AUTODESK UNIVERSITY

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

MFG501293

Curtis Waguespack
Automation Solution Consultant

AUTODESK UNIVERSITY

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.

AUTODESK UNIVERSITY

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

AUTODESK UNIVERSITY

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

AUTODESK UNIVERSITY

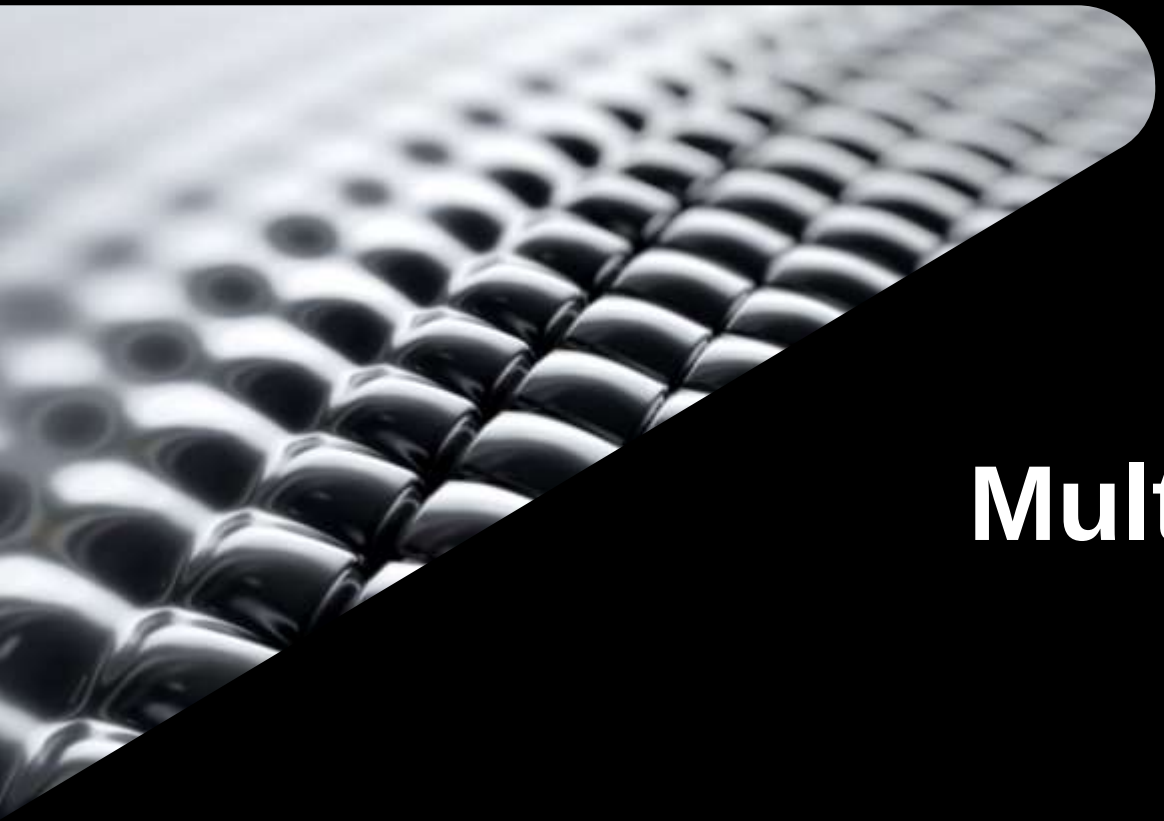
Objectives

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

AUTODESK UNIVERSITY

Objective:

Tips and Techniques Using the iLogic Editor



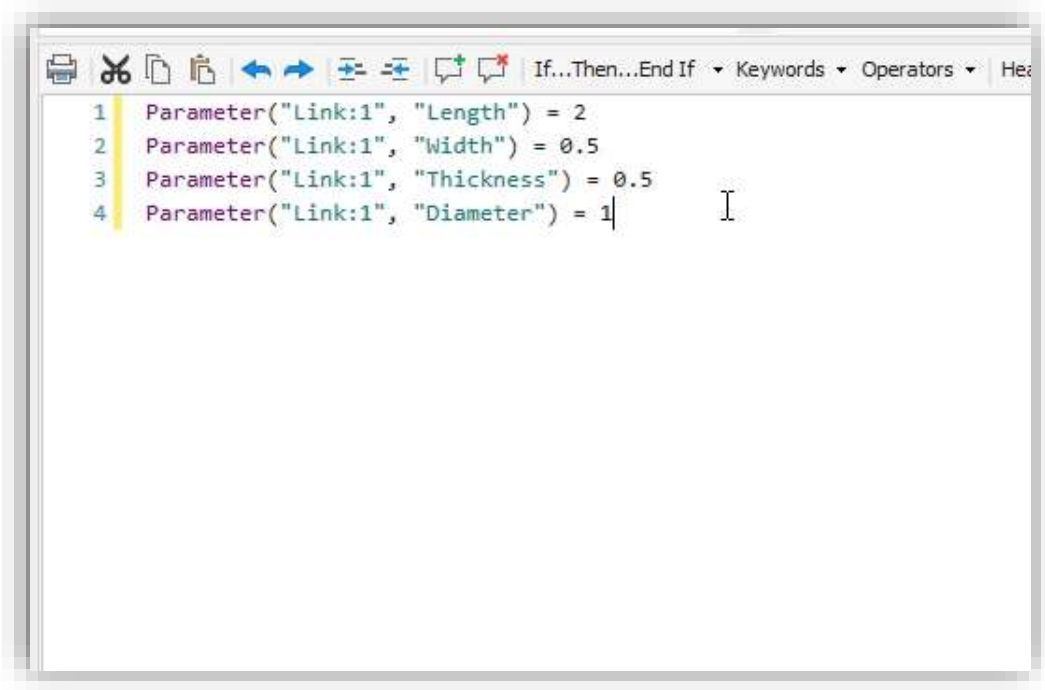
Multi-Line Typing

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

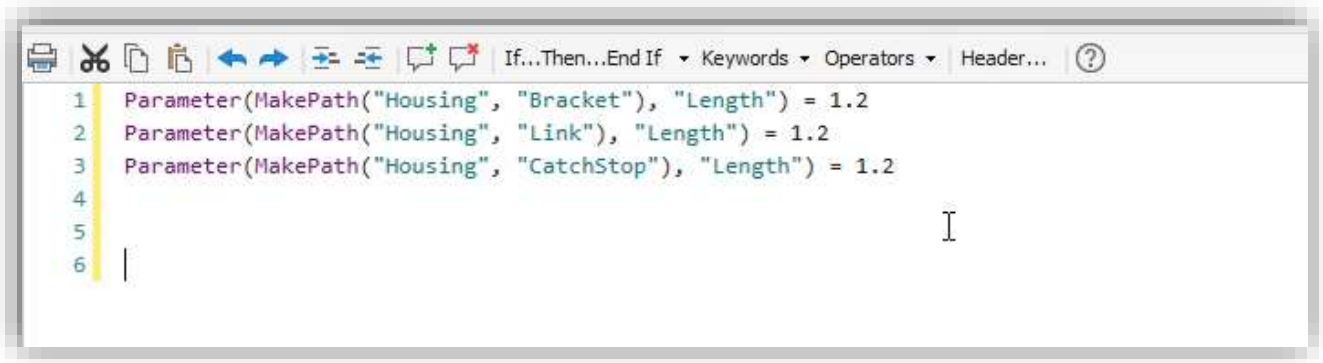
Tip provided by **Scott Hallmark**



Multi-location line editing

- We can do this in multiple

places within the same line as well

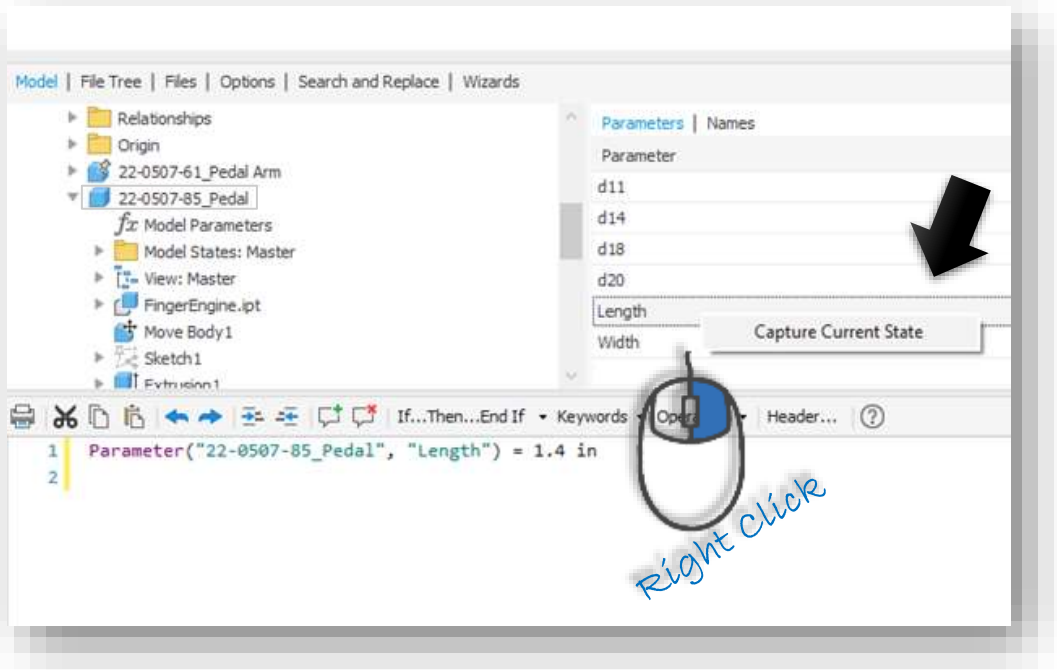


Extract to be exact!

Tip #2

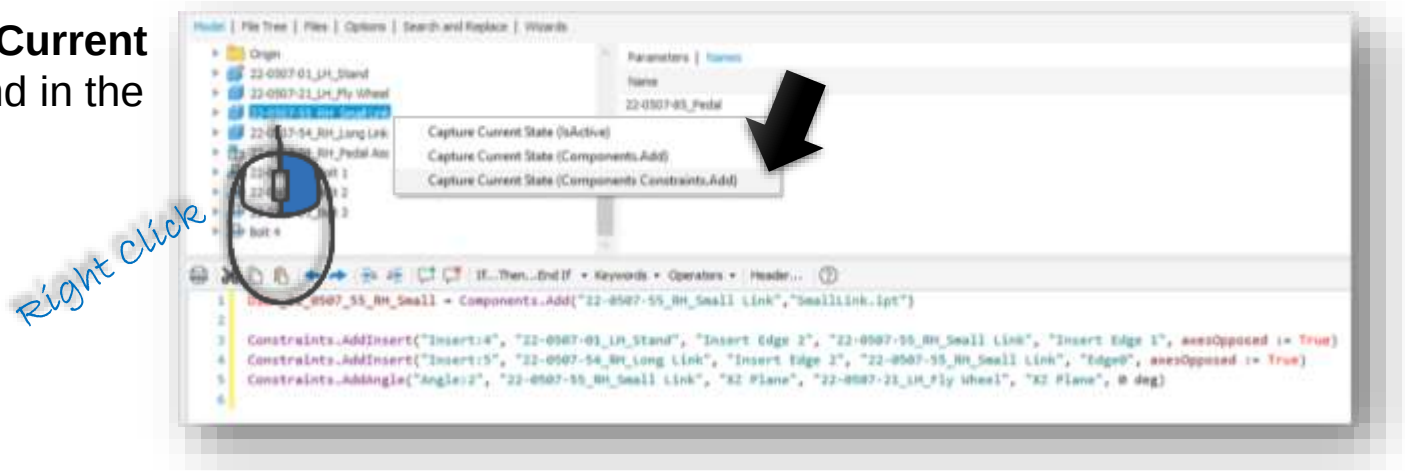
Extracting a Parameter from a sub-component

- 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



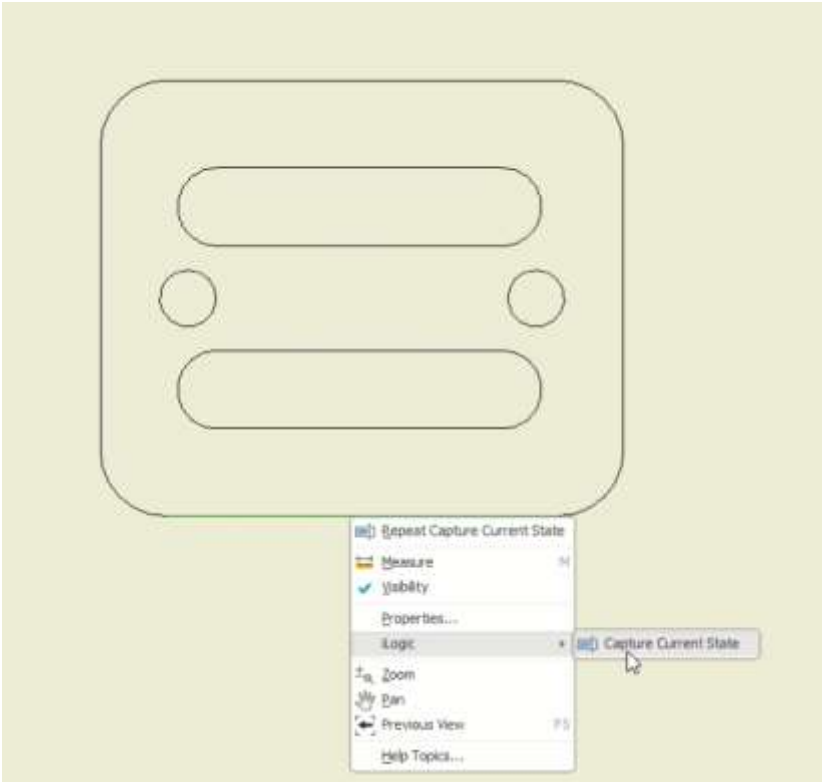
Extracting the code to place and constrain a sub-component

