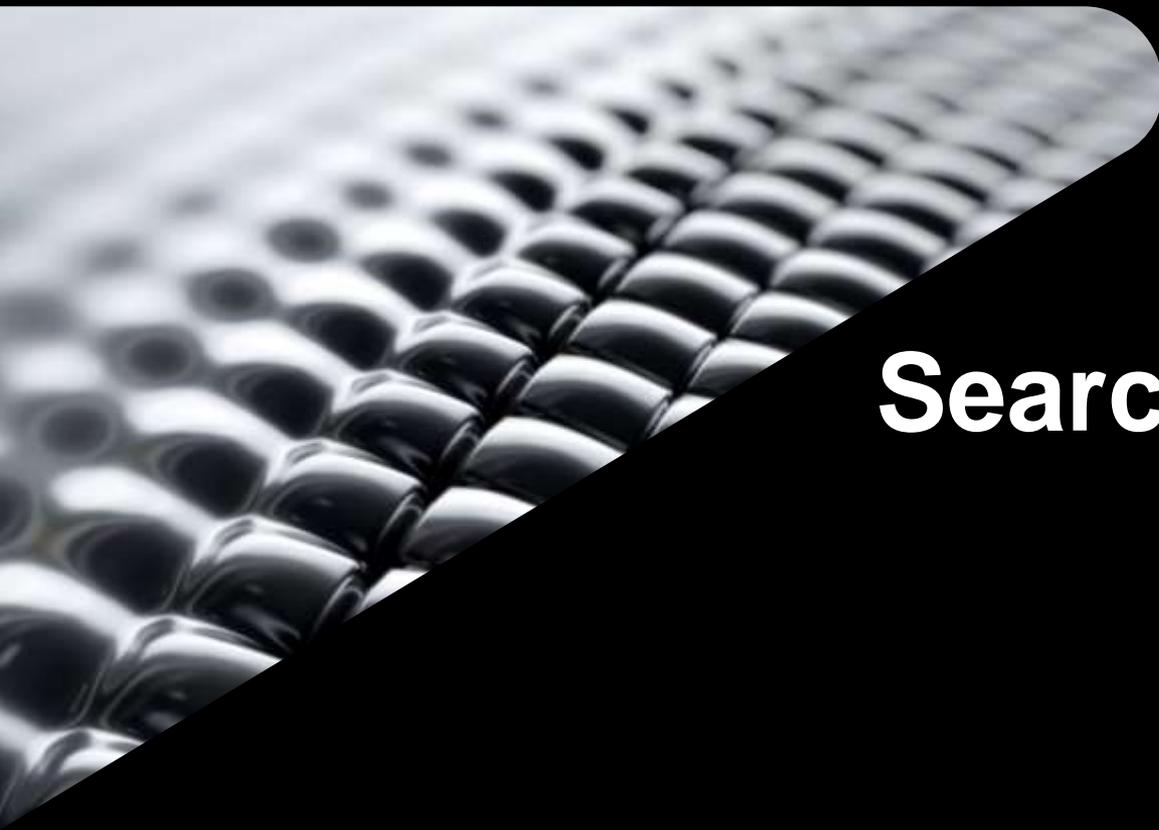
# Use a Code Comparison Tool

Tip #20

# Code Compare

- Assuming that you are using **External iLogic rules**, and are versioning your changes
- Use a tool such as **Beyond Compare** to track and visualize and merge changes to your code.



A close-up, black and white photograph of a metal mesh or woven fabric texture, showing a grid of small, rounded, interconnected elements. The texture is slightly out of focus in the background, creating a sense of depth. This image is partially obscured by a black diagonal shape that serves as a background for the text.

# Search Within Your iLogic Rules

Tip #21

# Search in iLogic Rule Files

- Use a tool such as **NotePad++** to search in your iLogic rules
- Easy to use with External rules
- Internal rules are a bit more of a challenge

Build time : Dec 6 2021 - 19:16:45  
Notepad++ v8.1.9.3 (32-bit)  
Home: <https://notepad-plus-plus.org/>

Find in Files

Find what:

Replace with:

Filters:

Directory:

Match whole word only  
 Match case

Search Mode  
 Normal  
 Extended (n, r, t, \, \x...)  
 Regular expression  . matches newline

Follow current doc.  
 In all sub-folders  
 In hidden folders  
 Transparency  
 On losing focus  
 Always

Search results - (16 hits)

Search "iLogicAuto": (16 hits in 4 files)

- D:\Logic Examples\Rules\Rules to handle iLogic rules\Delete Internal Rule in All Components.iLogicVb (3 hits)
- D:\Logic Examples\Rules\Rules to handle iLogic rules\List all Rules in occurrences.iLogicVb (3 hits)
- D:\Logic Examples\Rules\Rules to handle iLogic rules>Selective Delete Internal Rules.iLogicVb (5 hits)
- D:\Logic Examples\Rules\Rules to handle iLogic rules\Suppress - Unsuppress Internal Rules.iLogicVb (4 hits)

Search "RunExternal": (4 hits in 1 file of 18 searched)

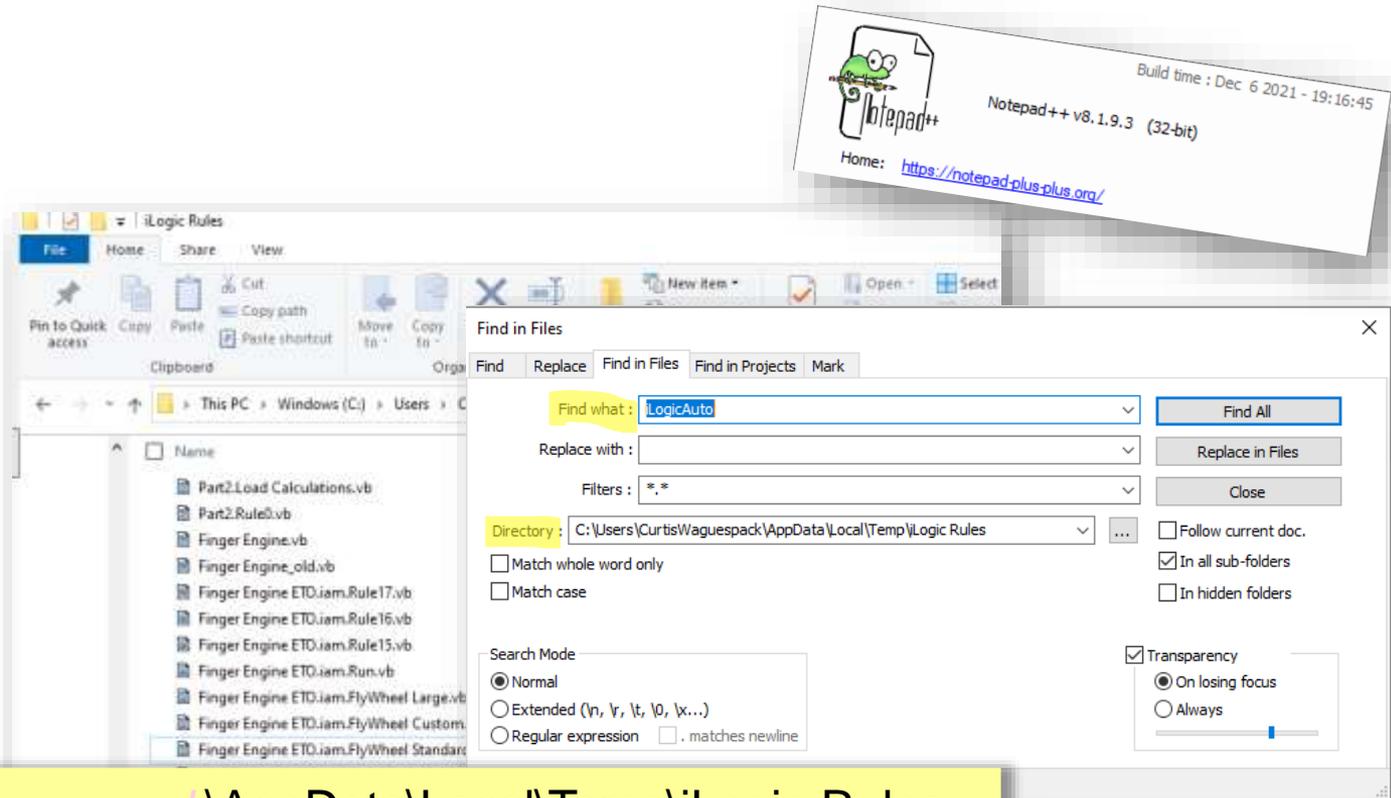
- D:\Logic Examples\Rules\Rules to handle iLogic rules\Create Internal Rule.iLogicVb (4 hits)

```
Line 19: "iLogicVb.RunExternalRule(" & Chr(34) & "Rule111" & Chr(34) & ") " & vbCrLf &
Line 20: "iLogicVb.RunExternalRule(" & Chr(34) & "Rule222" & Chr(34) & ") " & vbCrLf &
Line 21: "iLogicVb.RunExternalRule(" & Chr(34) & "Rule333" & Chr(34) & ") "
```

```
Line 35: iLogicVb.RunExternalRule("Set_Event_Trigger", oValueMap)
```

# Search in internal iLogic Rule Files via the Temp folder

- When you edit rules in Inventor, there is a temporary directory that creates copies of these rules as \*.vb files
- We can use NotePad++ to search in these files also



C:\Users\CurtisWaguespack\AppData\Local\Temp\iLogic Rules



Objective

**Create a Better Interface Between  
Your iLogic Automation and the  
Users Who Employ It**

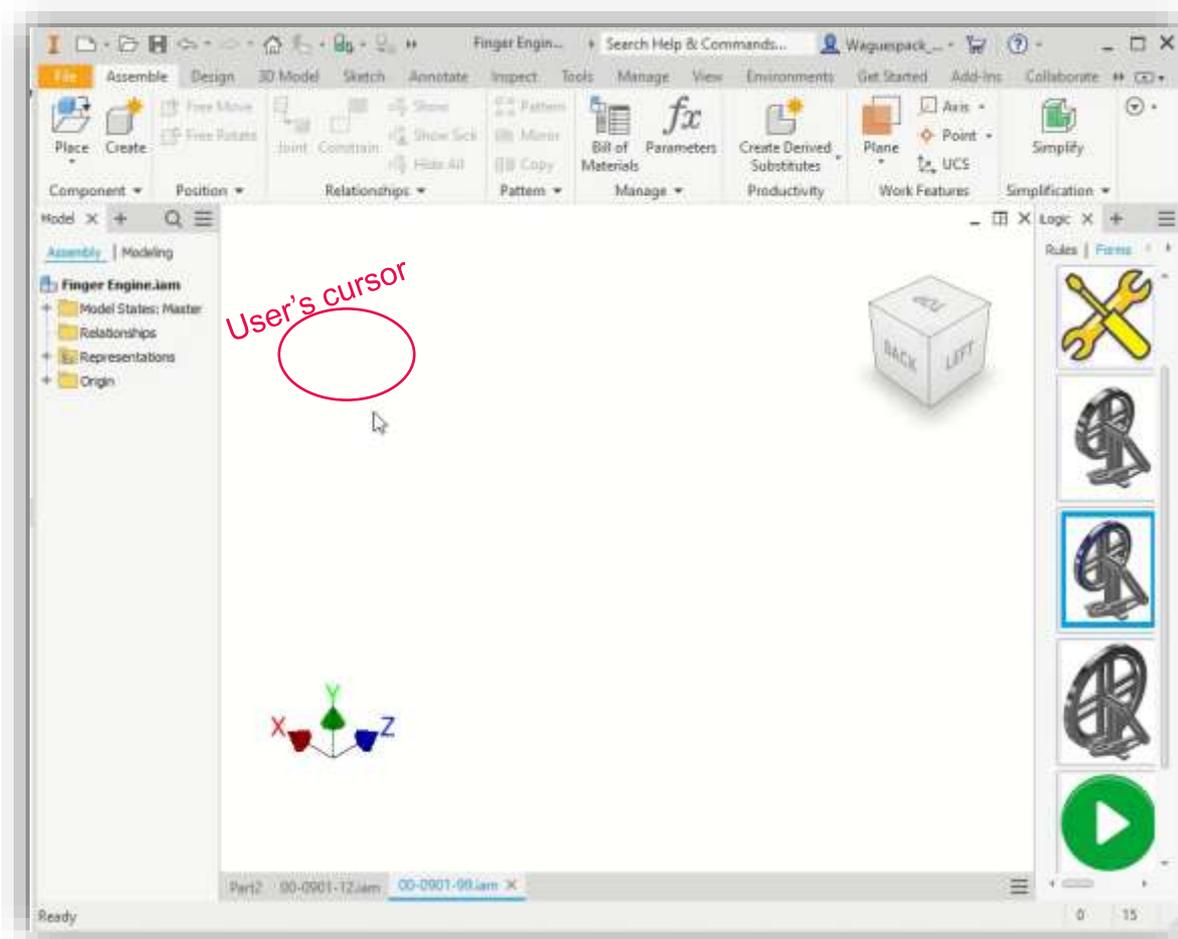
A close-up, black and white photograph of a metal mesh or chain link, showing the texture and perspective of the links. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# Use Timed Message Boxes

Tip #22

# Use a timed message box, between rules

- The message times out, so that it doesn't interrupt the automation, by waiting for a user to click the OK button
- In this example there are 2 rules running.
  - One builds the model then displays a timed message box
  - The second runs another rule, and provided another timed message to the user



# Time Message Box

★ Original code provided by **Mike Deck** of Autodesk on the Inventor iLogic and VB.net forum

- This example calls an external rule named “Timed Message”
- It passes the external rule 2 arguments:
  - Message text “Model created successfully...”
  - a “Timeout” value of 2.5 seconds

## Calling Rule

```
23
24 InventorVb.DocumentUpdate()
25 ThisApplication.ActiveView.Fit
26
27 Dim oValueMap As Inventor.NameValueMap = ThisApplication.TransientObjects.CreateNameValueMap()
28 oValueMap.Add("Message", "Model created successfully." & vbCrLf & vbCrLf & "Preparing to run motion test rule.")
29 oValueMap.Add("Timeout", "2.5")
30 iLogicVb.RunExternalRule("Timed Message", oValueMap)
31
32 iLogicVb.RunRule("Run")
```

## Timed Message rule

```
1 Sub Main
2   'set default
3   oTimeout = 1.5 ' seconds
4   oMsg = "Hello World"
5
6   'get value map argument from other rule
7   If Not RuleArguments("Timeout") = "" Then oTimeout = RuleArguments("Timeout")
8   If Not RuleArguments("Message") = "" Then oMsg = RuleArguments("Message")
9
10  Call TimedMessage(oMsg, oTimeout)
11
12 End Sub
13
14
15 Sub TimedMessage(oMsg As String, oTimeout As Double)
16
17 Dim oForm As New Form() With { .Enabled = True }
18 System.Threading.Tasks.Task.Delay(TimeSpan.FromSeconds(oTimeout)).ContinueWith(Sub(t)
19 oForm.Close()
20 End Sub ,
21 System.Threading.Tasks.TaskScheduler.FromCurrentSynchronizationContext())
22
23 MessageBox.Show(oForm, String.Format(oMsg, oTimeout), "iLogic")
24
25 End Sub
26
```



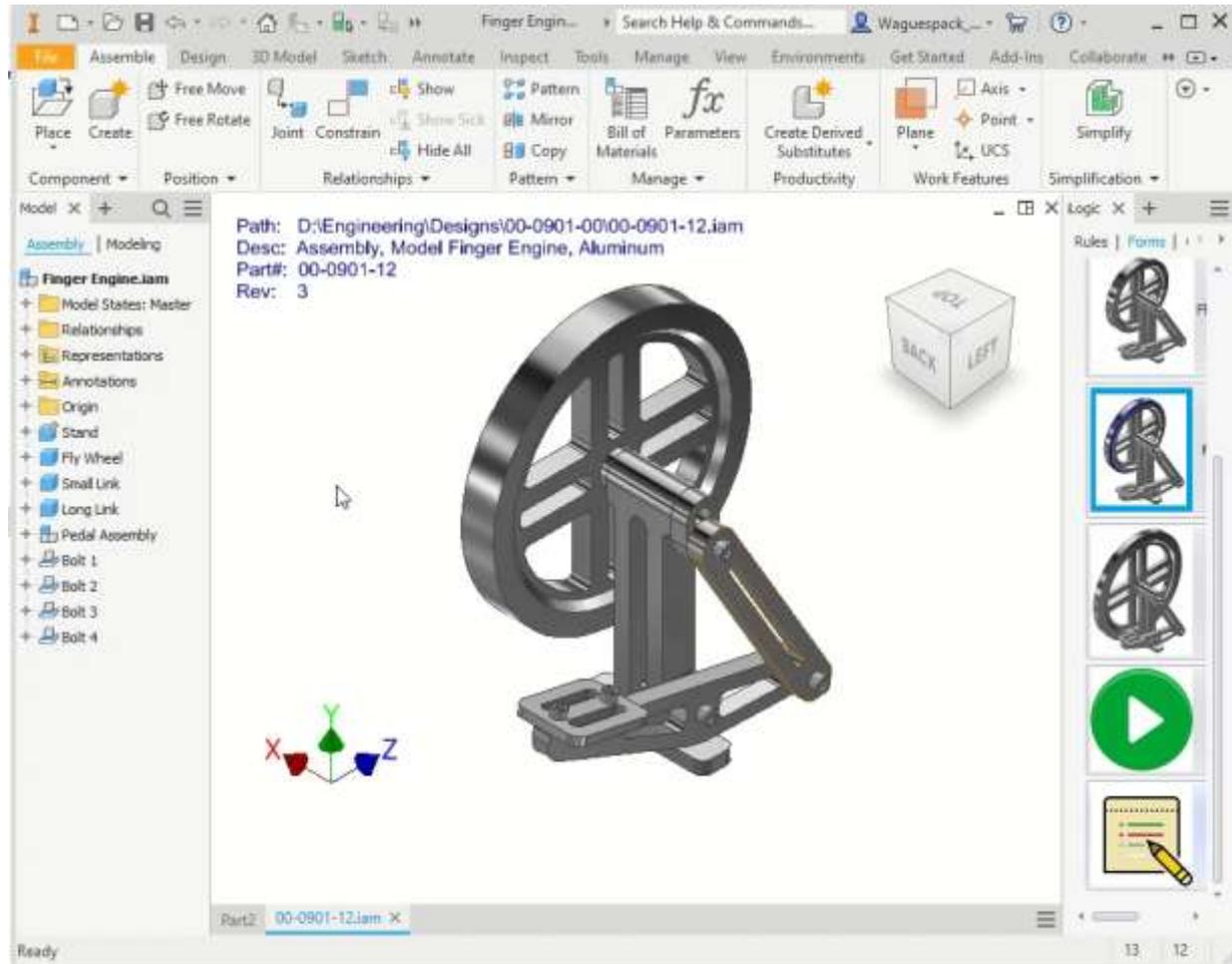
# Display Information in the graphics area