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



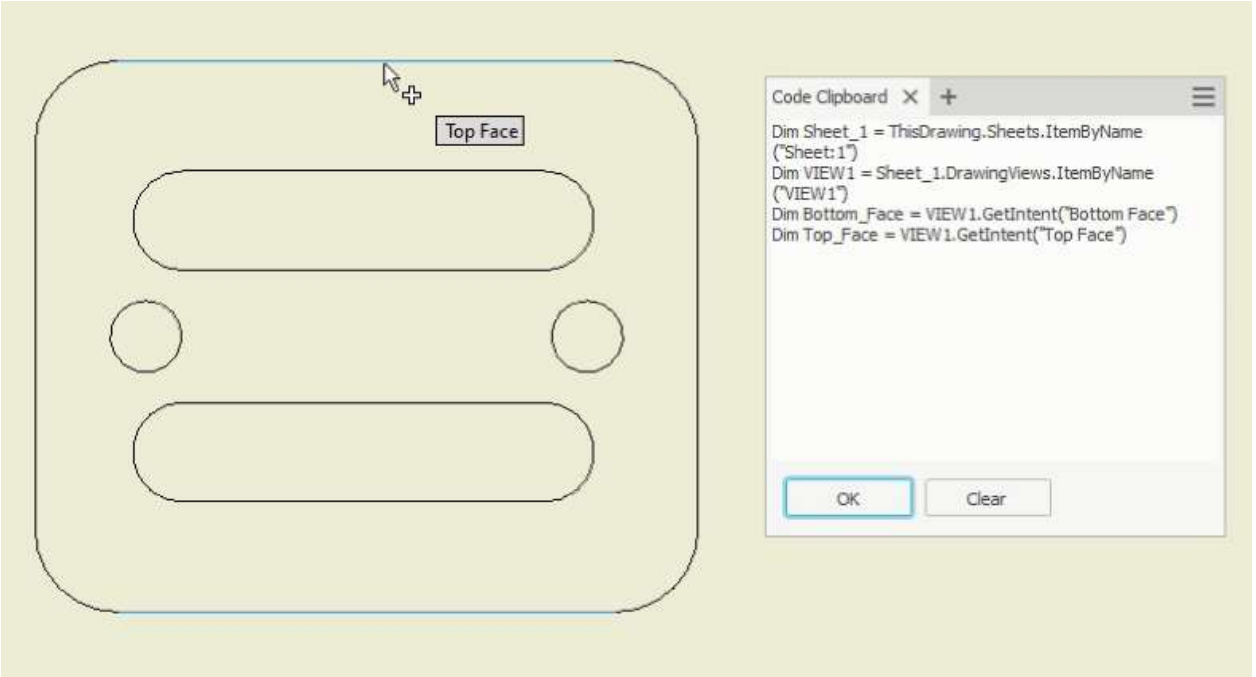
Extract code from drawing edges

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



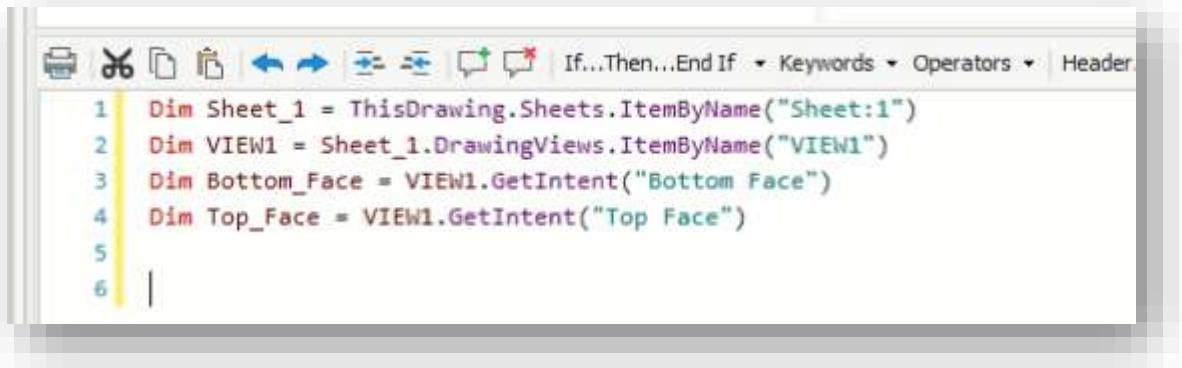
Extract code from drawing edges

- 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

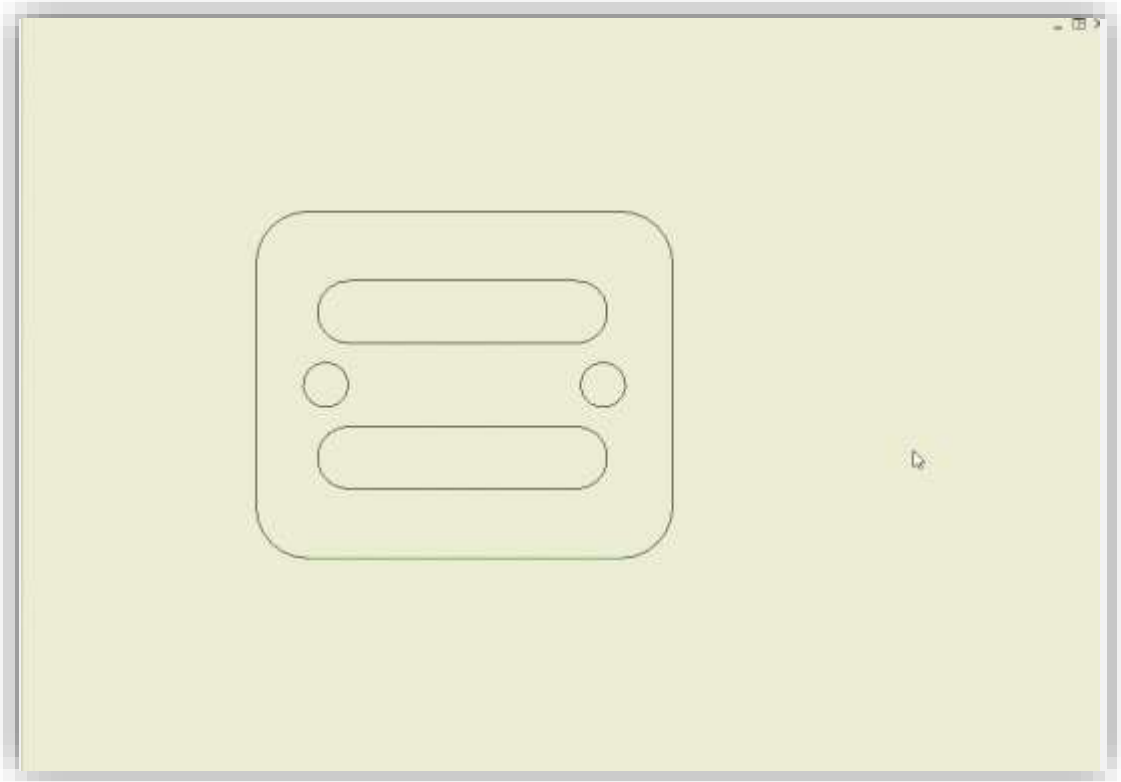


Paste the extracted code into a rule

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

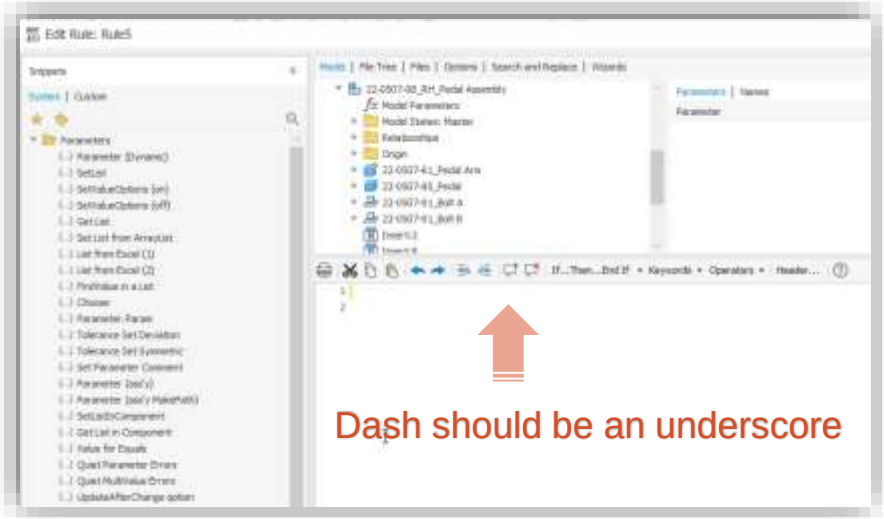


Extracting code to place dimensions

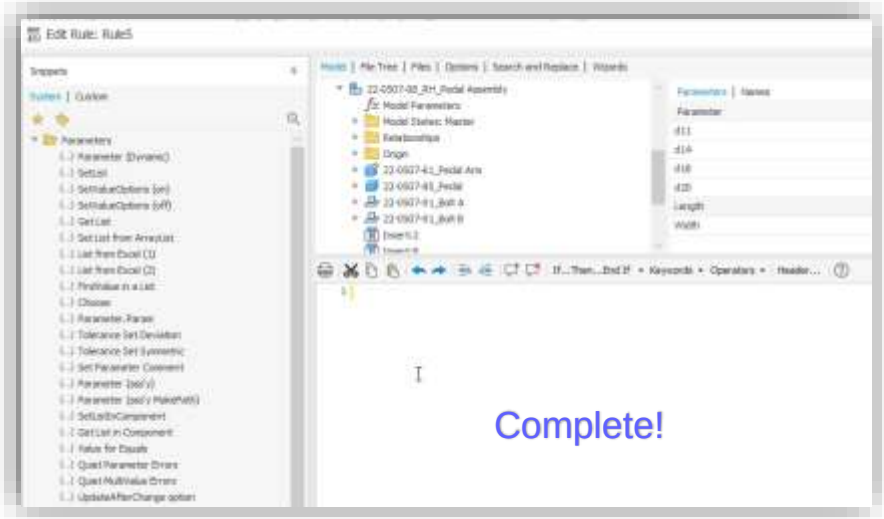


Extracting Comparison

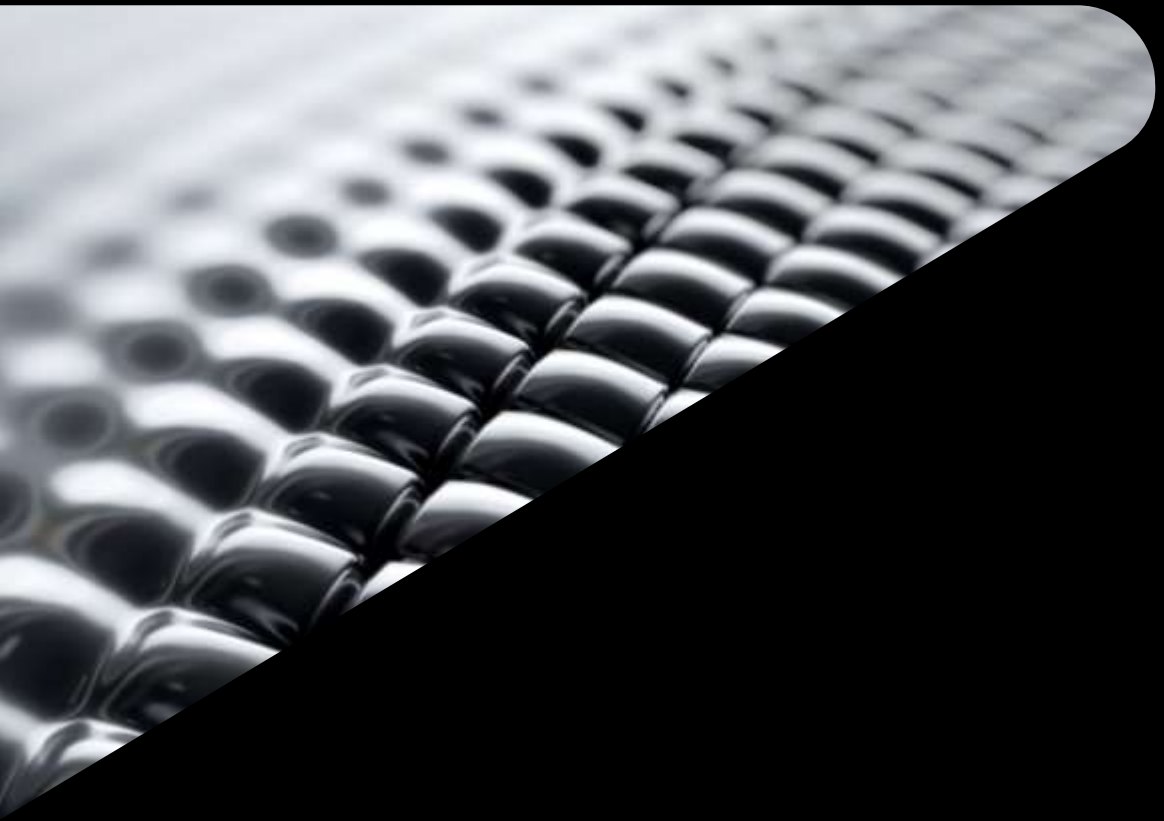
Typing vs extracting



Typing the code (with typo)



Extracting the code

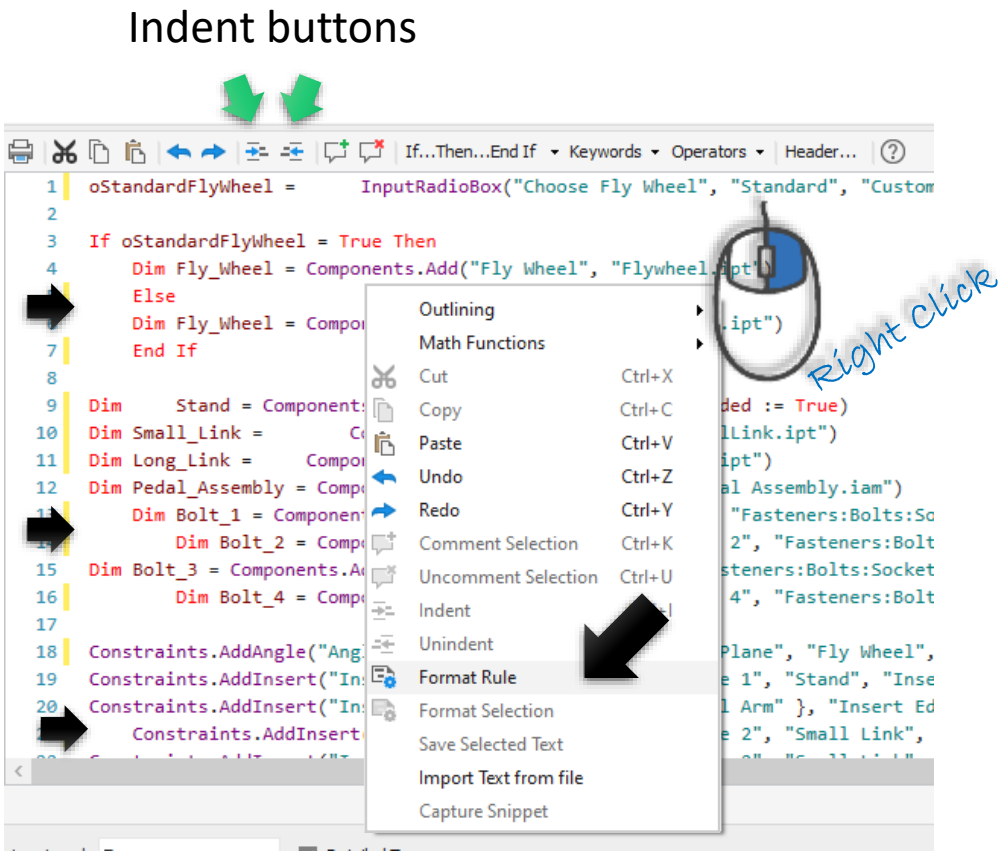


Right Click: Format Rule

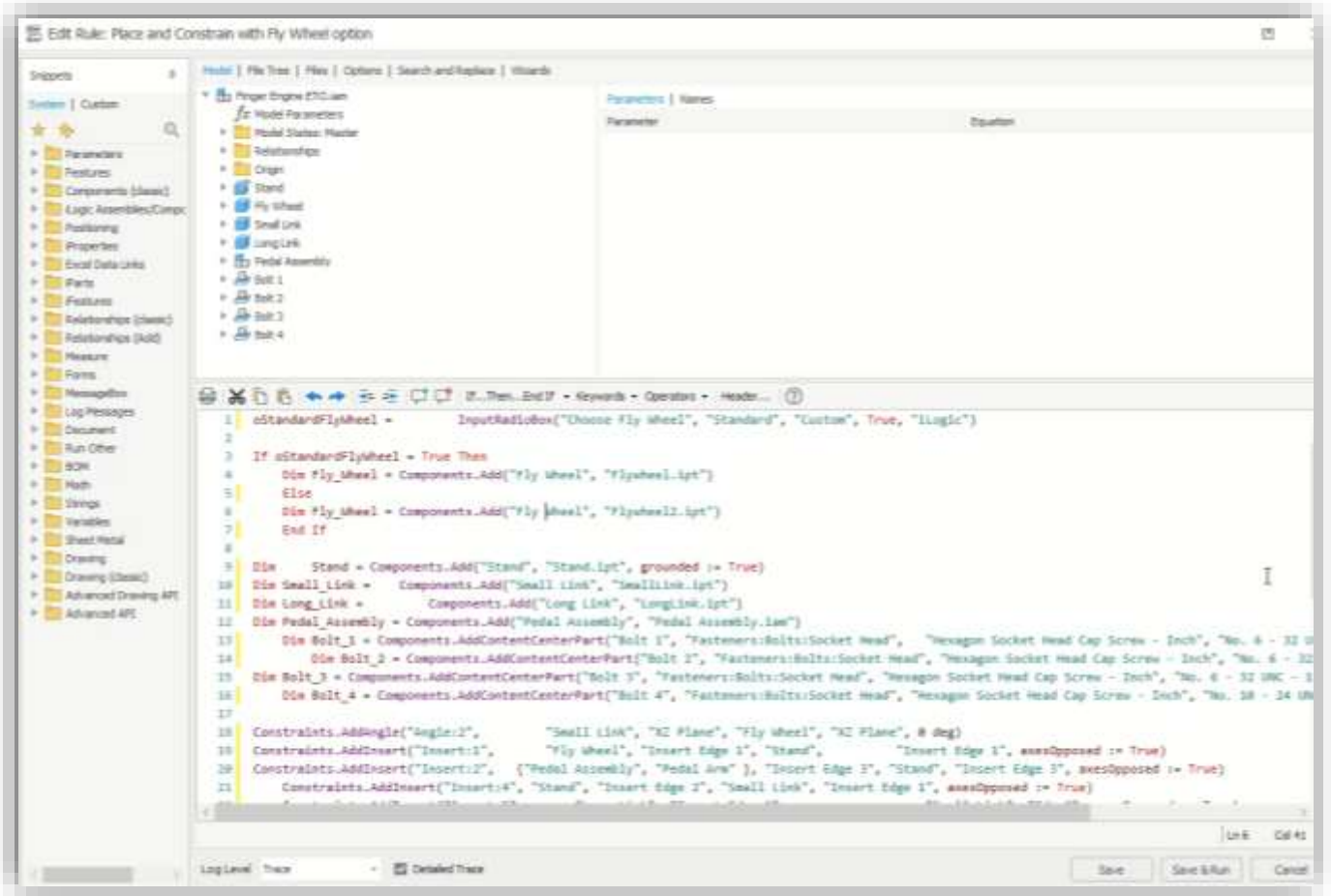
Tip #3

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



Format Rule


Edit Rule: Place and Constrain with Fly Wheel option
oStandardFlyWheel = InputRadioBox("Choose Fly Wheel", "Standard", "Custom", True, "iLogic")
If oStandardFlyWheel = True Then
Dim Fly_Wheel = Components.Add("Fly Wheel", "Flywheel.ipt")
Else
Dim Fly_Wheel = Components.Add("Fly Wheel", "Flywheel2.ipt")
End If
Dim Stand = Components.Add("Stand", "Stand.ipt", grounded := True)
Dim Small_Link = Components.Add("Small Link", "SmallLink.ipt")
Dim Long_Link = Components.Add("Long Link", "LongLink.ipt")
Dim Pedal_Assembly = Components.Add("Pedal Assembly", "Pedal Assembly.iam")
Constraints.AddAngle("Angle:2", "Small Link", "XZ Plane", "Fly Wheel", "XZ Plane", 0 deg)
Constraints.AddInsert("Insert:1", "Fly Wheel", "Insert Edge 1", "Stand", "Insert Edge 1", axesOpposed := True)
Constraints.AddInsert("Insert:2", {"Pedal Assembly", "Pedal Arm" }, "Insert Edge 3", "Stand", "Insert Edge 3", axesOpposed := True)
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
   Else If oSize < 7 Then
12 Parameter("Link Arm", "Length") = 9
13 Parameter("Link Arm", "Width") = 3
   ElseIf oSize < 9 Then
15     Parameter("Link Arm", "Length") = 11
16     Parameter("Link Arm", "Width") = 5
17 End If
```

Add Code Structure with Sub Procedures

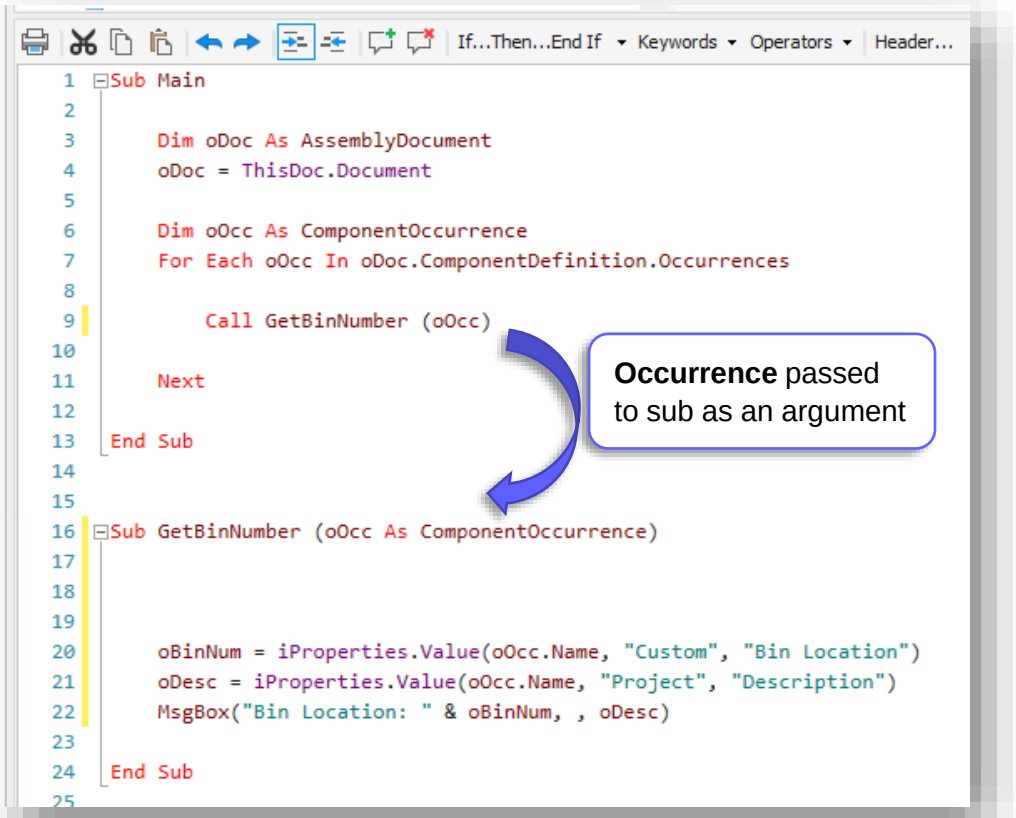
Tip #4

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

Passing information to the Sub

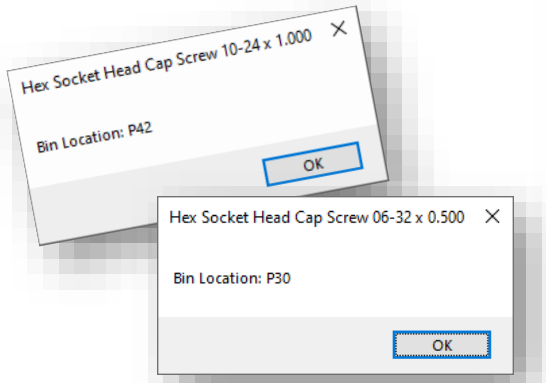
- Often, we pass information to a Sub procedure in the form of arguments
- The sub requires the argument's data type to be declared
  - technically the argument is called a parameter at the sub level
- Syntax for calling a Sub procedure:
  - Call SubName (Argument1, Argument2)
- The use of the **Call**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

Diagramming a Simple Sub Procedure

- Sub Name
- Argument/Parameter
- Sub Statement Code

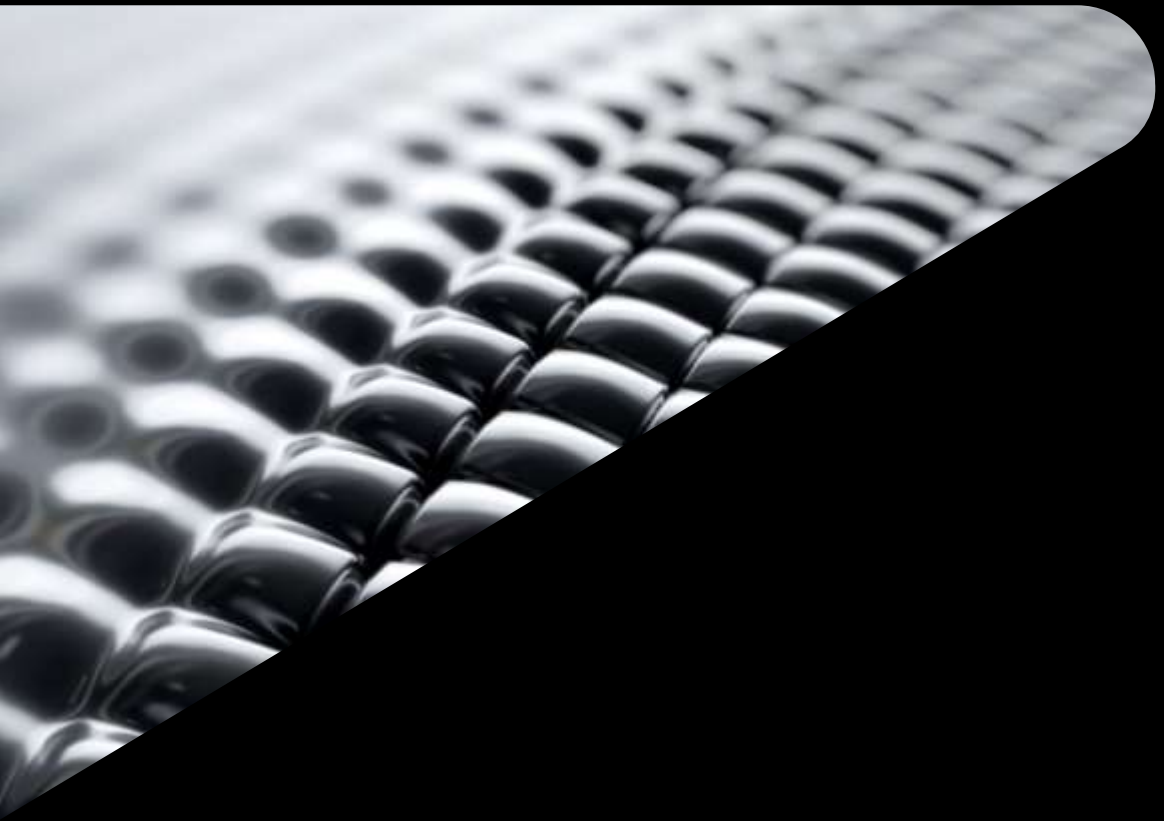