Tip #23

# Annotation Information

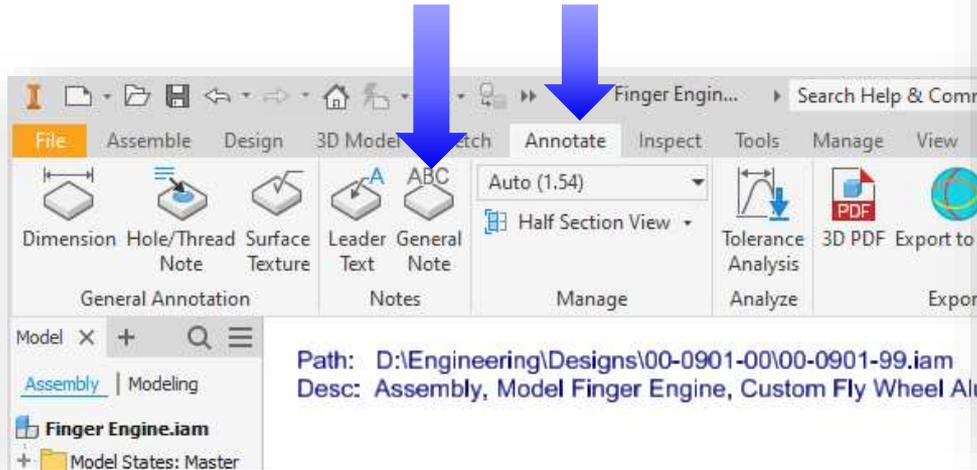
★ Tip provided by  
**Andrew Humiston**

- Use a General Annotation Link Note to display file information to your users.

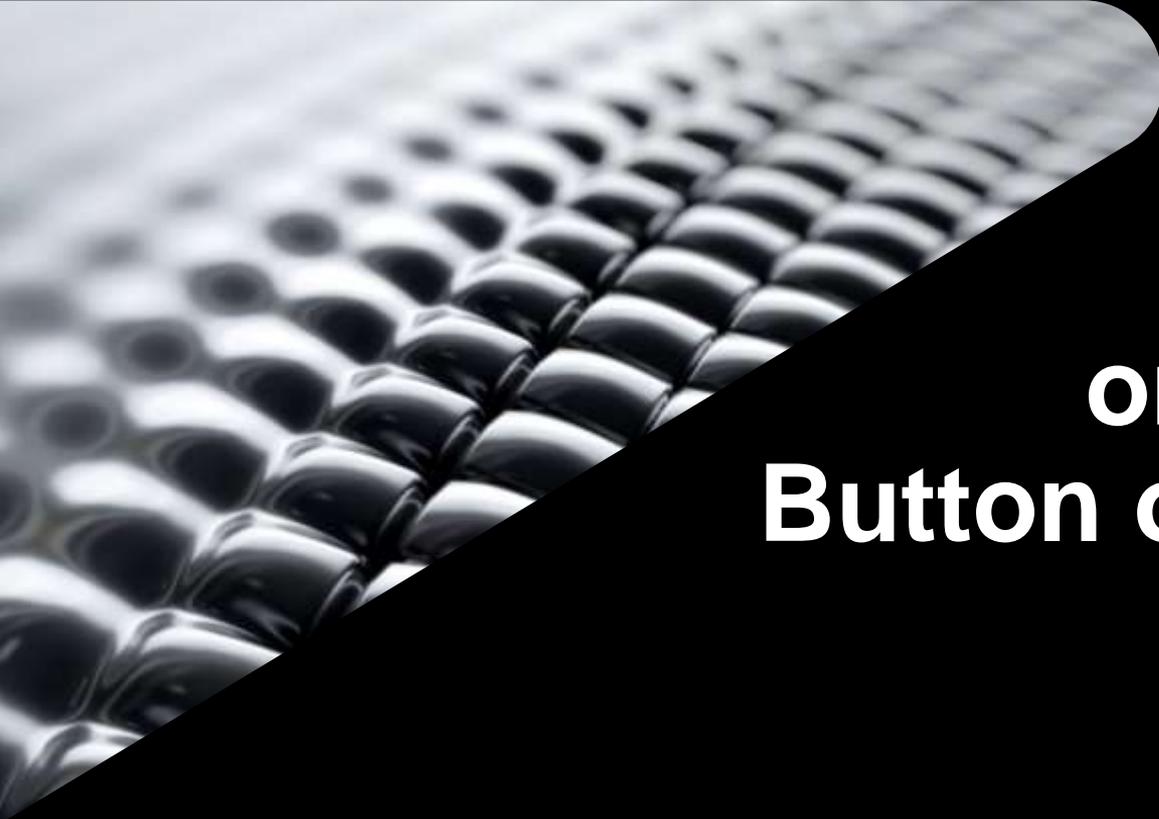


# Annotation Note

- Creates and updates a General Annotation Note
- We use iLogic Trigger Events to update the note.
  - Example: The Before Save event



```
1 Sub Main
2   Dim oDoc As Document = ThisDoc.Document
3
4   'get general notes collection
5   sNotes = oDoc.ComponentDefinition.ModelAnnotations.ModelGeneralNotes
6
7   Dim oInfoNote As ModelGeneralNote
8   Dim oNoteDef As ModelGeneralNoteDefinition
9
10  Try : sNotes.Item("Information Note").delete : Catch : End Try
11  dFontSize = 0.12
12  oNoteDef = sNotes.CreateDefinition("", True, kUpperLeftQuadrant)
13  oNoteDef.Text.Size = dFontSize
14  oInfoNote = sNotes.Add(oNoteDef)
15  oInfoNote.Name = "Information Note"
16
17  'get existing note text
18  sString = sNotes.Item(1).Definition.Text.FormattedText
19
20  sValue = oDoc.FullFileName
21  sText = sText & "Path: " & sValue & "<Br/>"
22
23  sName = "Description"
24  sValue = iProperties.Value("Project", sName)
25  sText = sText & "Desc: " & sValue & "<Br/>"
26
27  sFormattedText = FormatText(sText, dFontSize)
28  sNotes.Item(1).Definition.Text.FormattedText = sFormattedText
29 End Sub
30
31 Function FormatText(sString1 As String, dFontSize As Double)
32
33   sFormattedText = "<StyleOverride " & "FontSize='" & dFontSize & "' " & _
34   ">" & sString1 & "</StyleOverride>" & "<Br/>"
35   Return sFormattedText
36 End Function
```

A close-up, black and white photograph of a metal mesh or woven fabric, showing a grid of circular openings. The image is partially obscured by a black diagonal shape that serves as a background for the text.

# External Rule or Global Form Button on the Ribbon

Tip #24

# Add buttons for External Rules and Global Forms to the ribbon

The image shows the Autodesk Inventor 2023 ribbon and the Customize dialog box. A green banner at the top reads "New to Inventor 2023". Two blue arrows point to the "User Commands" section of the ribbon and the "Customize" dialog box. A third blue arrow points to the "iLogic Rules" list in the dialog box.

**New to Inventor 2023**

The ribbon shows the following sections: Application Options, Document Settings, Migrate Settings, Autodesk App Manager, Highlight New, Add-Ins, Clipboard (Copy, Cut, Paste Special), Find (Find Component), Team Web, and User Commands (PDF all IDWs in folder).