```
1  Sub Main
2
3      Dim oDoc As AssemblyDocument
4      oDoc = ThisDoc.Document
5
6      Dim oOcc As ComponentOccurrence
7      For Each oOcc In oDoc.ComponentDefinition.Occurrences
8
9          Call GetBinNumber (oOcc)
10
11     Next
12
13 End Sub
14
15
16 Sub GetBinNumber (oOcc As ComponentOccurrence)
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)...
16
17 Sub Publish_PDF(oDraw As DrawingDocument)...
20
21 Sub DXF_Out(oDraw As DrawingDocument)...
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)

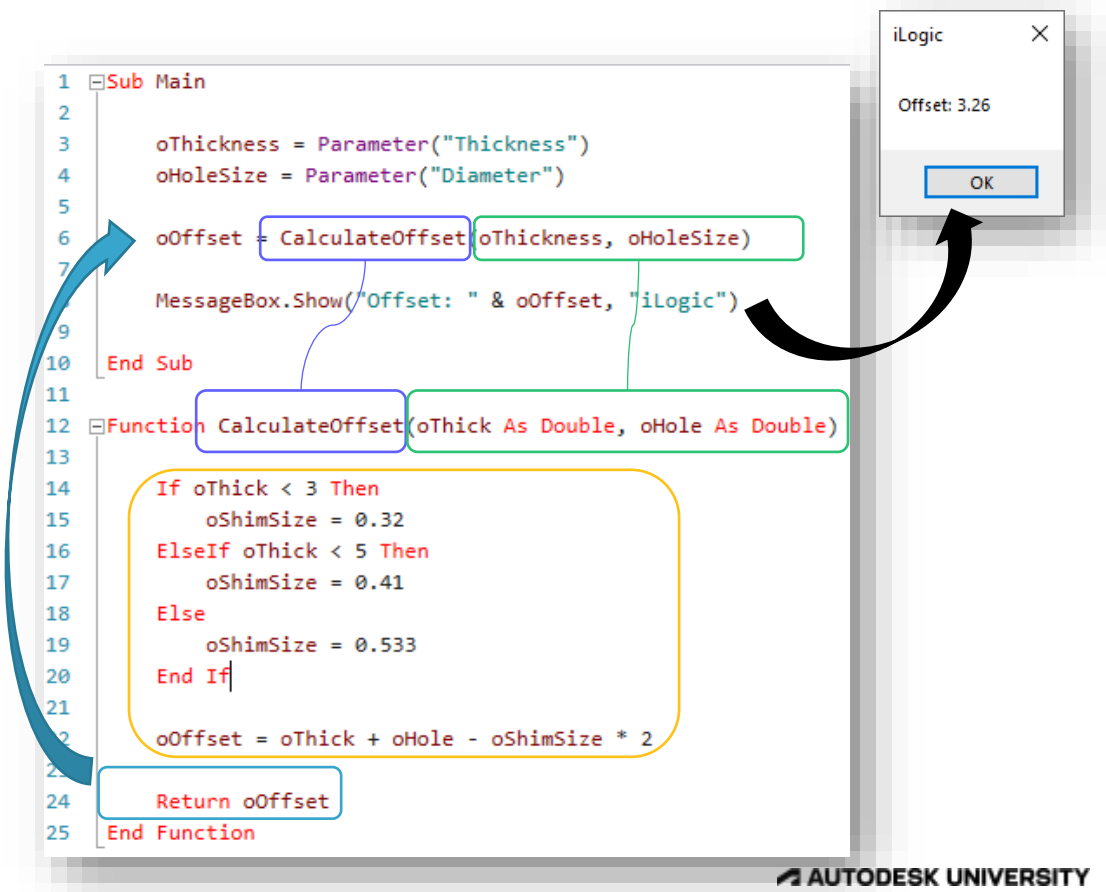
```
Sub Main

    oThickness = Parameter("Thickness")
    oHoleSize = Parameter("Diameter")

    oOffset = CalculateOffset(oThickness, oHoleSize)
    MessageBox.Show("Offset: " & oOffset, "iLogic")

End Sub

Function CalculateOffset (oThick As Double , oHole As Double)

    If oThick < 3 Then
        oShimSize = 0.32
    ElseIf oThick < 5 Then
        oShimSize = 0.41
    Else
        oShimSize = 0.533
    End If

    oOffset = oThick + oHole - oShimSize*2
    Return oOffset
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



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”

```
Class ThisRule

    Shared oOcc As ComponentOccurrence
    Shared oBinNum As String

    Sub Main

        Dim oDoc As AssemblyDocument
        oDoc = ThisDoc.Document

        For Each oOcc In oDoc.ComponentDefinition.Occurrences

            'call the sub routine to get bin number
            Call GetBinNumber

            oDesc = iProperties.Value(oOcc.Name, "Project", "Description")

            MsgBox("Bin Location: " & oBinNum,, oDesc)
        Next

    End Sub

    Sub GetBinNumber

        oBinNum = iProperties.Value(oOcc.Name, "Custom", "Bin Location")

    End Sub

End Class
```

Occurrence is held at the rule level

Value 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 nu
        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

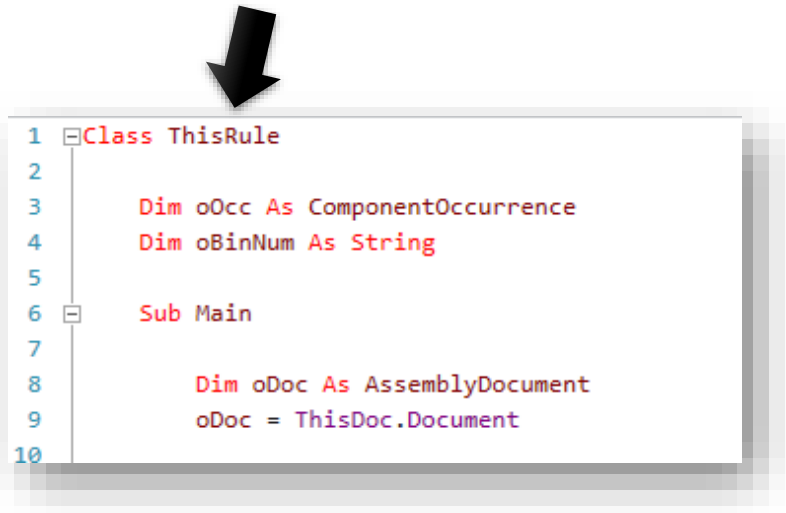
Be Aware:

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

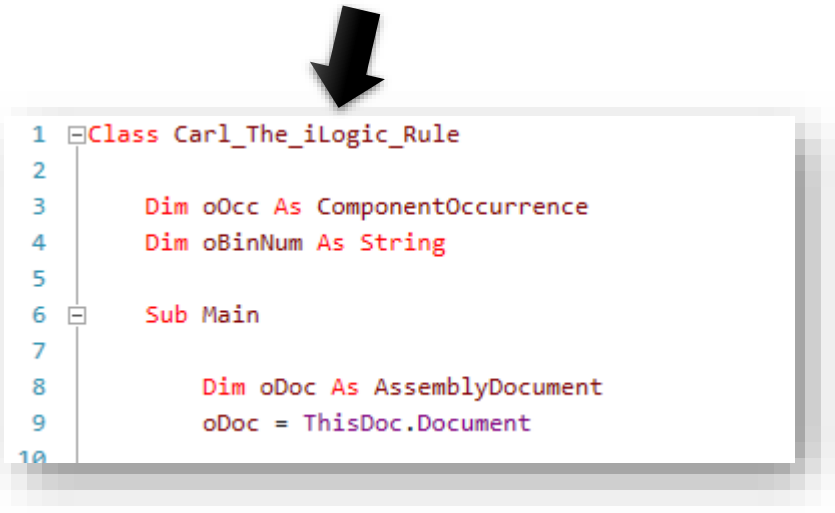
```
1  Class ThisRule
2
3      Shared oDocName As String
4
5      Sub Main
6
7          oDocName = ThisDoc.FileName
8
9          Call GetPartNumber
10         Call GetRelatedComps
11         Call CalcOffset
12
13         Call GetPartNumber
14         Call GetRelatedComps
15         Call CalcOffset
16
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

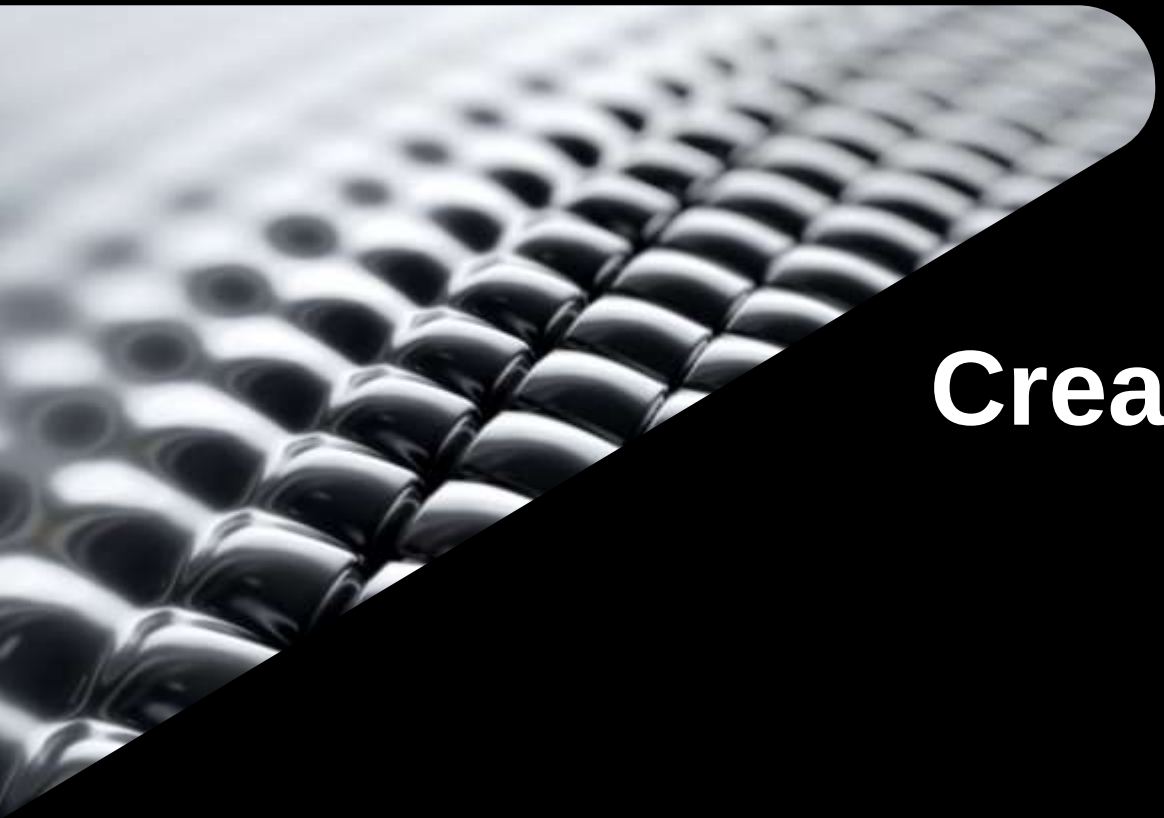
- We typically just call the class ThisRule
- But it can be named anything you like
- Both examples work the same:



```
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
```

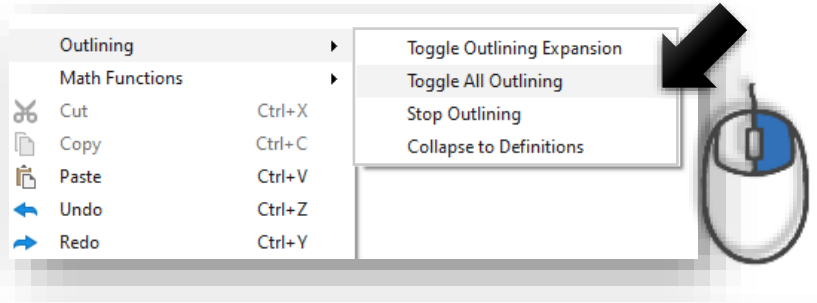
```
1
2   Dim oDoc As AssemblyDocument
3   oDoc = ThisDoc.Document
4
5   For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7       Get iProps
14
15      Write to logger
23  Next
```

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 ()

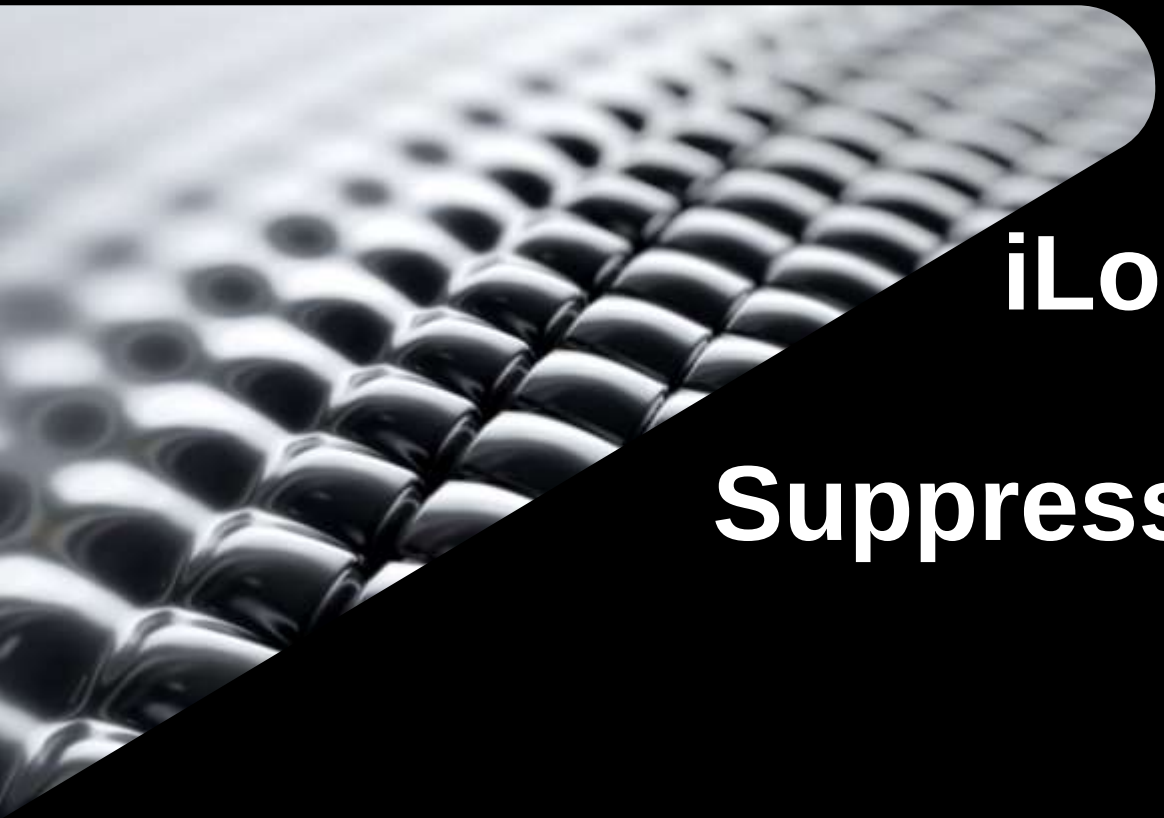
```
1   Sub Main()
2
3   Dim oDoc As AssemblyDocument
4   oDoc = ThisDoc.Document
5
6   For Each oOcc In oDoc.ComponentDefinition.Occurrences
7
8       Get iProps
15
16      Write to logger
24  Next
25
26  End Sub
```