The **Customize** dialog box is open, showing the **Ribbon** tab. The "Choose commands from:" dropdown is set to **iLogic Rules**. The list of commands includes:

Name
Assign_Color
Author Roof Entities
Auto-Arrange Dimensions
Automatic WorkPoints
Balloon Filtering iLogic

The "Choose tab to add custom panel to:" dropdown is set to **Part | Tools**. The list of commands in this tab includes:

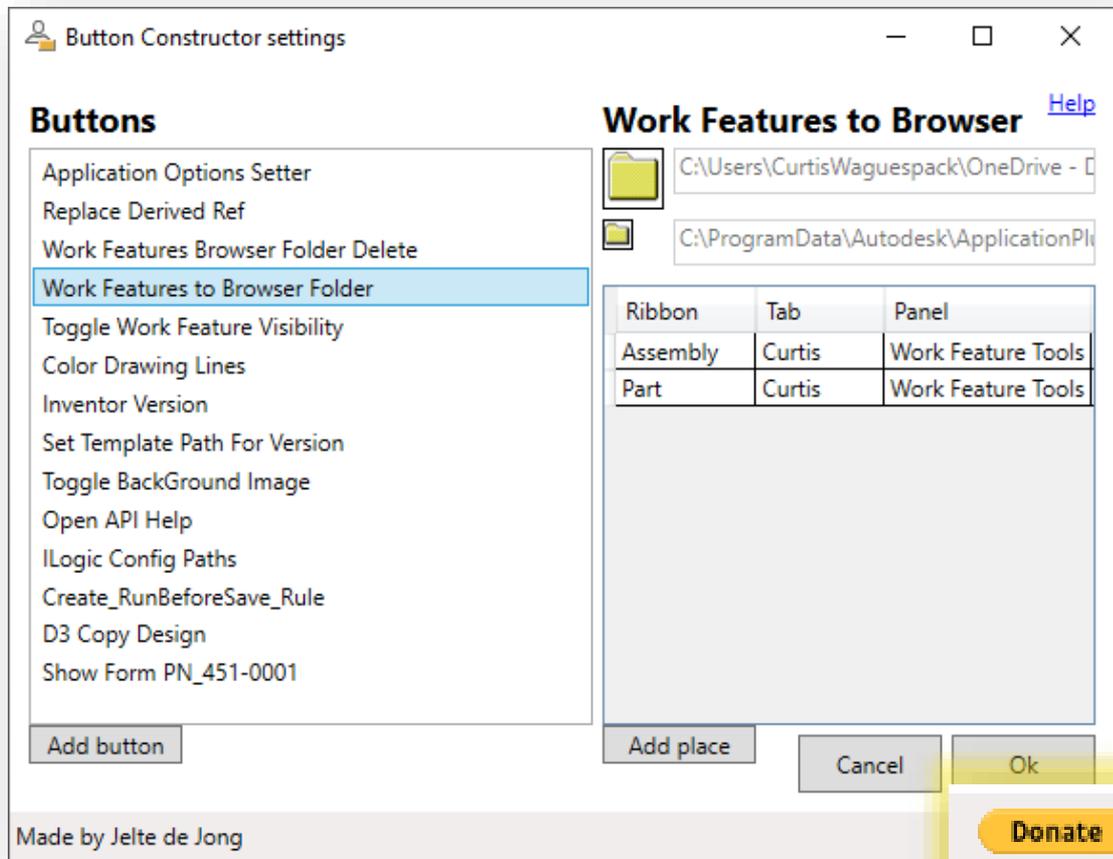
Large	Text	Name
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PDF all IDWs in folder

# Button Constructor Add in

Get it at from the  
Autodesk App Store

Created by: Jelte de Jong

[www.hjalte.nl](http://www.hjalte.nl)



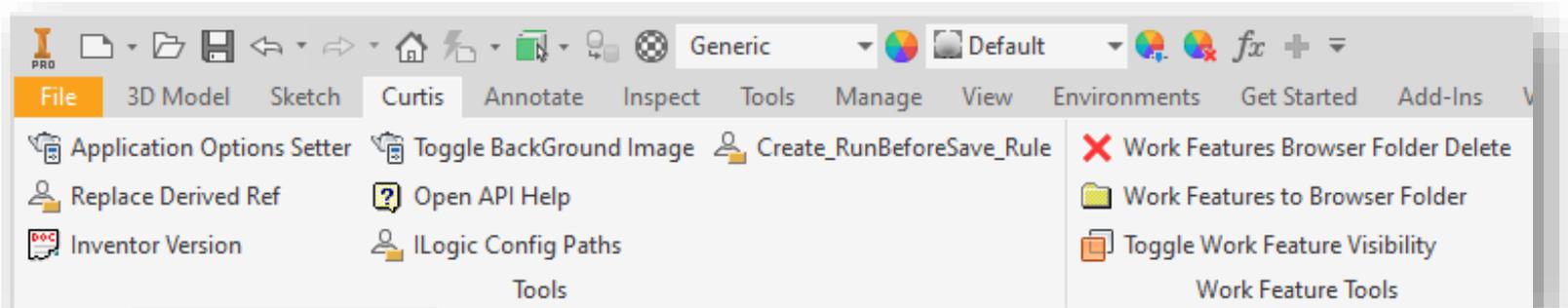
# Button Constructor Add in

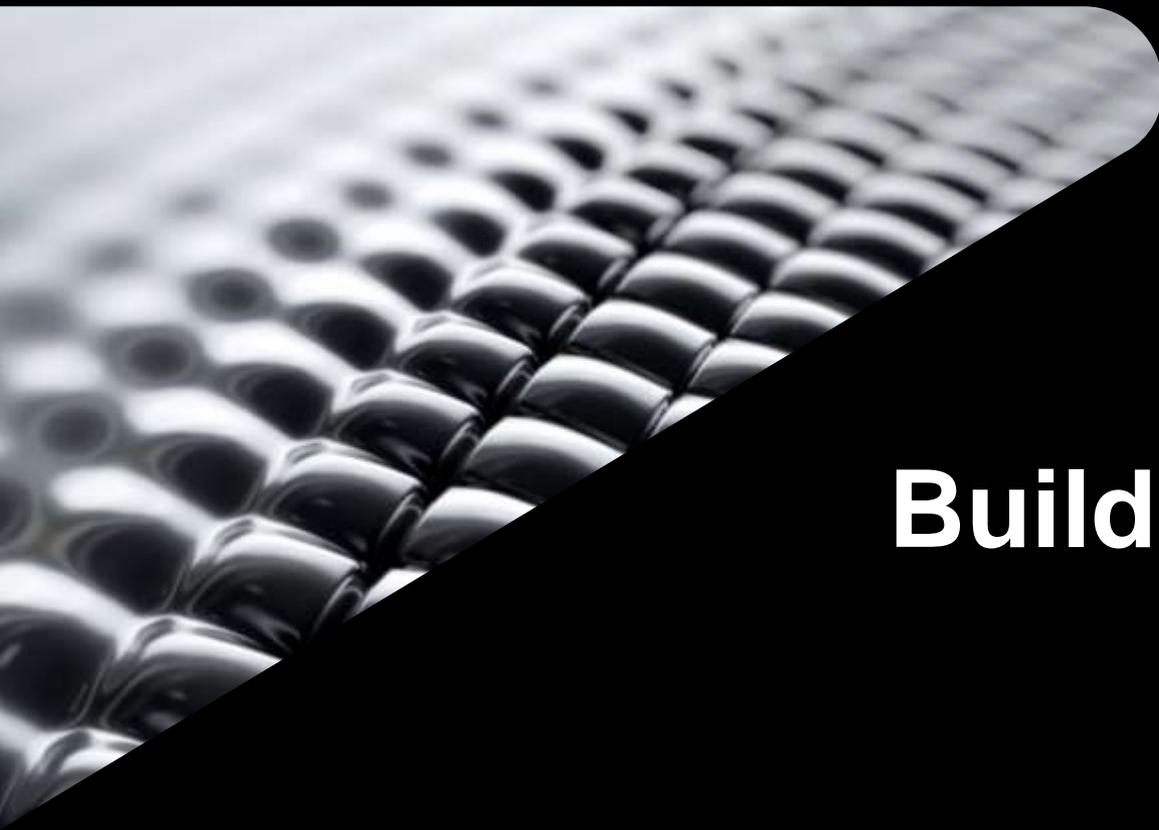
Get it at from the  
Autodesk App Store

Created by: Jelte de Jong

Example:

- Custom tab called Curtis
- Contains buttons for some of my External rules



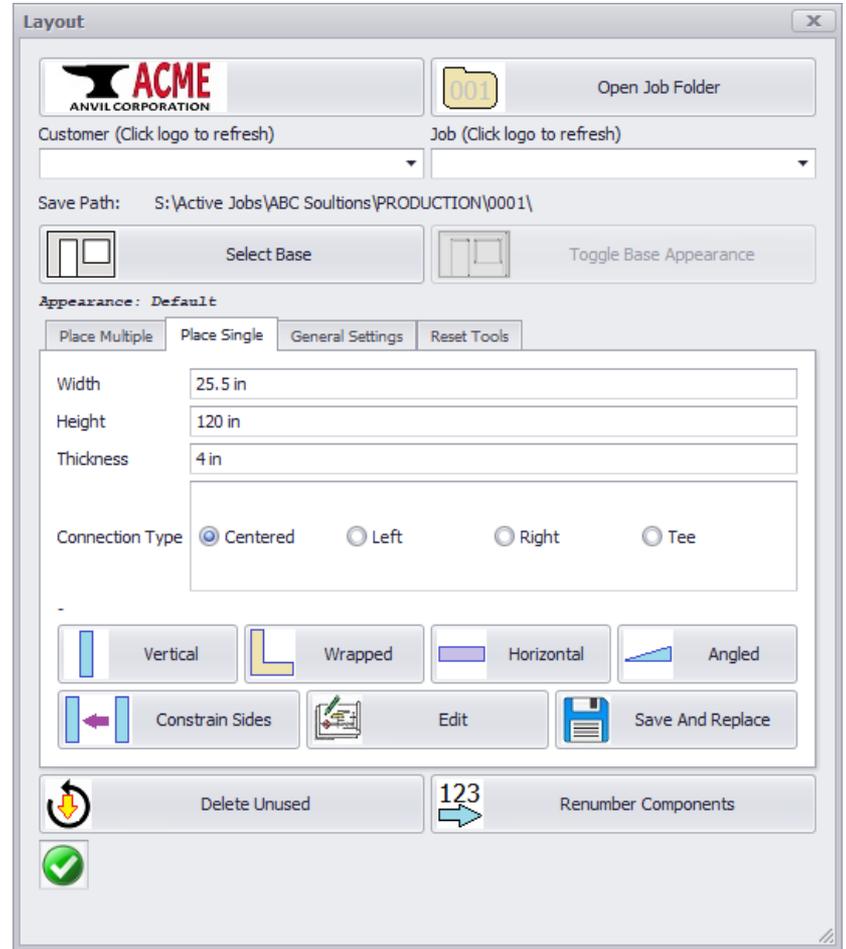
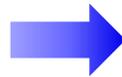


# Build Better Forms

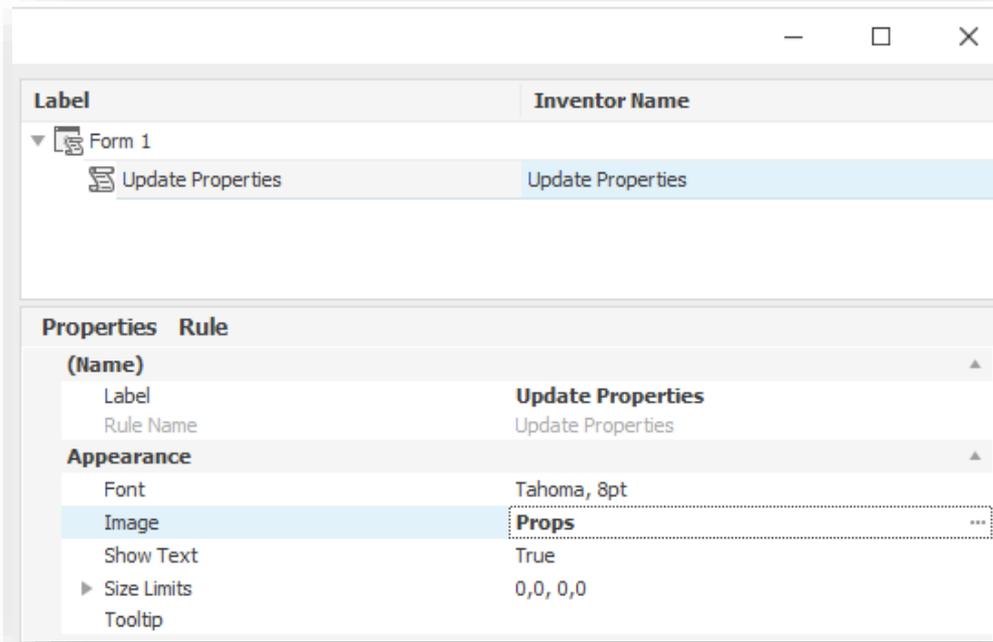
Tip #25

# Make it more visual

- Add buttons to run rules
- Add icons to the buttons

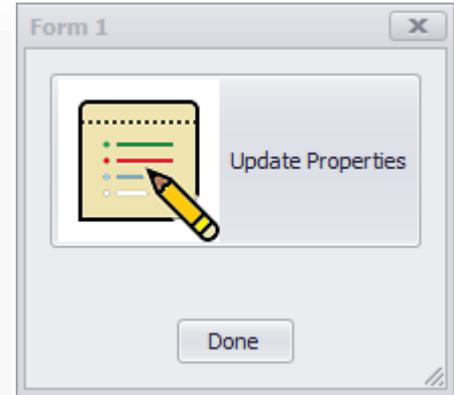


# Add icons to the buttons



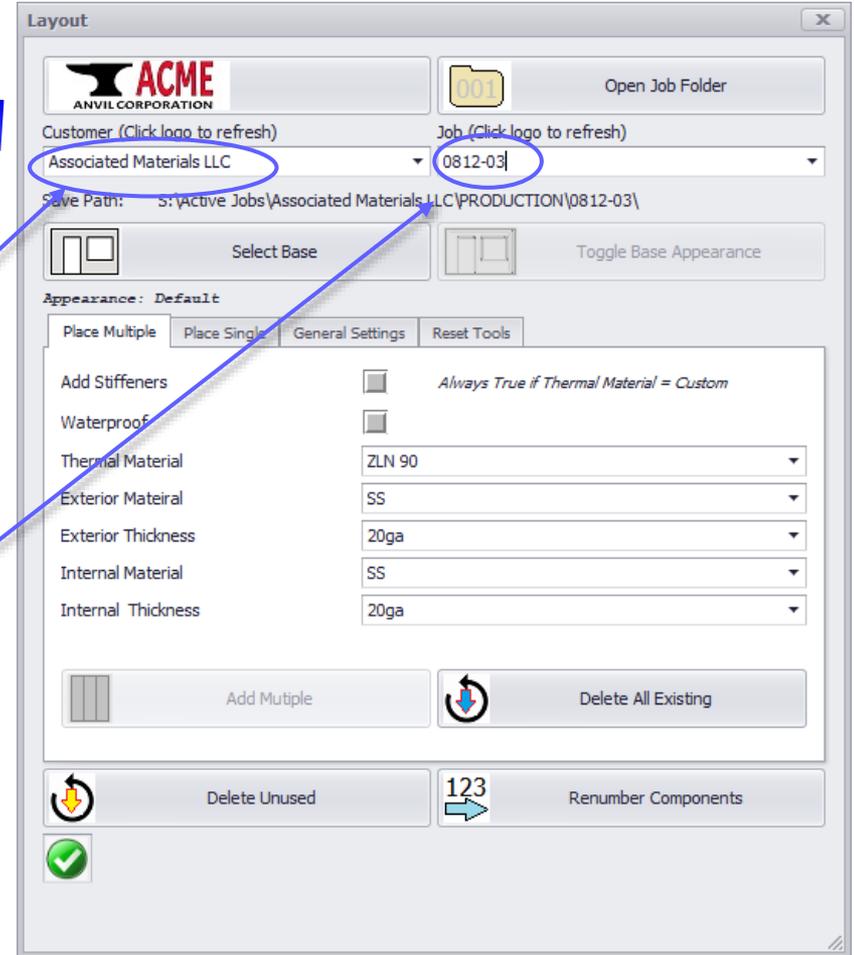
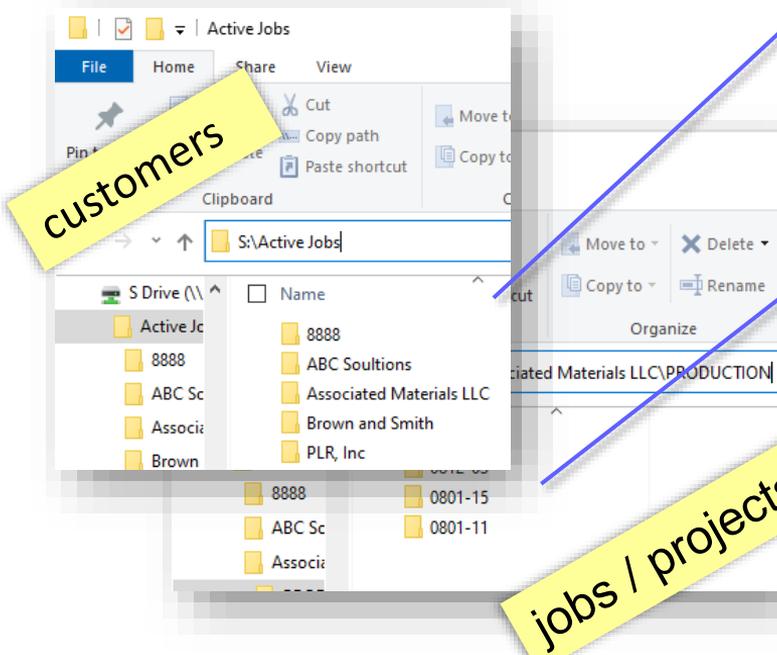
The screenshot shows the 'Properties' window for a button labeled 'Update Properties'. The 'Image' property is selected, and the 'Props' button is highlighted with a blue arrow. The 'Rule' column shows 'Update Properties'.