With parenthesis, no outlining for Main sub

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
```

Without parenthesis, outlining included for Main sub



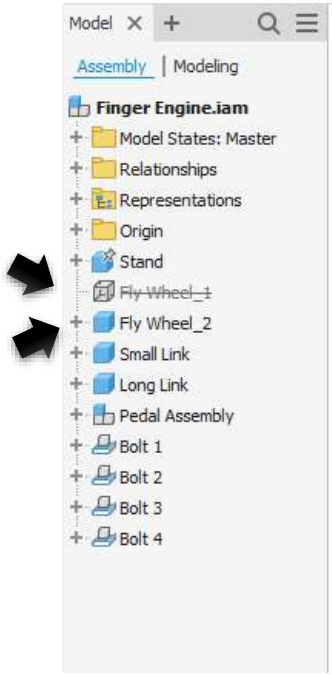
iLogic Place and Constrain vs Suppress/Unsuppress

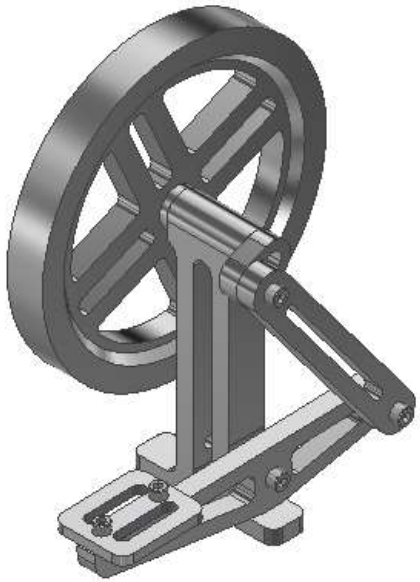
Tip #8

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

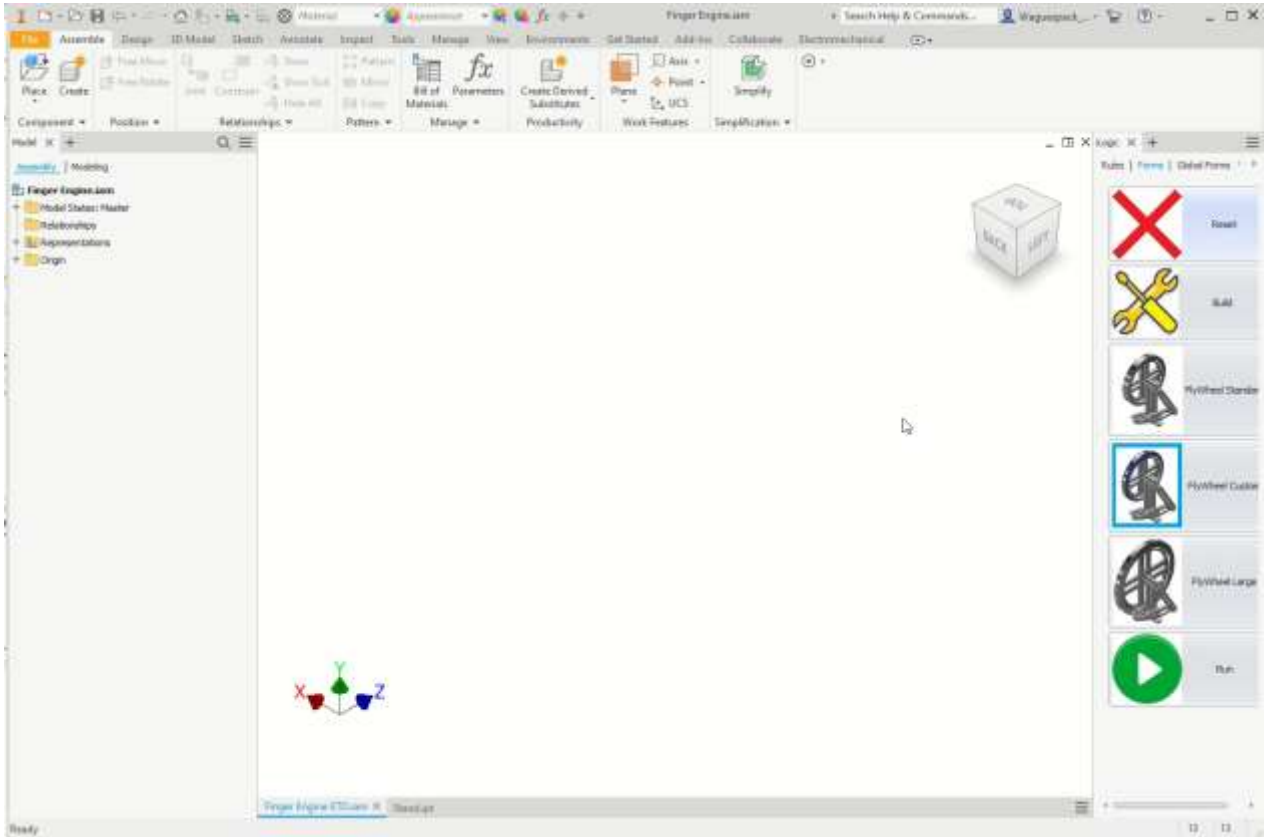
NEW ILOGIC APPROACH

- 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

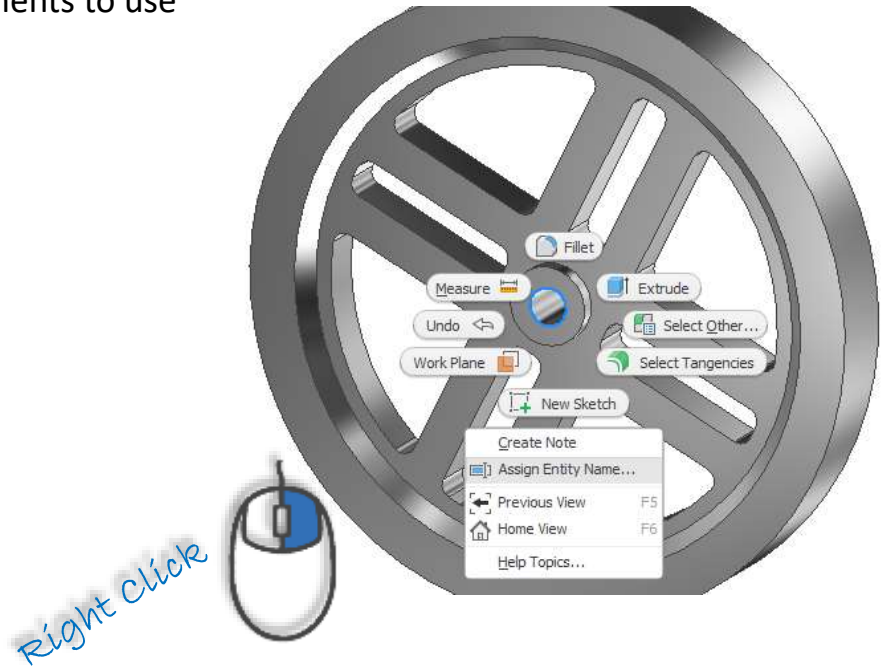
- The advantage is there are is no need to manage suppressed or invisible components for drawings, Bill of Materials, etc.

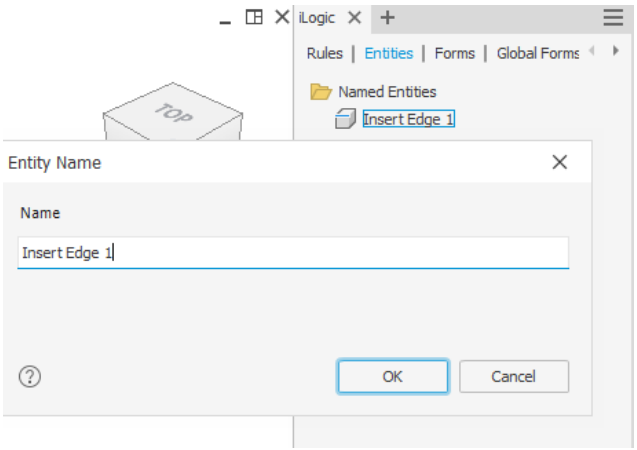


Place and Constrain

NEW ILOGIC APPROACH

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

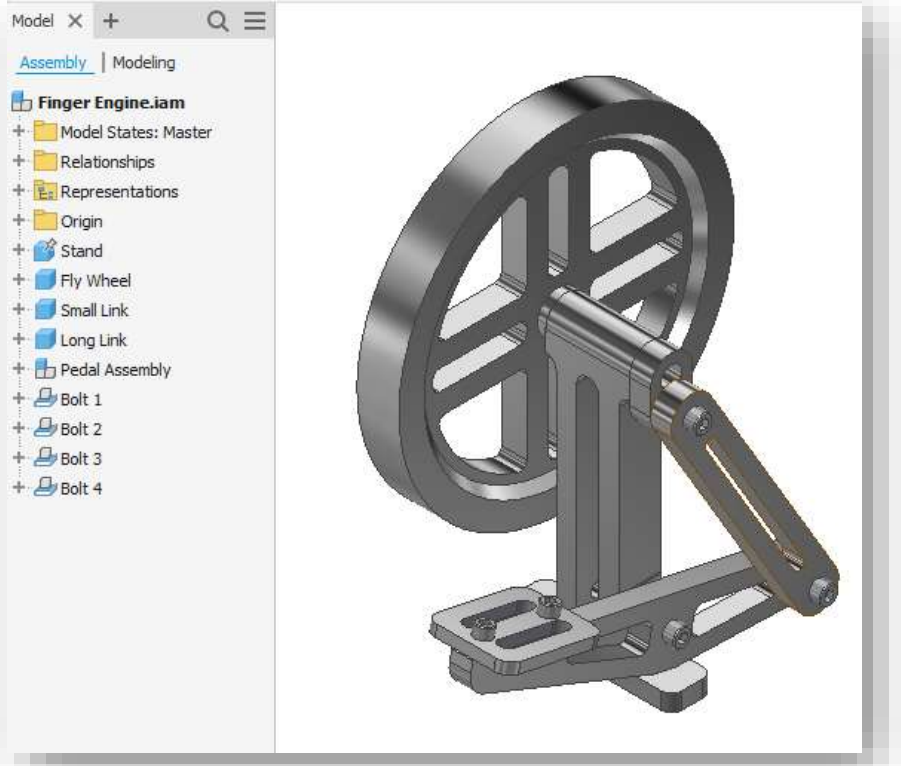




Place and Constrain

NEW ILOGIC APPROACH

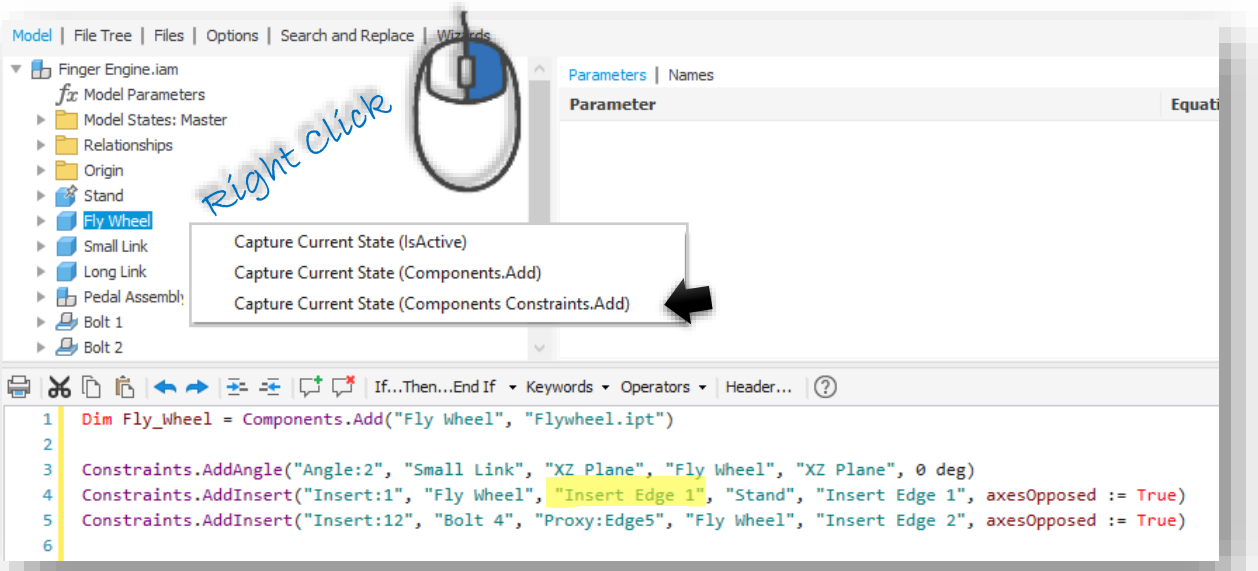
1. Create Named Geometry Entities in the components to use for constraints
2. Create and constrain your assembly



Extract the Code

NEW ILOGIC APPROACH

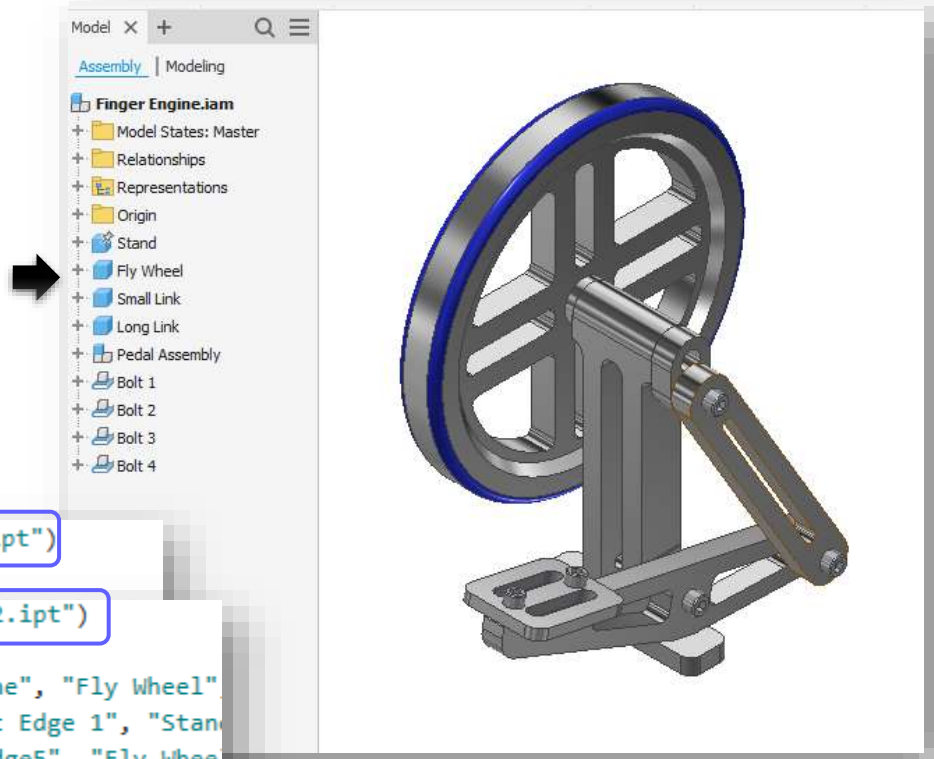
1. Create Named Geometry Entities in the components to use for constraints
2. Create and constrain your assembly
3. Extract the code from your fully assembled model



Configure the rule

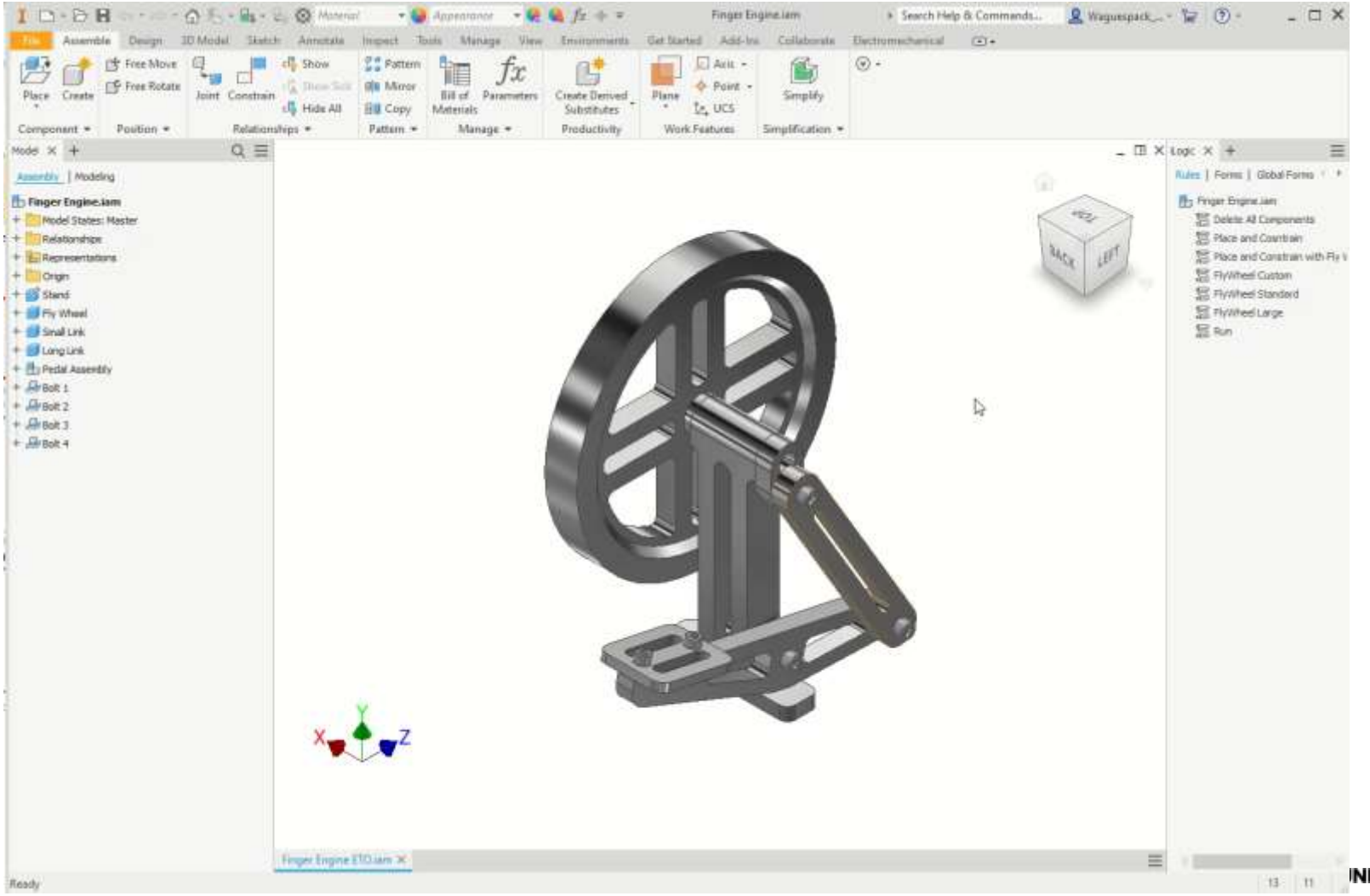
NEW ILOGIC APPROACH

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.

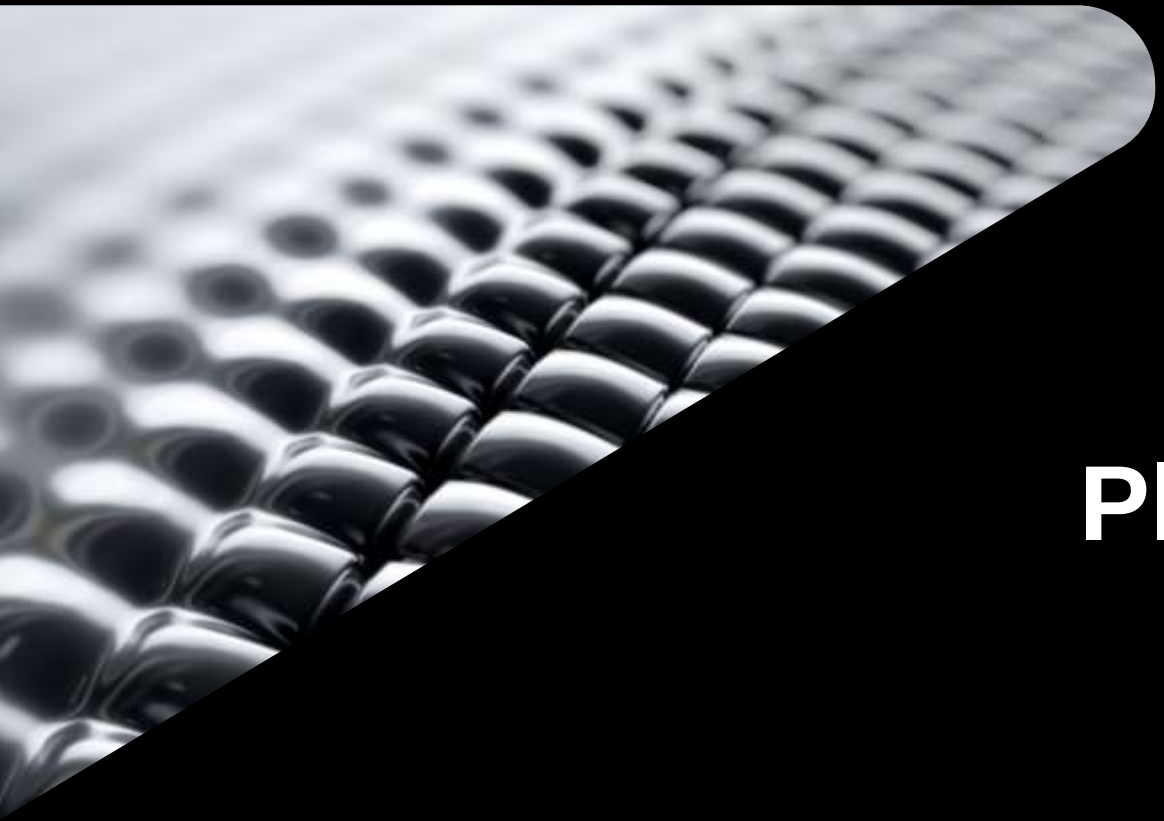