Properties	Rule
<b>(Name)</b>	
Label	<b>Update Properties</b>
Rule Name	Update Properties
<b>Appearance</b>	
Font	Tahoma, 8pt
Image	<b>Props</b>
Show Text	True
Size Limits	0,0, 0,0
Tooltip	



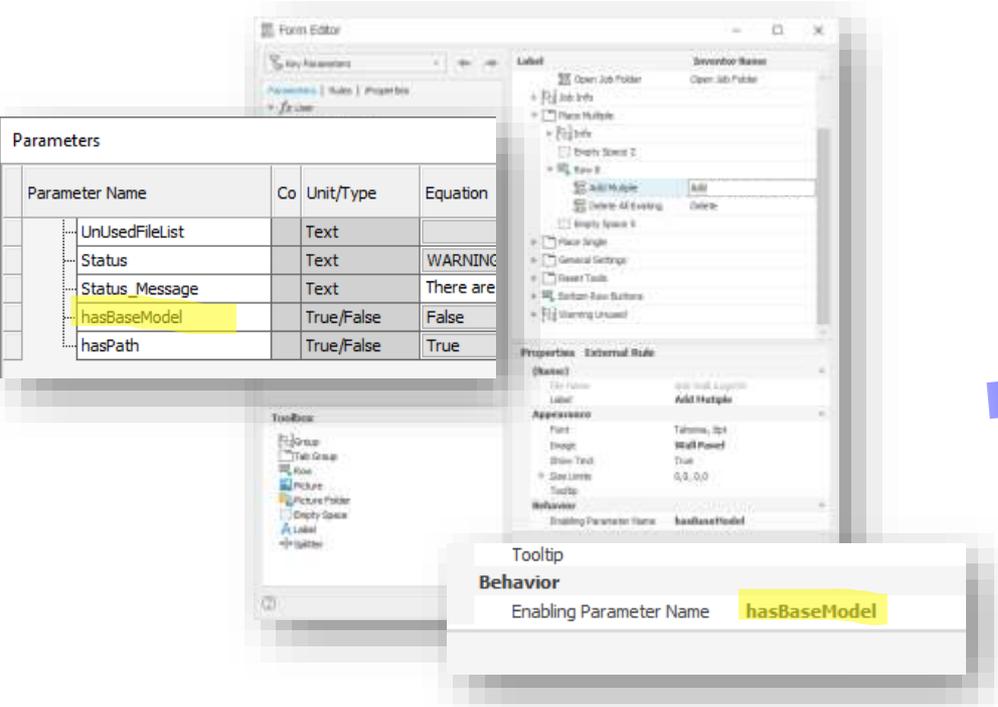
# Make it more interactive

- Button for a rule that gets customer and job folders from directory, and populates the lists
- Prevents errors by disallowing invalid input



# Make it more interactive

- Uses a True / False parameter as an **Enabling Parameter** to enable/disable the button
- Prevents errors by disallowing invalid options



Parameter Name	Co	Unit/Type	Equation
UnusedFileList		Text	
Status		Text	WARNING
Status_Message		Text	There are
hasBaseModel		True/False	False
hasPath		True/False	True

Form Editor

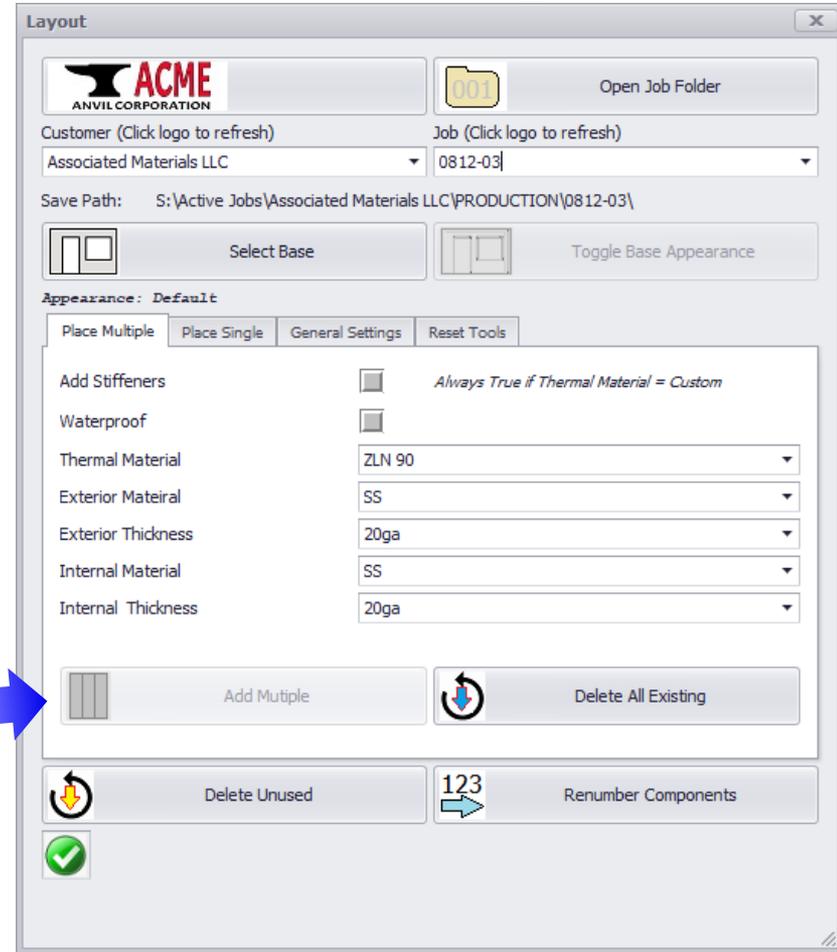
Label

Serverdoor Base

Properties - External Rule

Behavior

Enabling Parameter Name: **hasBaseModel**



Layout

ACME ANVIL CORPORATION

001 Open Job Folder

Customer (Click logo to refresh) Job (Click logo to refresh)

Associated Materials LLC 0812-03

Save Path: S:\Active Jobs\Associated Materials LLC\PRODUCTION\0812-03\

Select Base Toggle Base Appearance

Appearance: Default

Place Multiple Place Single General Settings Reset Tools

Add Stiffeners  Always True if Thermal Material = Custom

Waterproof

Thermal Material ZLN 90

Exterior Material SS

Exterior Thickness 20ga

Internal Material SS

Internal Thickness 20ga

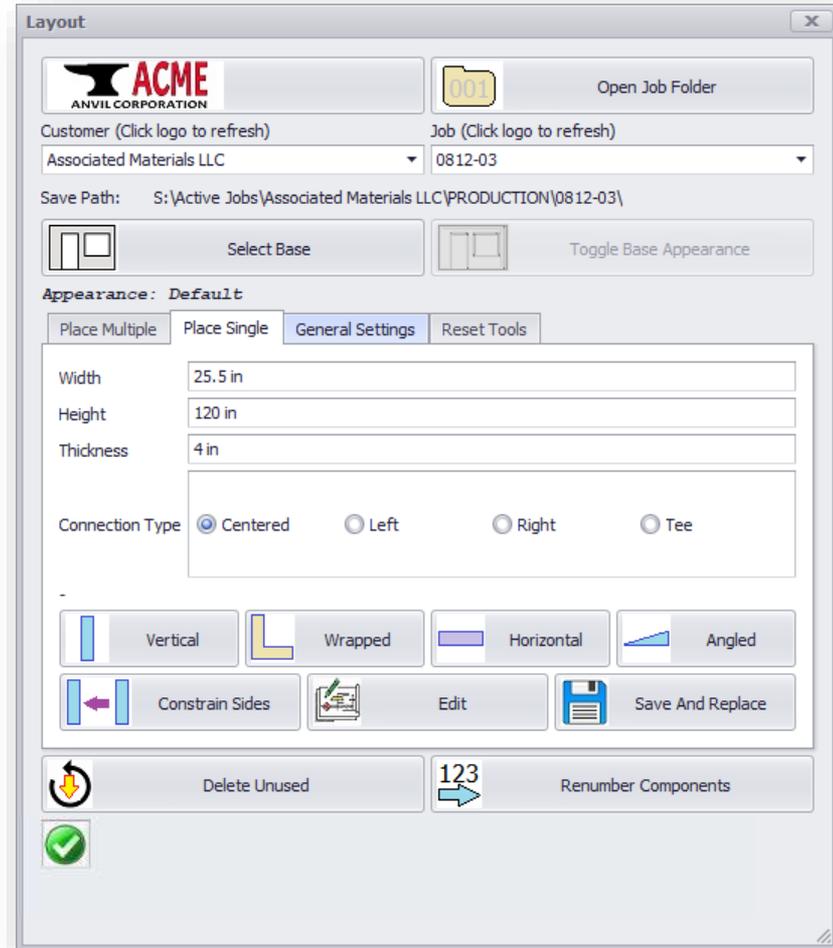
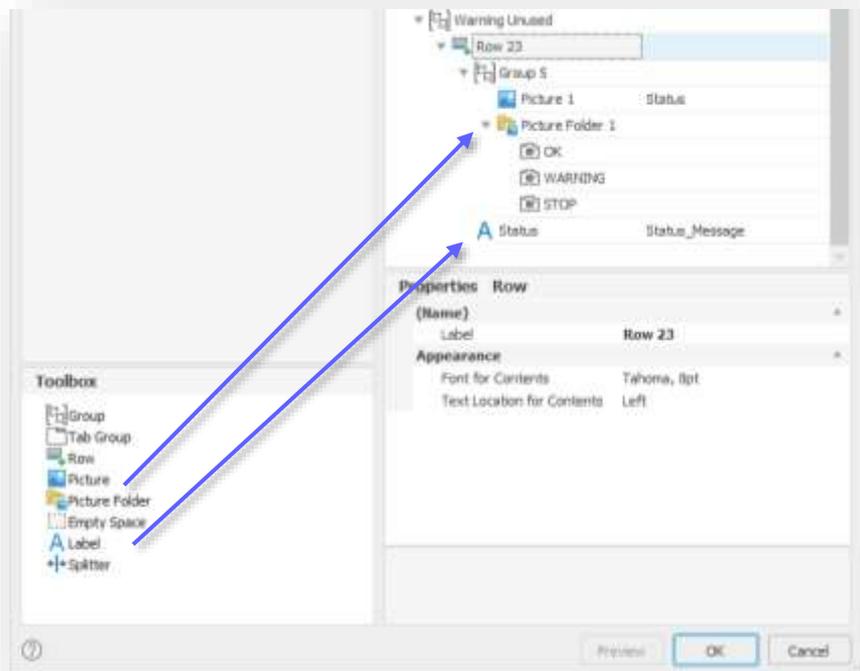
Add Multiple Delete All Existing