```
1  Dim Fly_Wheel = Components.Add("Fly Wheel", "Flywheel.ipt")
2
```

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


Finger Engine.iam
Search Help & Commands...
Waguespack_...
File
Assemble
Design
3D Model
Sketch
Annotate
Inspect
Tools
Manage
View
Environments
Get Started
Add-Ins
Collaborate
Electromechanical
Place
Create
Free Move
Free Rotate
Joint
Constrain
Show
Hide All
Pattern
Mirror
Copy
Bill of Materials
Parameters
Create Derived Substitutes
Plane
Axis
Point
UCS
Simplify
Component
Position
Relationships
Pattern
Manage
Productivity
Work Features
Simplification
Model
Assembly
Modeling
Finger Engine.iam
Model States: Master
Relationships
Representations
Origin
Stand
Fly Wheel
Small Link
Long Link
Pedal Assembly
Bolt 1
Bolt 2
Bolt 3
Bolt 4
Logic
Rules
Forms
Global Forms
Finger Engine.iam
Delete All Components
Place and Constrain
Place and Constrain with Fly
FlyWheel Custom
FlyWheel Standard
FlyWheel Large
Run
BACK
LEFT
X
Z
Finger Engine ETO.iam
Ready

NIVERSITY



Plan your code

Tip #9

Planning our code allows us to write much more maintainable iLogic rules

2510 lines of code!

Rule to do the same thing, but only 67 lines of code!

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

```
oTrigger = Height
If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then
    Parameter("Size") = 4
    Parameter("Count") = 3

    Component.Visible("NG4 AssemblyLH:1") = True
    oCompOcc = Component.InventorComponent("NG4 AssemblyLH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
    Component.Visible("NG4 AssemblyRH:1") = True
    oCompOcc = Component.InventorComponent("NG4 AssemblyRH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
    Constraint.IsActive("NG4 _Mate") = True

    Component.Visible("NG5 AssemblyLH:1") = False
    oCompOcc = Component.InventorComponent("NG5 AssemblyLH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
    Component.Visible("NG5 AssemblyRH:1") = False
    oCompOcc = Component.InventorComponent("NG5 AssemblyRH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
    Constraint.IsActive("NG5 _Mate") = False

    Component.Visible("NG6 AssemblyLH:1") = False
    oCompOcc = Component.InventorComponent("NG6 AssemblyLH:1")
```

Evaluate and outline what you have

```
oTrigger = Height
If Parameter("Height") >= 24 And Parameter("Height") <= 32 Then
    Parameter("Size") = 4
    Parameter("Count") = 3

    Component.Visible("NG4 AssemblyLH:1") = True
    oCompOcc = Component.InventorComponent("NG4 AssemblyLH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
    Component.Visible("NG4 AssemblyRH:1") = True
    oCompOcc = Component.InventorComponent("NG4 AssemblyRH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kDefaultBOMStructure
    Constraint.IsActive("NG4 _Mate") = True

    Component.Visible("NG5 AssemblyLH:1") = False
    oCompOcc = Component.InventorComponent("NG5 AssemblyLH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
    Component.Visible("NG5 AssemblyRH:1") = False
    oCompOcc = Component.InventorComponent("NG5 AssemblyRH:1")
    oCompOcc.BOMStructure = BOMStructureEnum.kReferenceBOMStructure
    Constraint.IsActive("NG5 _Mate") = False

    Component.Visible("NG6 AssemblyLH:1") = False
    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

140 LINES

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,

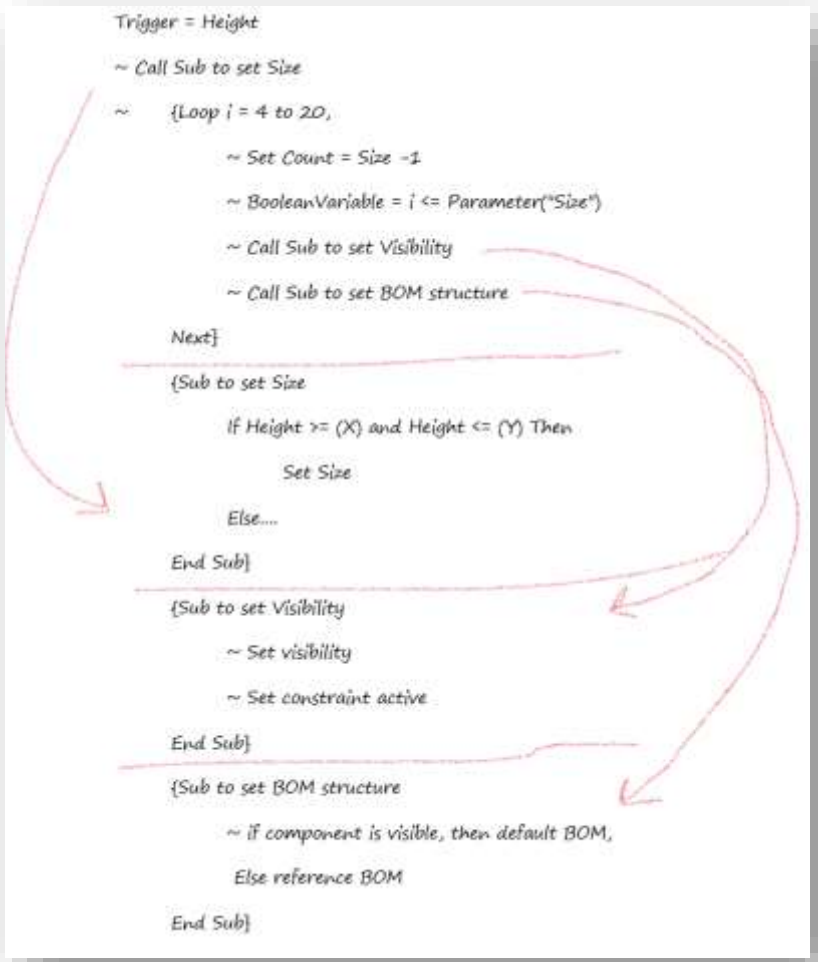
all others are invisible and reference

Else if....

140 x 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



```
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " Assembly1:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " Assembly0:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub

Sub Set_Visible(oOccName As String, oBoolean As Boolean)

    Component.InventorComponent(oOccName).Visible = oBoolean
    Constraint.IsActive("NG" & i & " _Mate") = oBoolean

    'step into subroutine to set BOM Structure
    Call BOMStructure(oOccName)

End Sub

Sub BOMStructure(oOccName As String)

    If Component.Visible(oOccName) = True Then
       Component.InventorComponent(oOccName).BOMStructure = _
       BOMStructureEnum.kDefaultBOMStructure
    ElseIf Component.Visible(oOccName) = False Then
       Component.InventorComponent(oOccName).BOMStructure = _
       BOMStructureEnum.kReferenceBOMStructure
    End If

End Sub

Sub SetSize(oHeight As Integer)

    If Parameter("Height") >= 24 And Parameter("Height") <= 31 Then : Parameter("Size") = 4
    ElseIf Parameter("Height") > 31 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()
    Inventor0.DocumentUpdate()
End Sub
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
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 Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

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
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
Sub Set_Visible(oOccName As String, oBoolean As Boolean)

    Component.InventorComponent(oOccName).Visible = oBoolean
    Constraint.IsActive("NG" & i & " _Mate") = oBoolean

    'step into subroutine to set BOM Structure
    Call BOMStructure(oOccName)

End Sub
```

```vb
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
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
Sub Set_Visible(oOccName As String, oBoolean As Boolean)

    Component.InventorComponent(oOccName).Visible = oBoolean
    Constraint.IsActive("NG" & i & " _Mate") = oBoolean

    'step into subroutine to set BOM Structure
    Call BOMStructure(oOccName)

End Sub
```

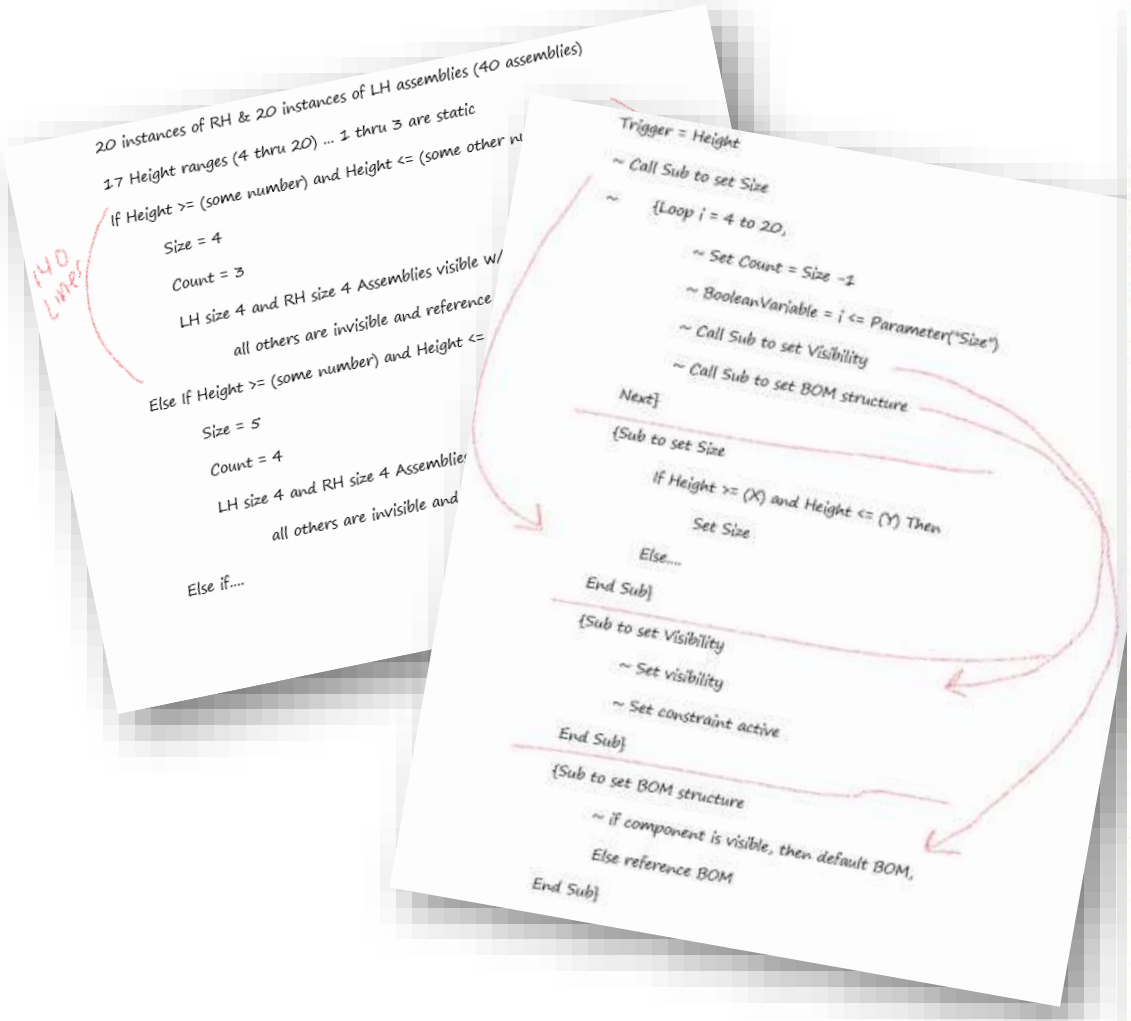
```vb
Sub SetSize(oHeight As Integer)

    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
```

```vb
Sub Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NG" & i & " AssemblyLH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NG" & i & " AssemblyRH:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub
```

```vb
Sub Set_Visible(oOccName As String, oBoolean As Boolean)

    Component.InventorComponent(oOccName).Visible = oBoolean
    Constraint.IsActive("NG" & i & " _Mate") = oBoolean

    'step into subroutine to set BOM Structure
    Call BOMStructure(oOccName)

End Sub
```

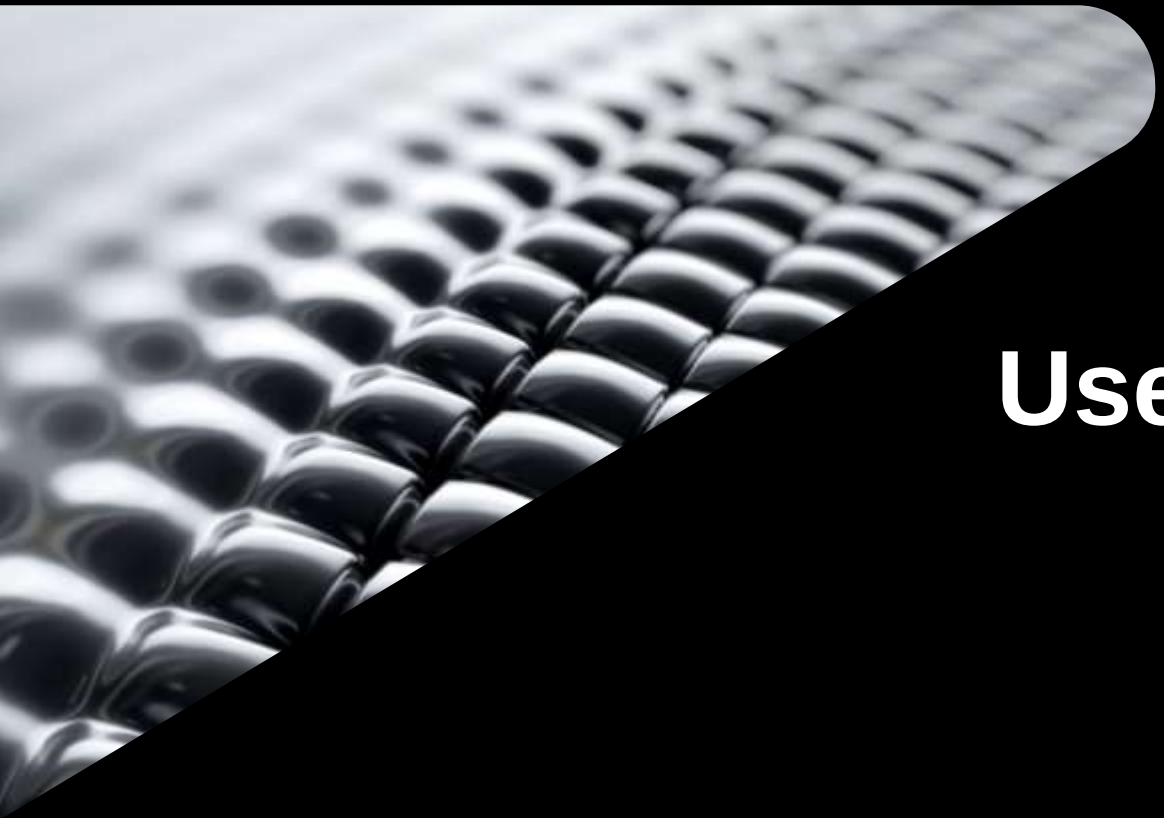
```vb
Sub BOMStructure(oOccName As String)

    If Component.Visible(oOccName) = True Then
        Component.InventorComponent(oOccName).BOMStructure = _
        BOMStructureEnum.kDefaultBOMStructure
    ElseIf Component.Visible(oOccName) = False Then
        Component.InventorComponent(oOccName).BOMStructure = _
        BOMStructureEnum.kReferenceBOMStructure
    End If

End Sub
```

```vb
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 Main

    oTrigger = Height

    'step into subroutine to set Size
    Call SetSize(oHeight)

    For i = 4 To 20
        Parameter("Count") = Parameter("Size") -1

        oBoolean = i <= Parameter("Size")

        oOccName = "NS" & i & " Assembly1:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility

        oOccName = "NS" & i & " Assembly0:1"
        Call Set_Visible(oOccName, oBoolean) 'step into subroutine to set visibility
    Next

End Sub

Sub Set_Visible(oOccName As String, oBoolean As Boolean)

    Component.InventorComponent(oOccName).Visible = oBoolean
    Constraint.IsActive("NS" & i & " _Mate") = oBoolean

    'step into subroutine to set BOM Structure
    Call BOMStructure(oOccName)

End Sub

Sub BOMStructure(oOccName As String)

    If Component.Visible(oOccName) = True Then
       Component.InventorComponent(oOccName).BOMStructure = _
       BOMStructureEnum.kDefaultBOMStructure
    ElseIf Component.Visible(oOccName) = False Then
       Component.InventorComponent(oOccName).BOMStructure = _
       BOMStructureEnum.kReferenceBOMStructure
    End If

End Sub

Sub SetSize(oHeight As Integer)

    If Parameter("Height") >= 24 And Parameter("Height") <= 31 Then : Parameter("Size") = 4
    ElseIf Parameter("Height") > 31 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
```

IIVERSITY



Use Loops when possible

Tip #10

Use Loops to eliminate repetitious lines

Unoptimized rule

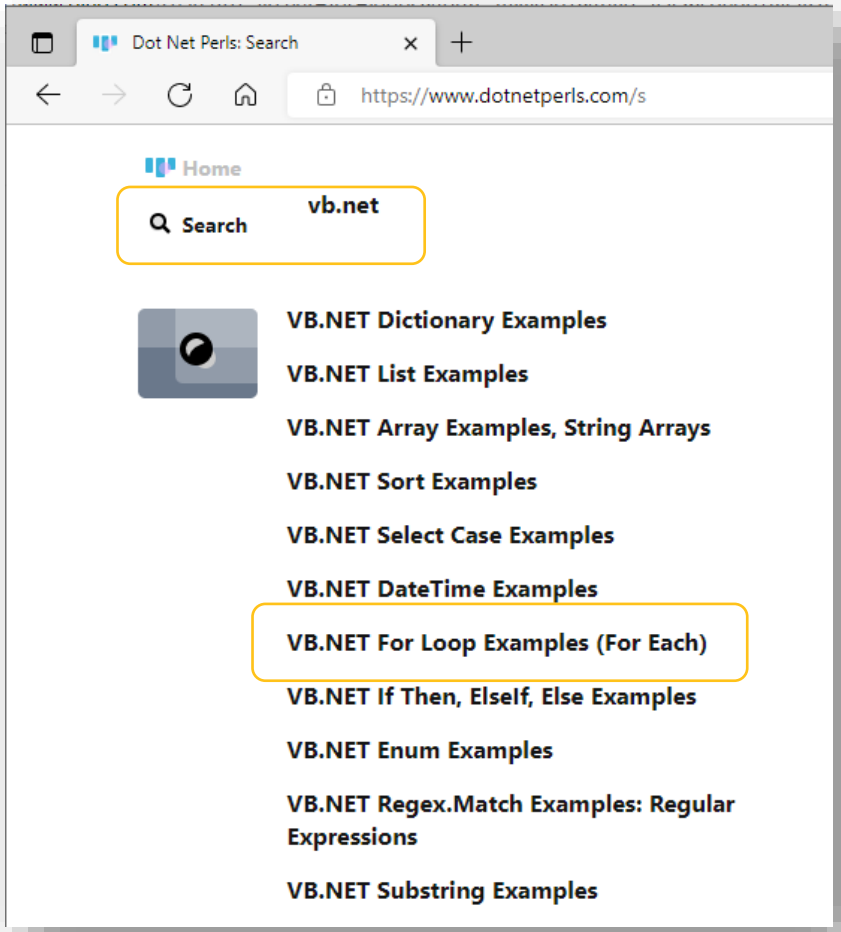
```
If Parameter("IsRightHand")True Then
    Component.IsActive("RH_Brace") = True
    Component.IsActive("RH_TopBracket") = True
    Component.IsActive("RH_BottomBracket") = True
    Component.IsActive("RH_Lid") = True
    Component.IsActive("RH_Shim") = True

    Component.IsActive("LH_Brace") = False
    Component.IsActive("LH_TopBracket") = False
    Component.IsActive("LH_BottomBracket") = False
    Component.IsActive("LH_Lid") = False
    Component.IsActive("LH_Shim") = False

Else
    Component.IsActive("RH_Brace") = False
    Component.IsActive("RH_TopBracket") = False
    Component.IsActive("RH_BottomBracket") = False
    Component.IsActive("RH_Lid") = False
    Component.IsActive("RH_Shim") = False

    Component.IsActive("LH_Brace") = True
    Component.IsActive("LH_TopBracket") = True
    Component.IsActive("LH_BottomBracket") = True
    Component.IsActive("LH_Lid") = True
    Component.IsActive("LH_Shim") = True
End If
```