Delete Unused 123 Renumber Components

✓

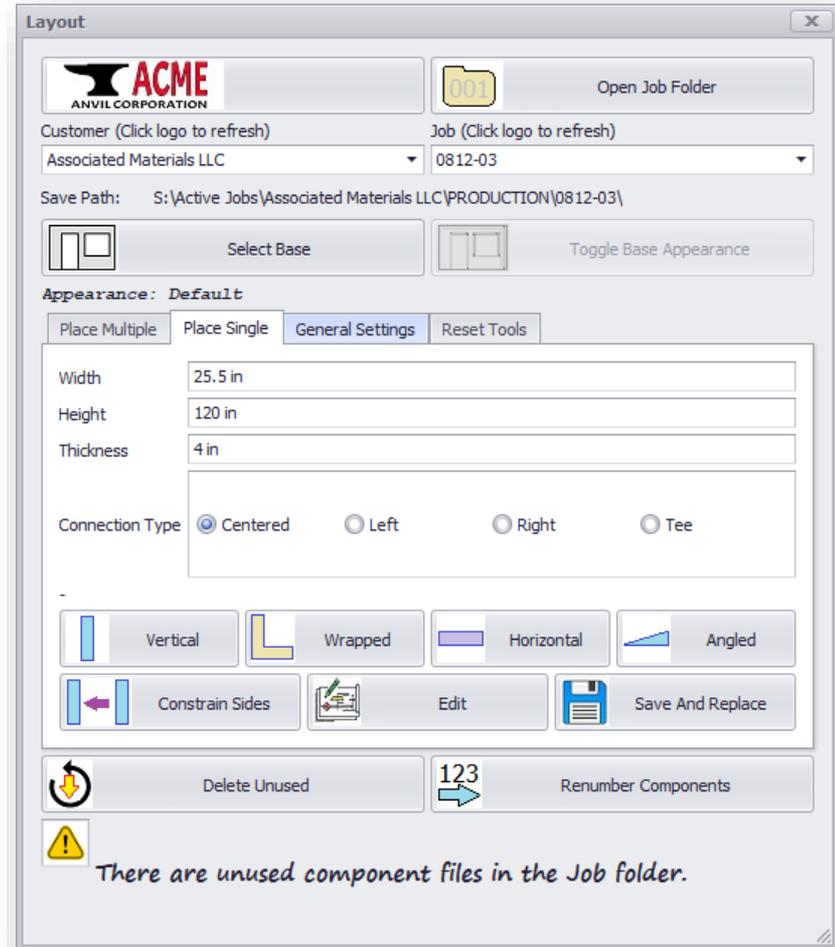
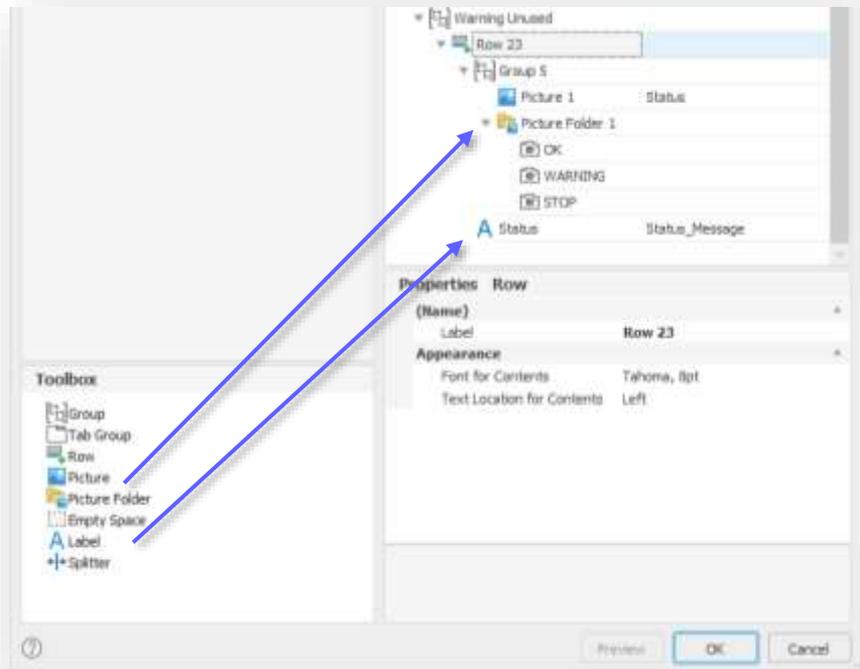
# Make it more interactive

- Uses **Picture Folders and labels** to communicate interactively with the user



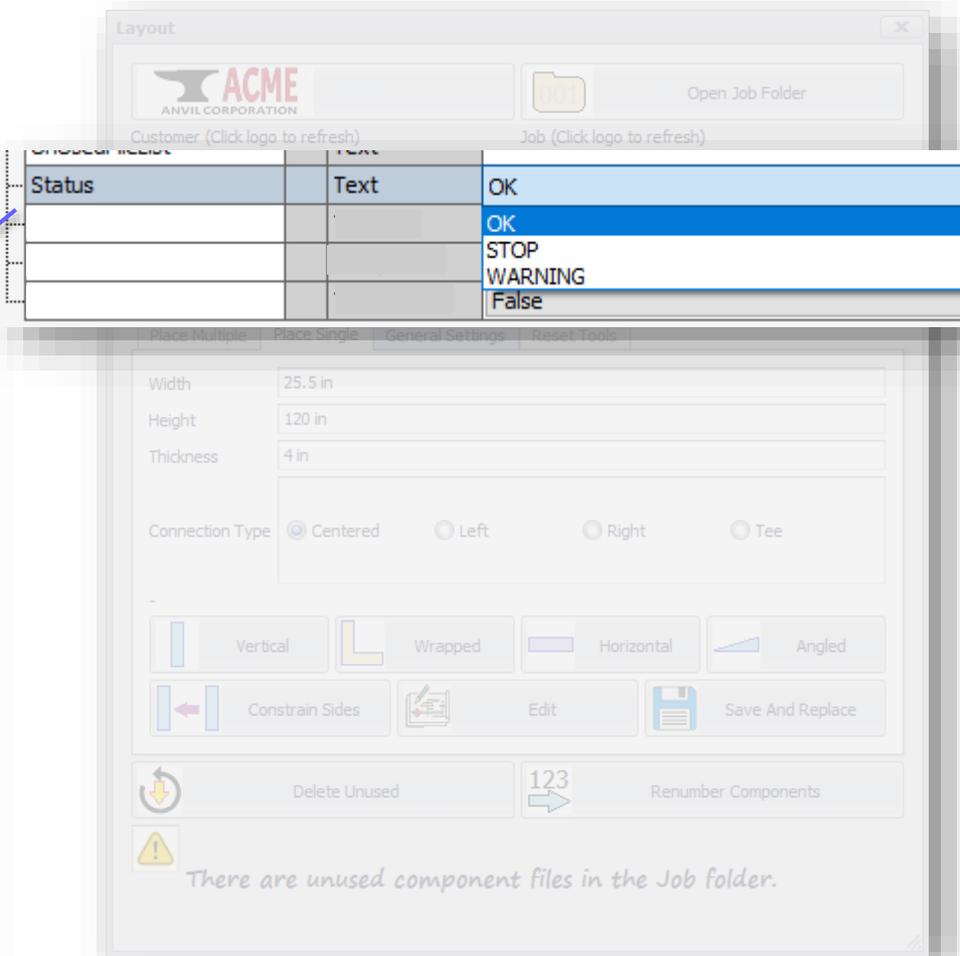
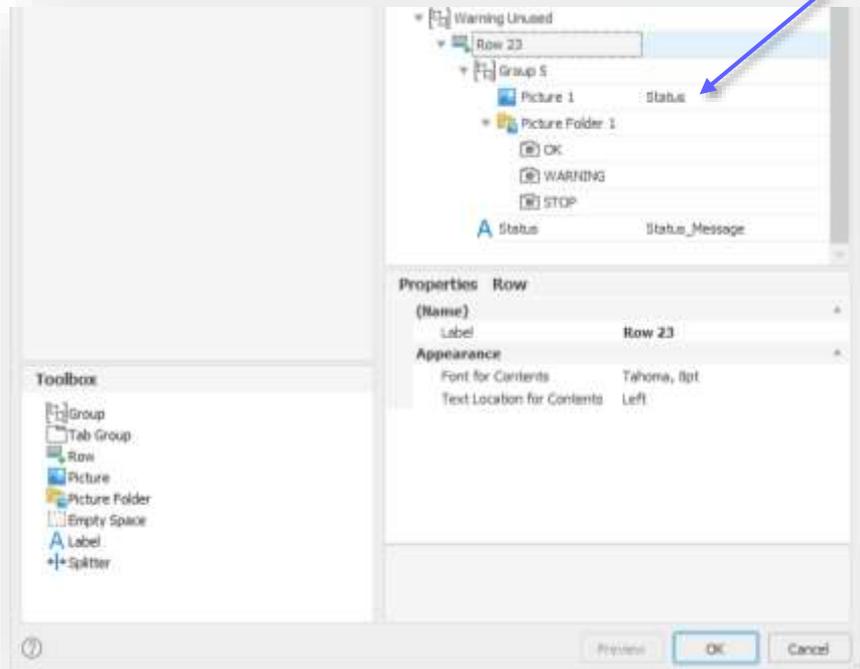
# Make it more interactive

- Uses Picture Folders and labels to communicate interactively with the user



# Make it more interactive

- Uses Picture Folders and labels to communicate interactively with the user

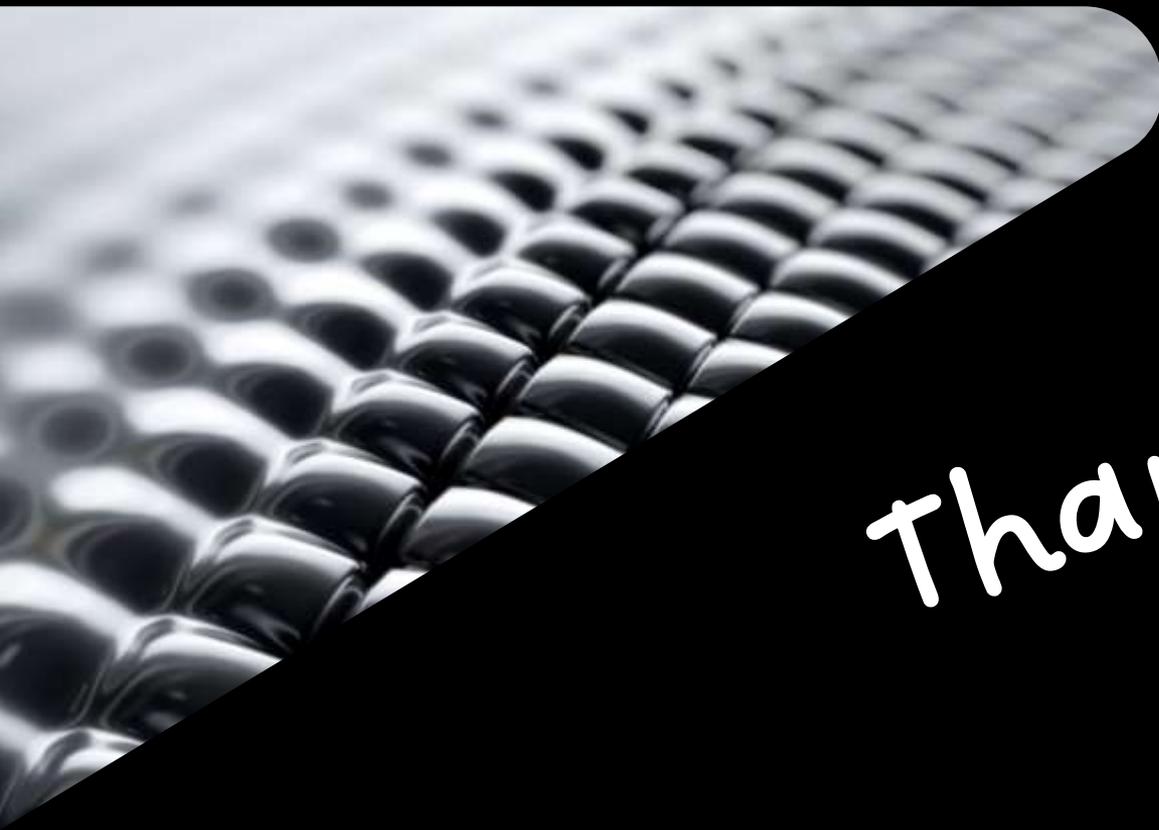




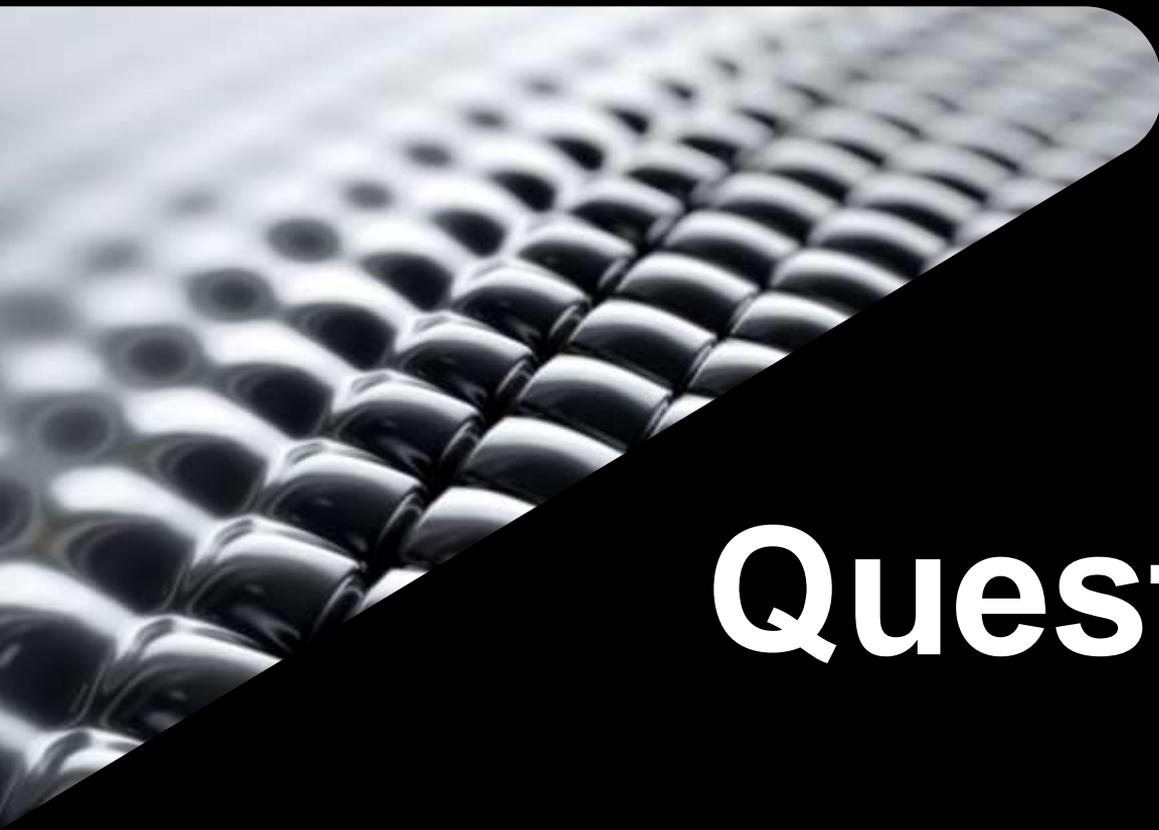
**FINISH LINE IN SIGHT**







Thank you!



**Questions?**



Autodesk and the Autodesk logo are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product offerings, specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document.

© 2022 Autodesk. All rights reserved.