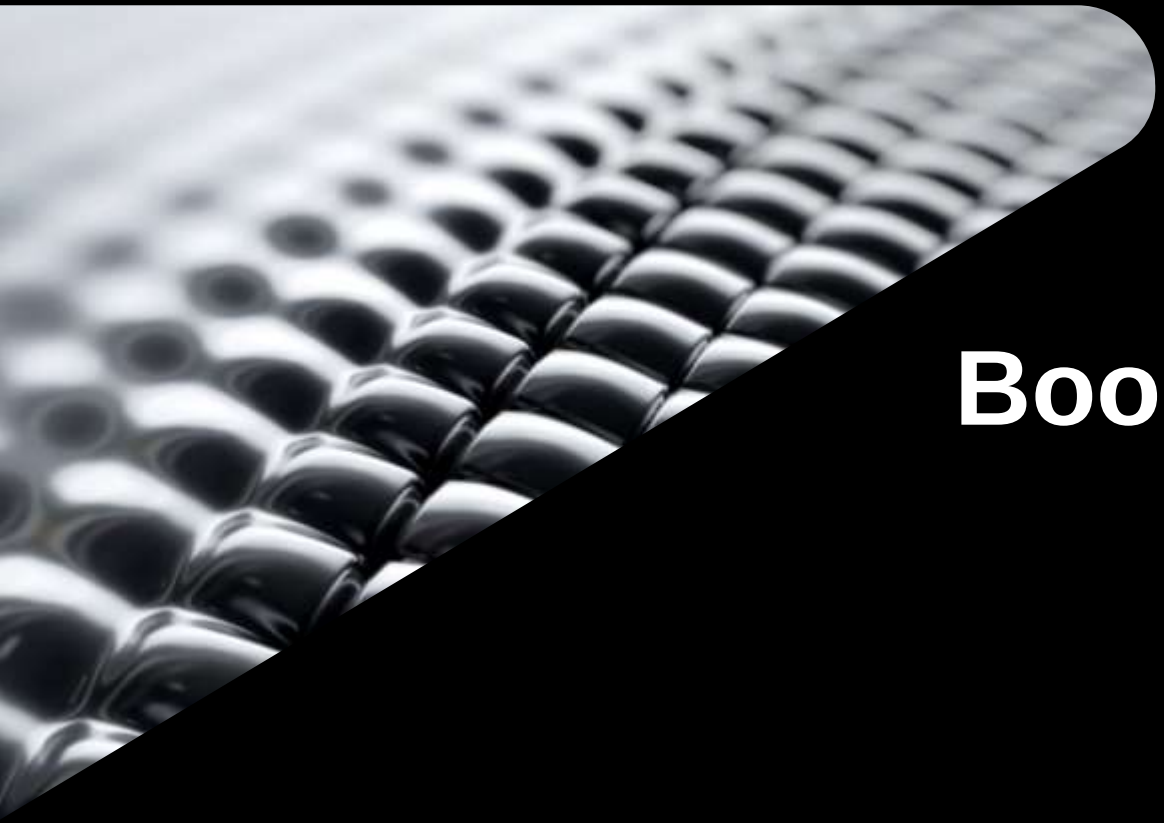
Optimized rule using a For Each loop

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

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

    If Left(oOcc.Name, 3) = "RH_" Then
        Component.IsActive(oOcc.Name) = Parameter("IsRightHand")
    ElseIf Left(oOcc.Name, 3) = "LH_" Then
        Component.IsActive(oOcc.Name) = Not Parameter("IsRightHand")
    End If
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 is a site that has a lot of examples

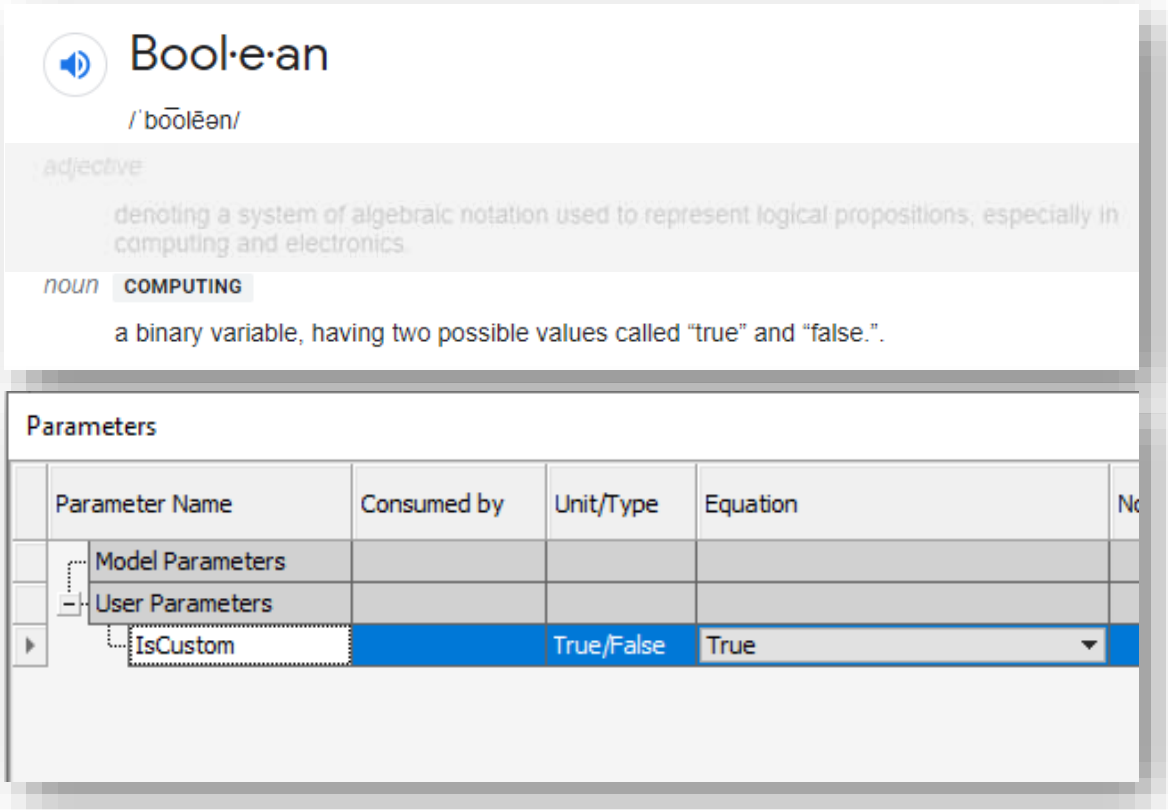




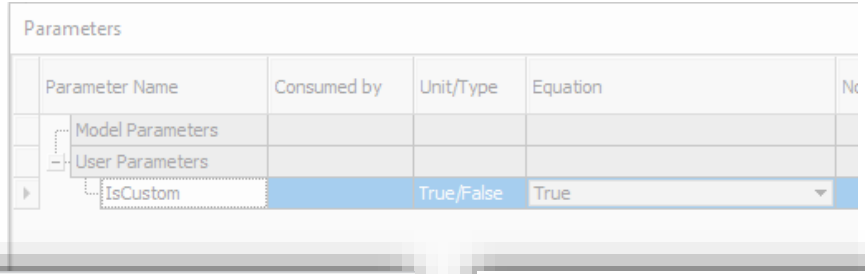
Booleans and the Not operator

Tip #11

Boolean = True/False



Skip the If/Then statements and use the Not operator




```
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")
```

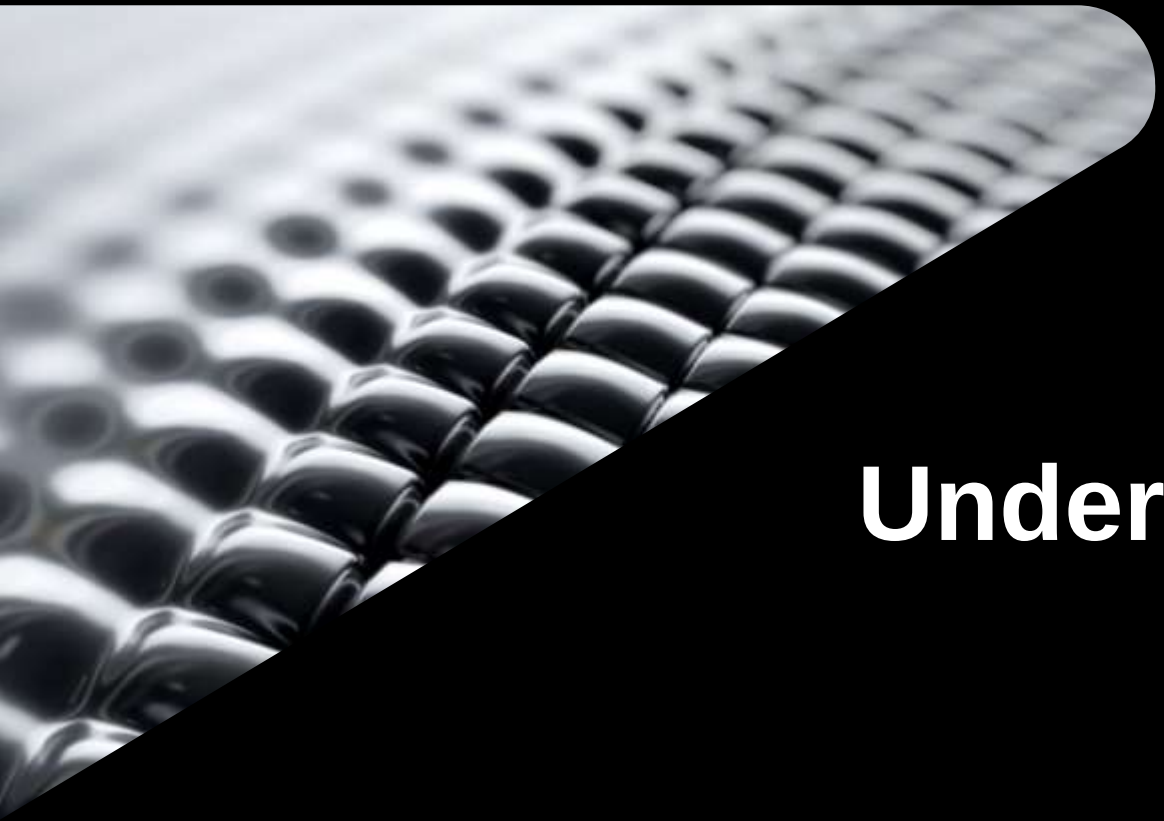
Optimized rule using the Not Operator

AUTODESK UNIVERSITY

Objective:

Finding your way around the Inventor API

API: Application Programming Interface



Understand the API

Tip #12

An Analogy

Inventor's Source Code

Inventor's API
(**A**pplication **P**rogramming **I**nterface)

iLogic

- 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

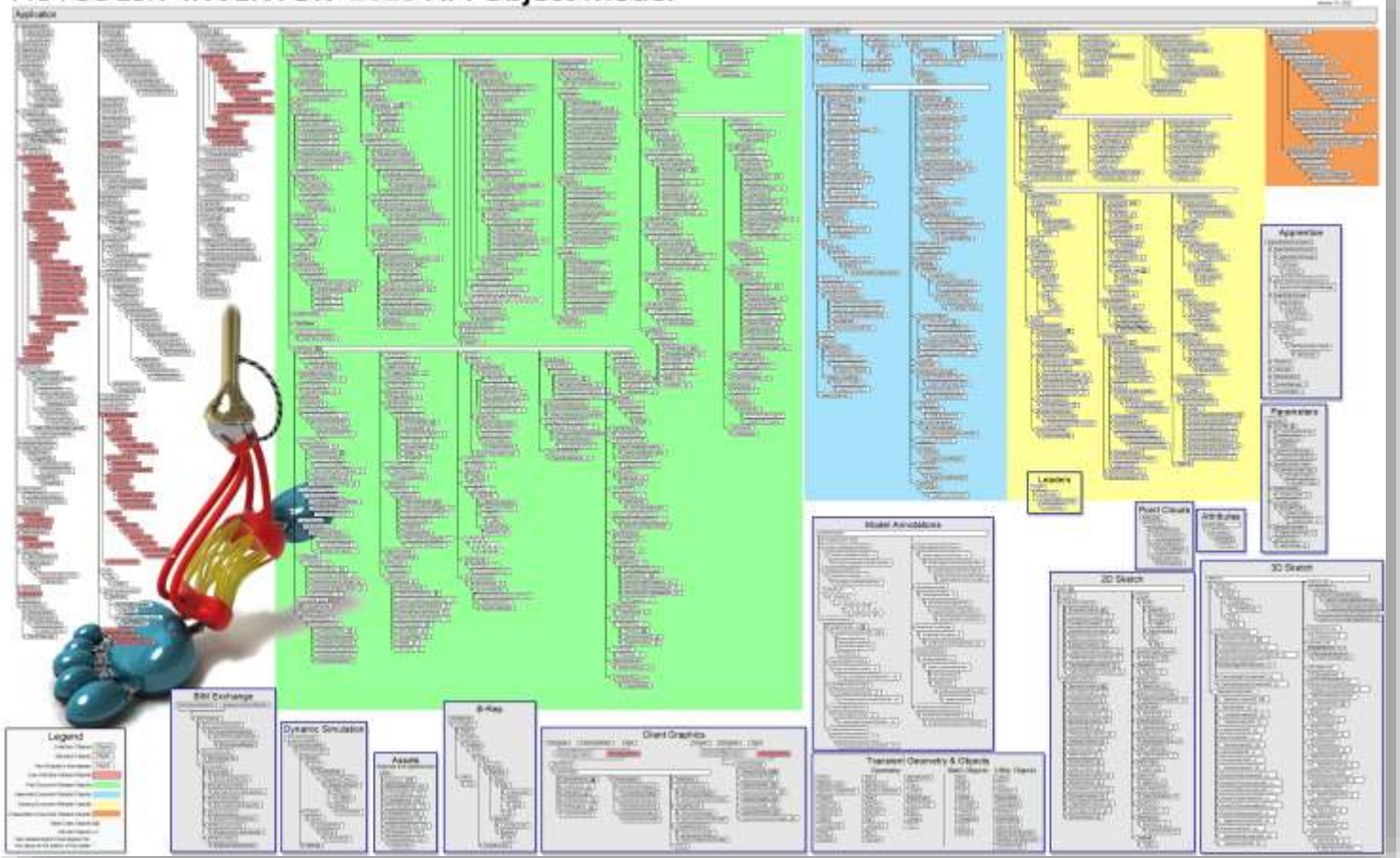
```
Dim oDoc As PartDocument
oDoc = ThisApplication.ActiveDocument

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"
```

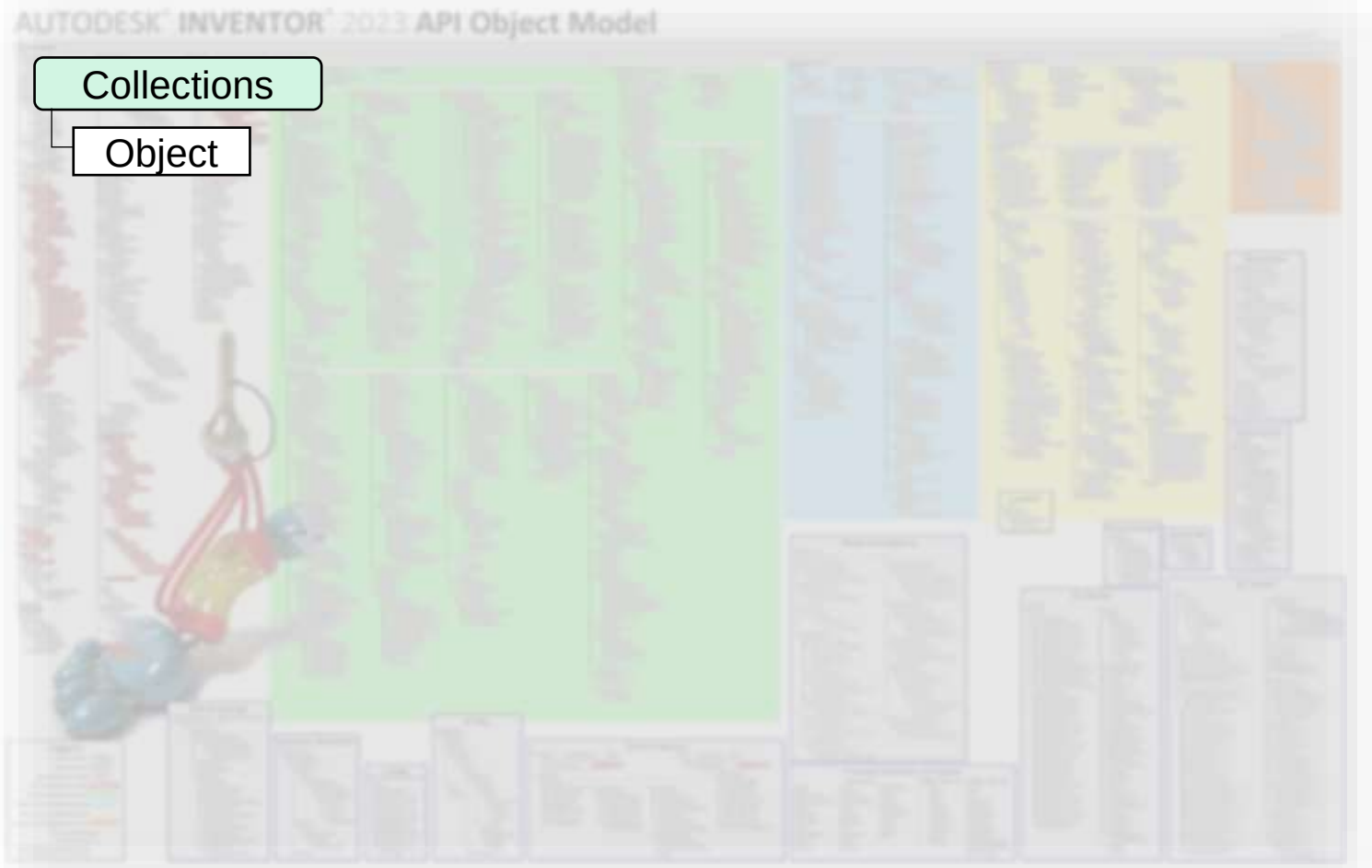
Both do the exact same thing

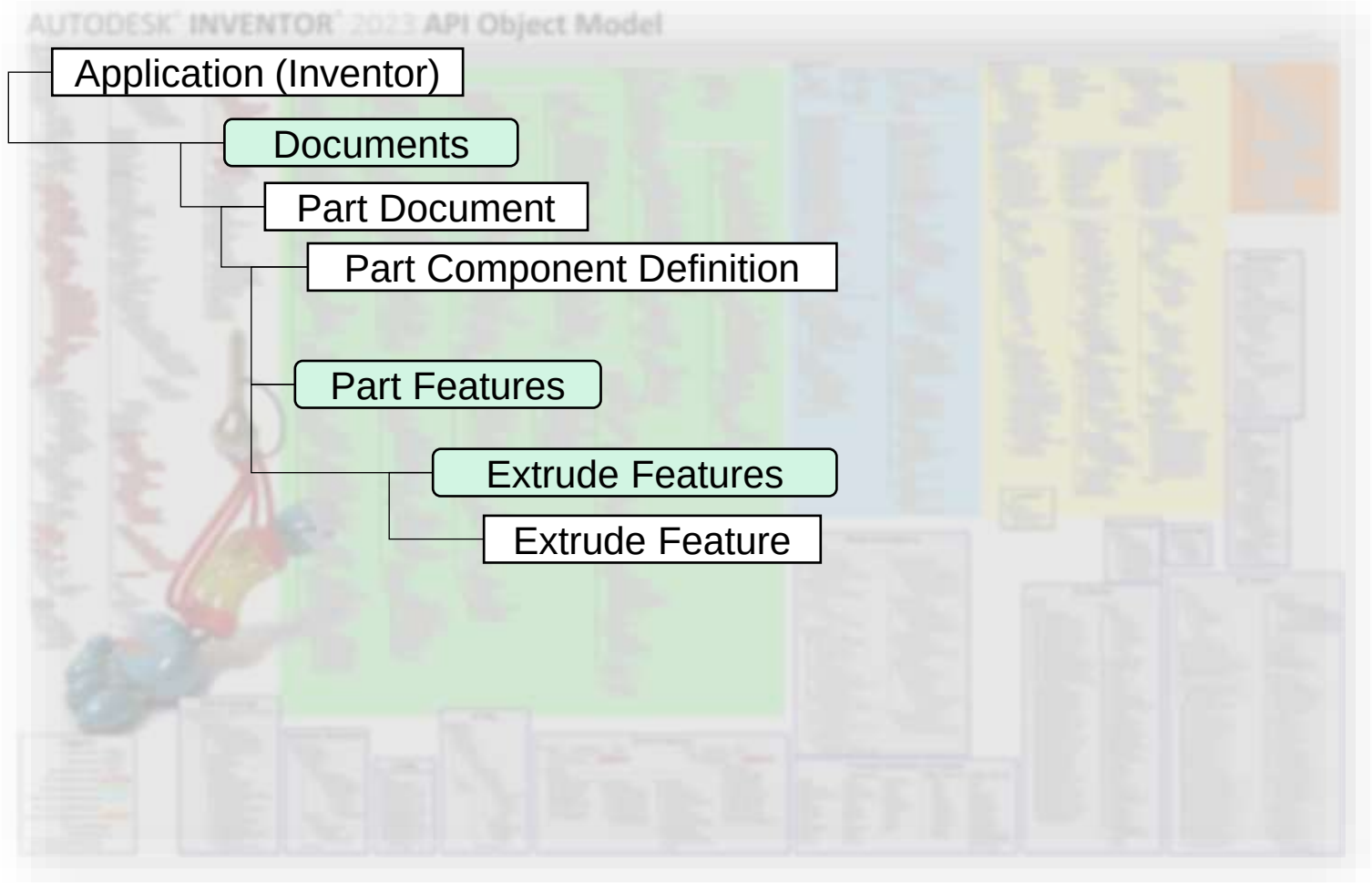
iLogic version

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

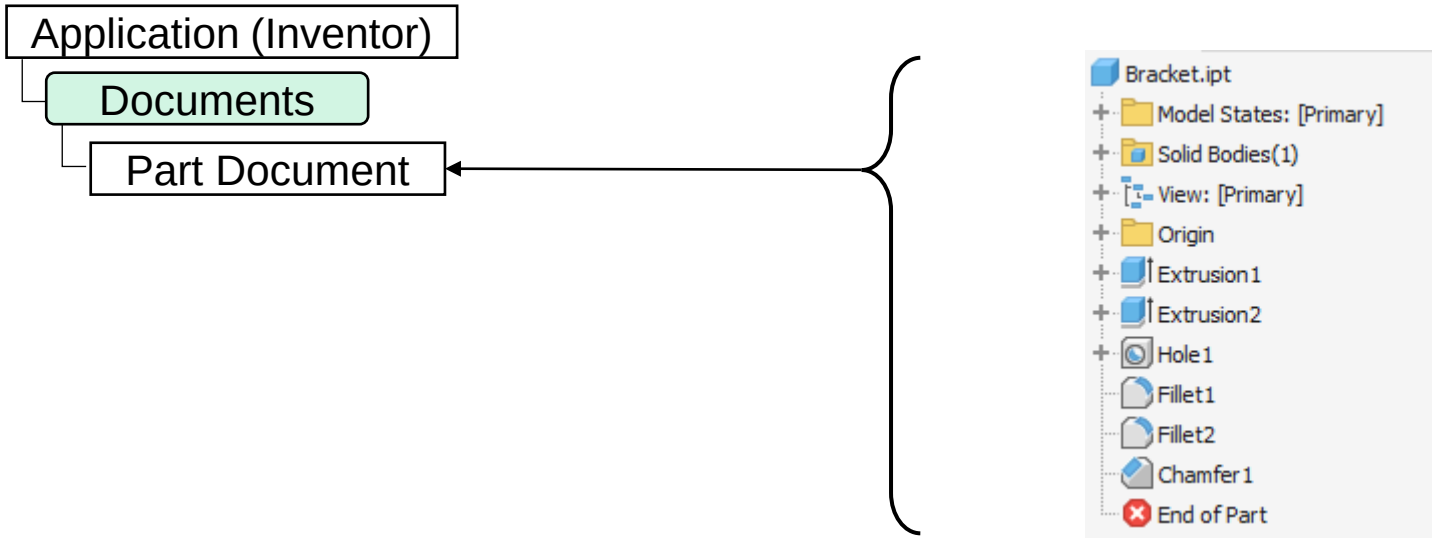
AUTODESK® INVENTOR® 2023 API Object Model



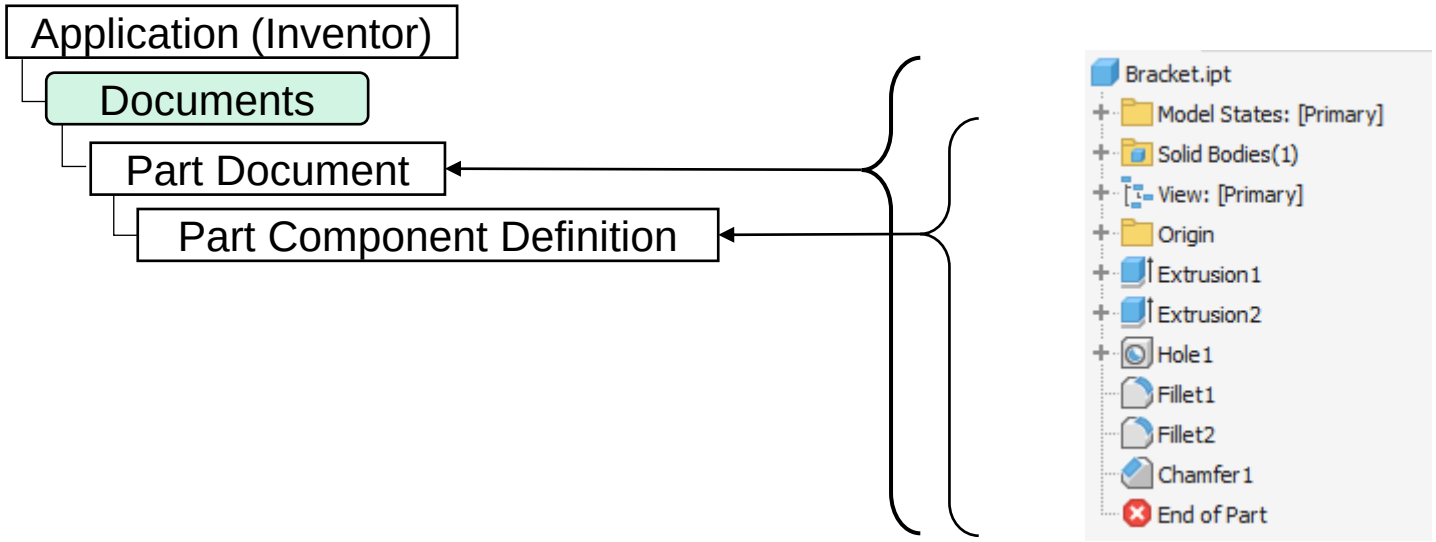

AUTODESK® INVENTOR® 2023 API Object Model
Collections
Object


AUTODESK® INVENTOR® 2023 API Object Model
Application (Inventor)
Documents
Part Document
Part Component Definition
Part Features
Extrude Features
Extrude Feature

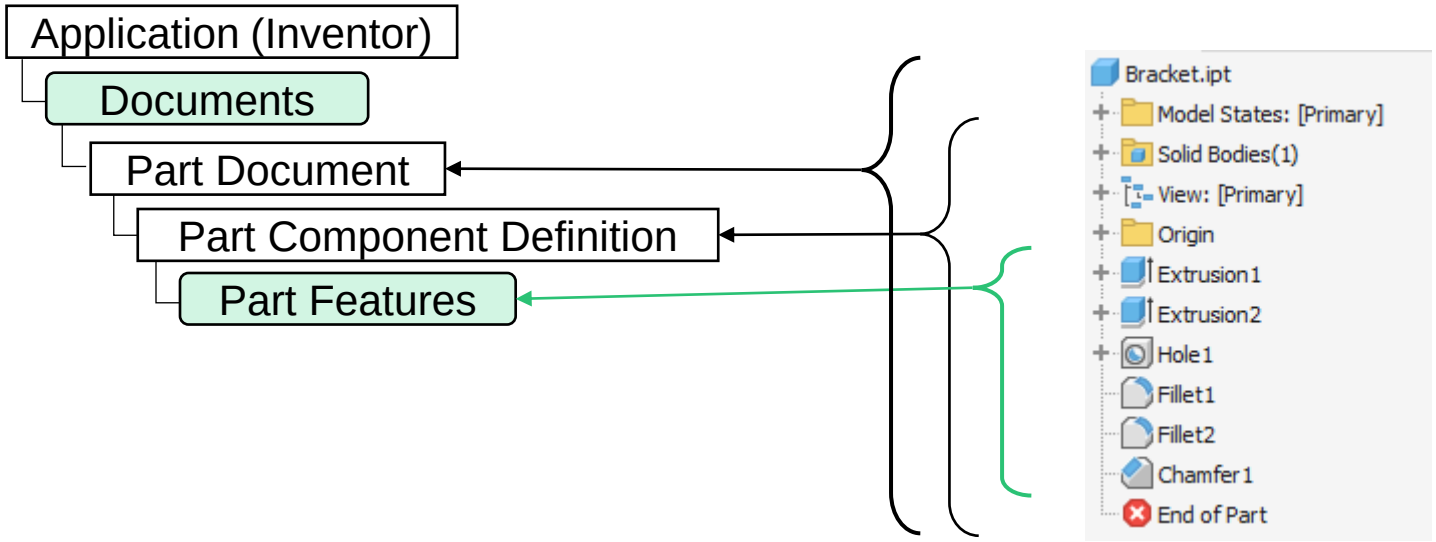
API Object Model of a part



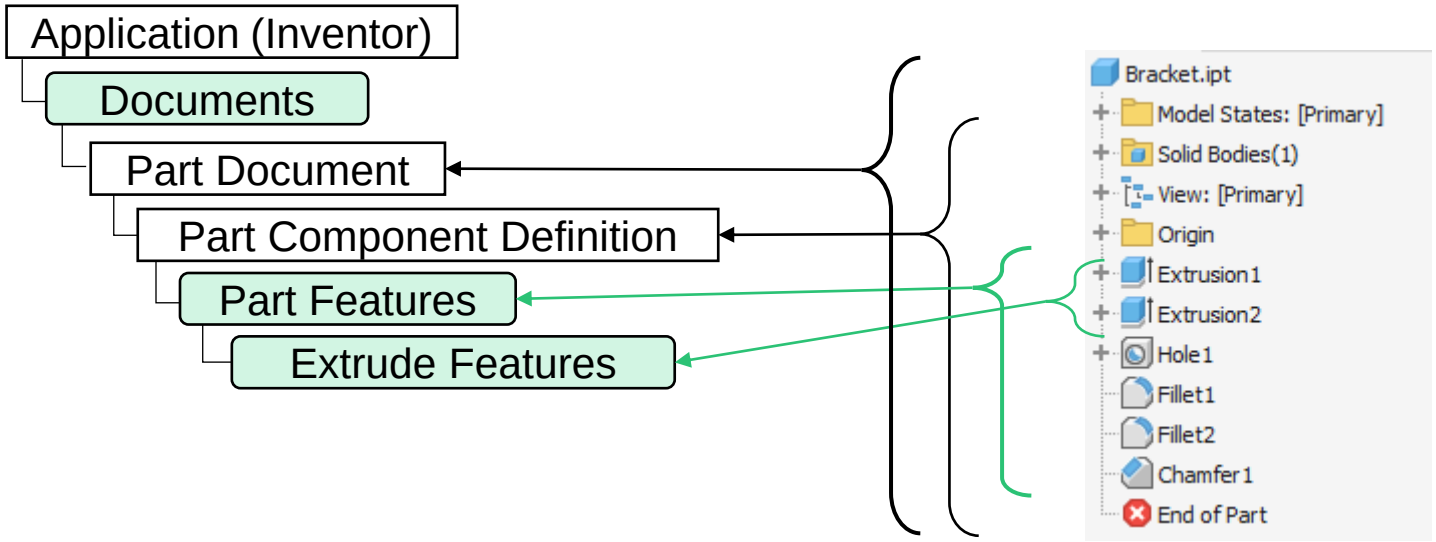
API Object Model of a part



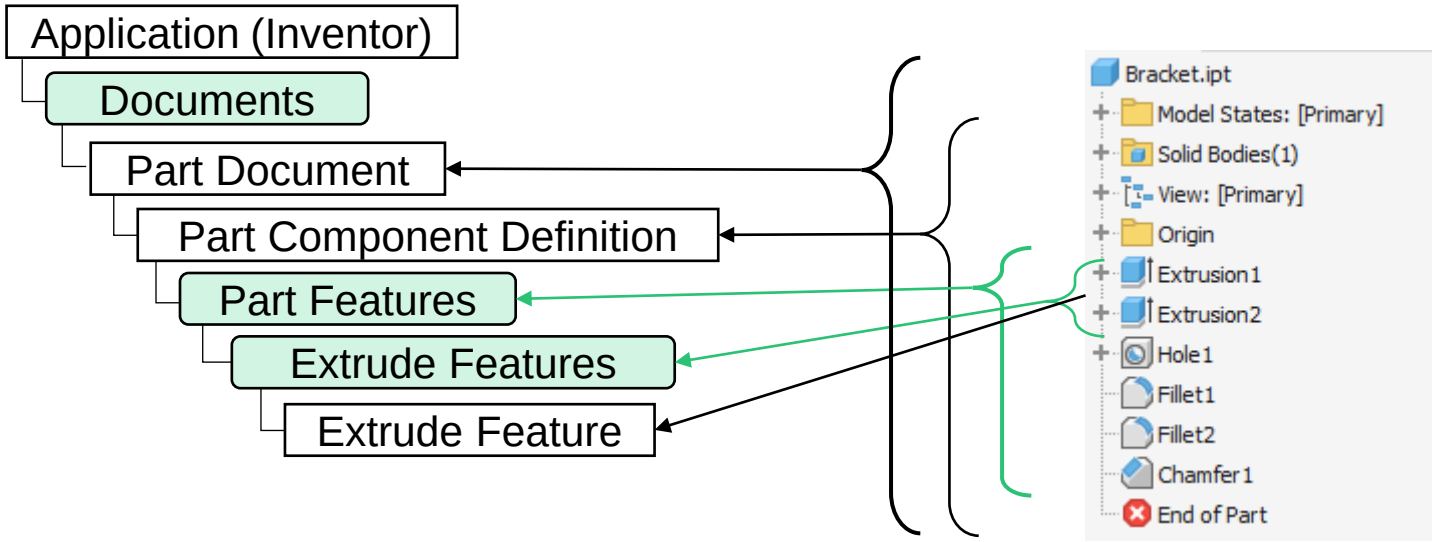
API Object Model of a part



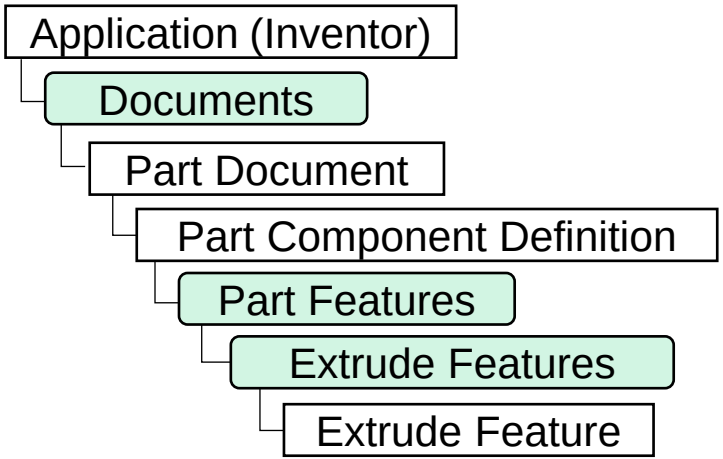
API Object Model of a part



API Object Model of a part

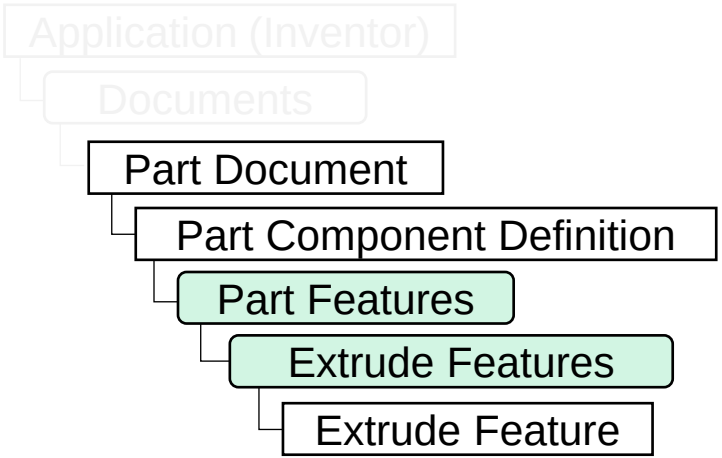


Code Comparison




```
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")
```

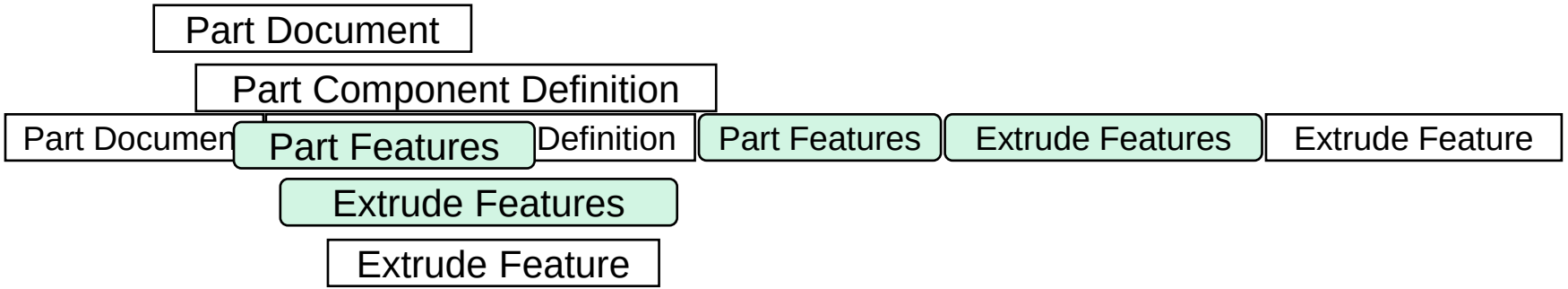
Using iLogic's ThisDoc.Document to get the document the rule is triggered from

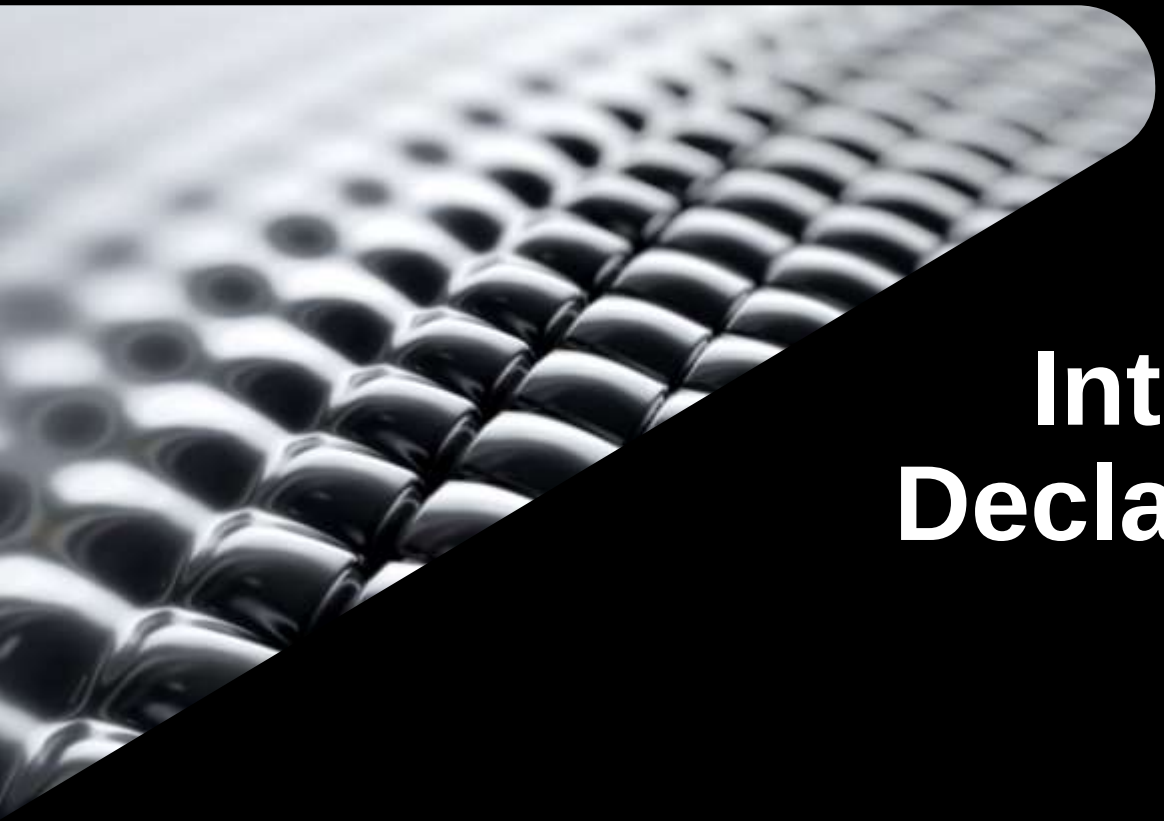



```
1   Dim oDoc As PartDocument
2   oDoc = ThisDoc.Document
3
4   Dim oDefinition As ComponentDefinition
5   oDefinition = oDoc.ComponentDefinition
6
7   Dim oFeatures As PartFeatures
8   oFeatures = oDefinition.Features
9
10  Dim oExtrusions As ExtrudeFeatures
11  oExtrusions = oFeatures.ExtrudeFeatures
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")
```

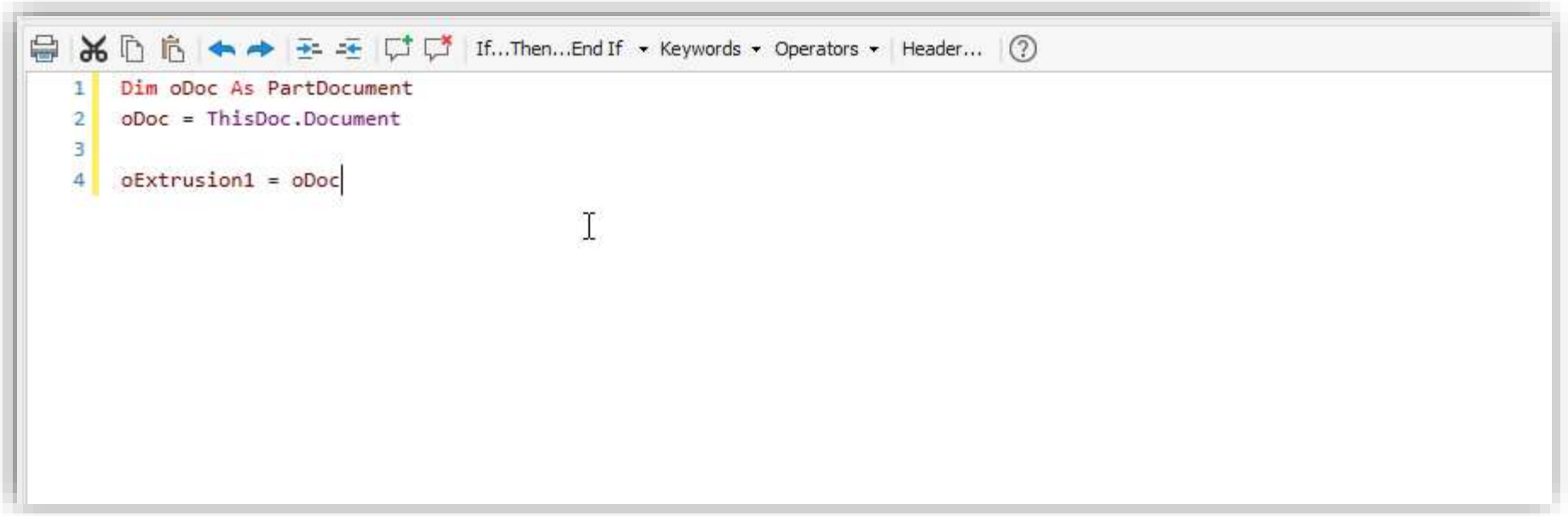




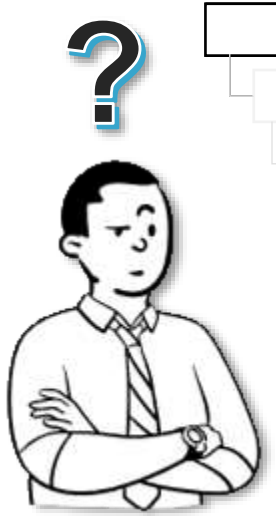
IntelliSence and Declared Variables

Tip #13

Using IntelliSense to navigate the API



Why can’t I find the API object I was expecting?



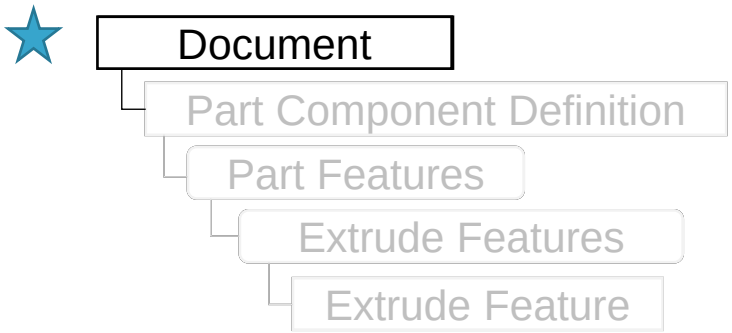
Document

- Part Component Definition
  - Part Features
    - Extrude Features
      - Extrude Feature

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

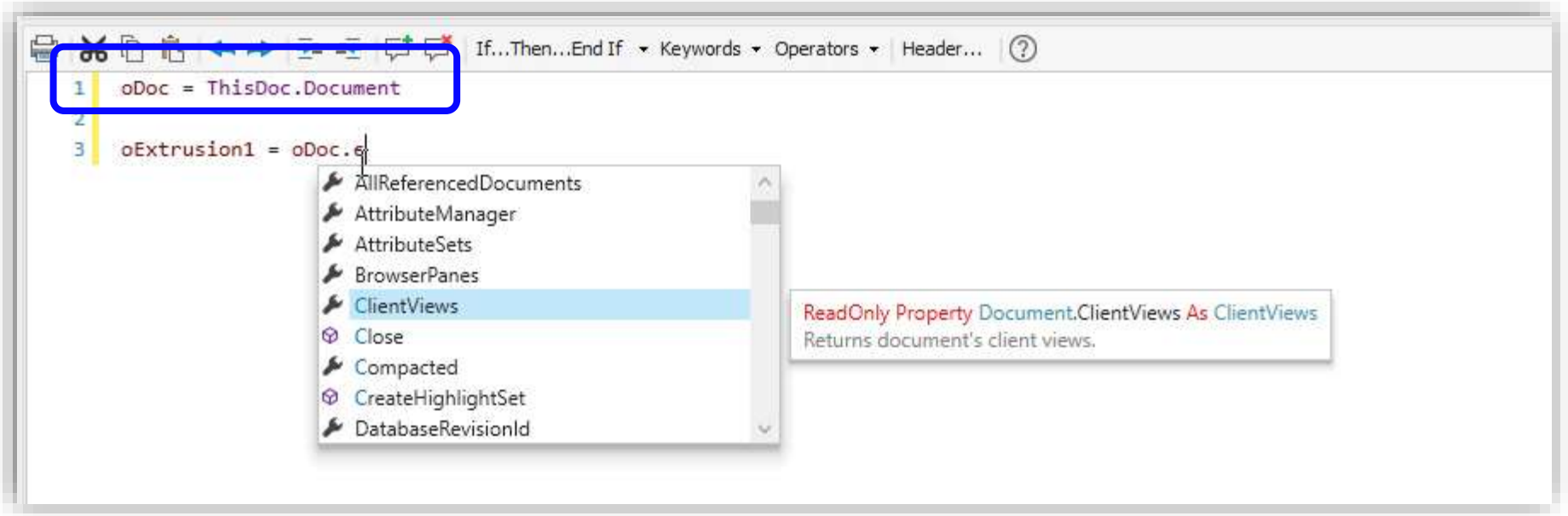
ComponentDefinition ?

Solution: Declare the variable

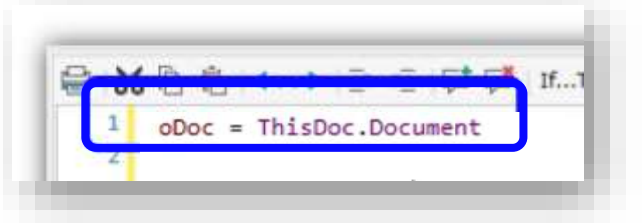



```
oDoc = ThisDoc.Document
```

Declaring the Variable

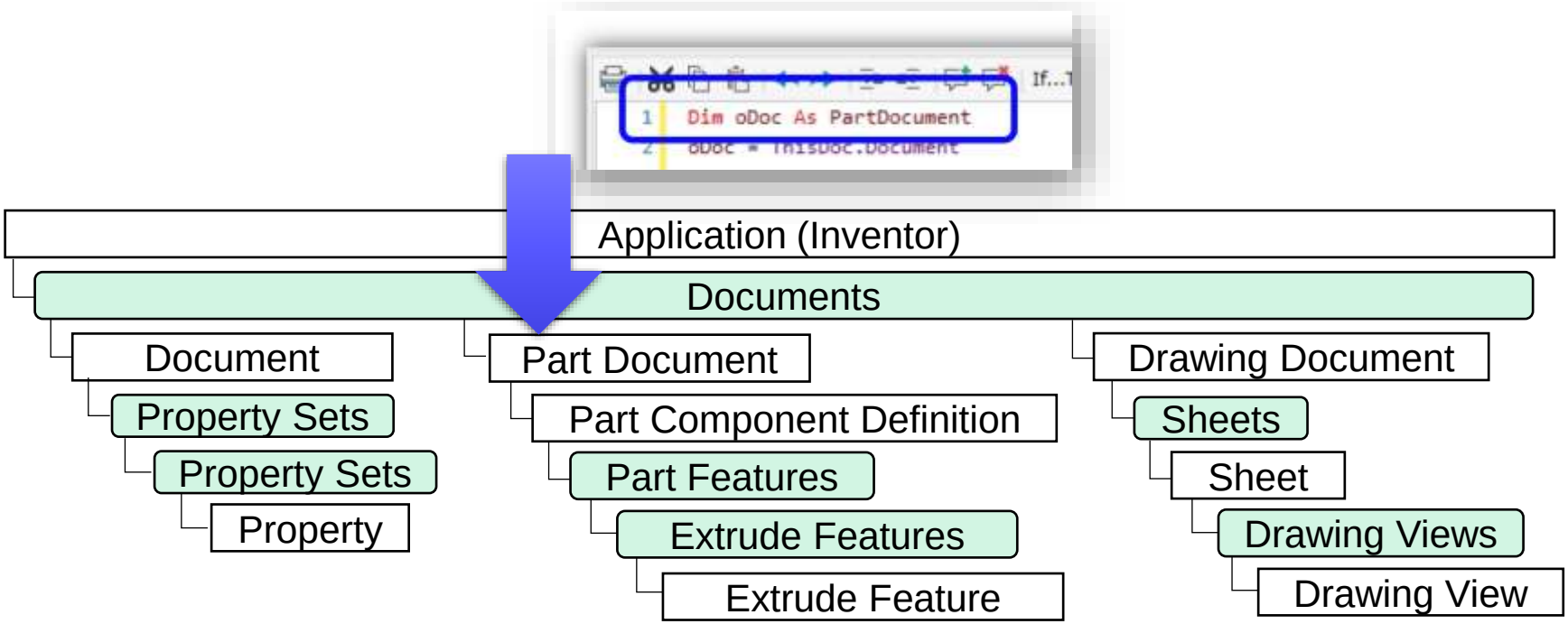


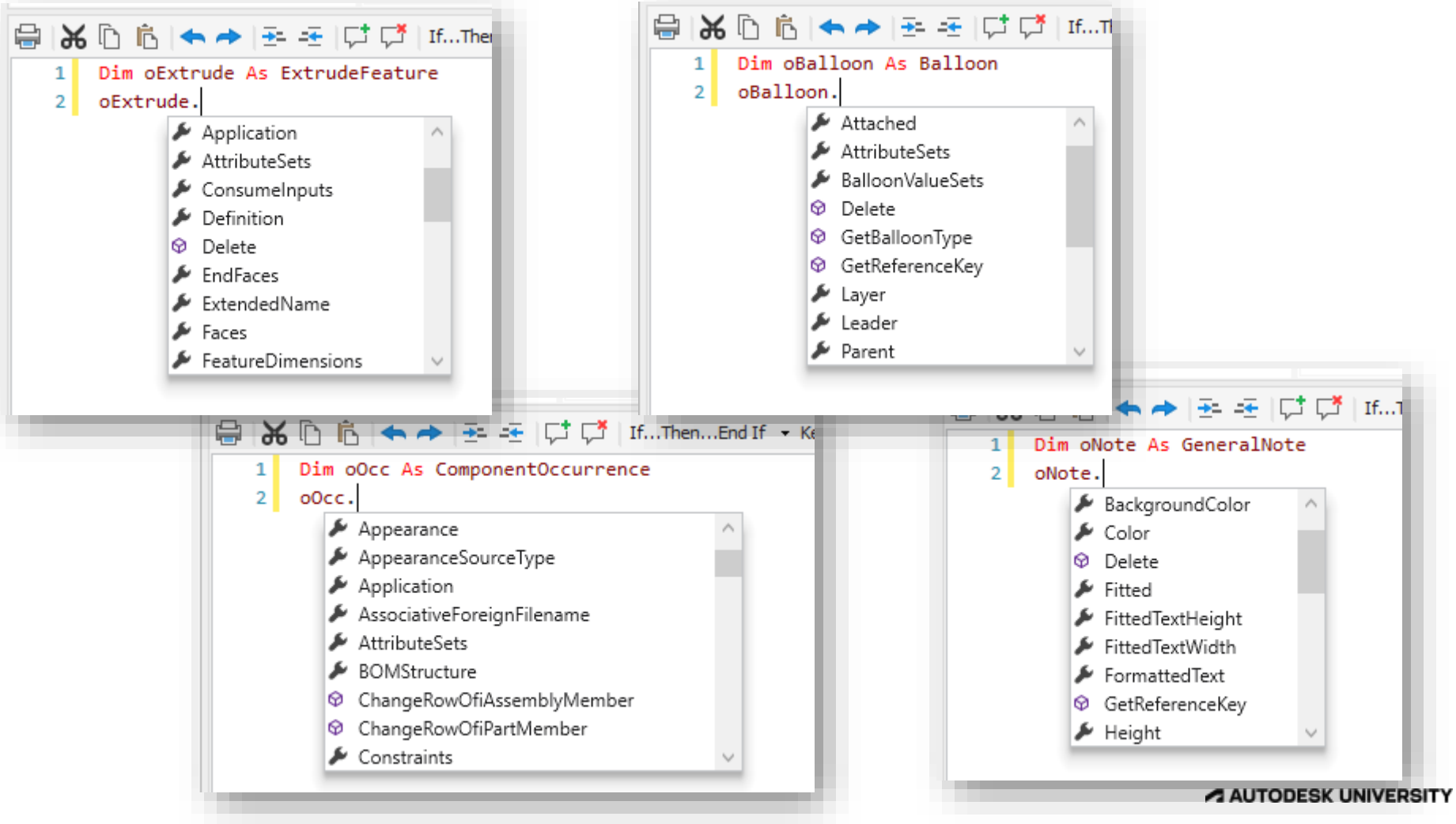
What happens when we declare the variable?



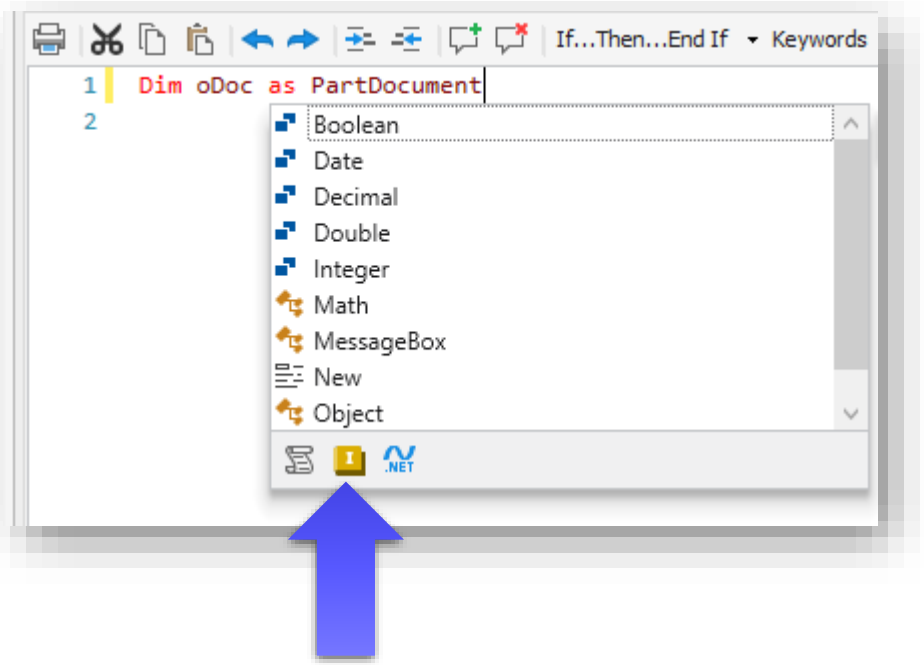
- Application (Inventor)
  - Documents
    - Document
      - Property Sets
        - Property Sets
          - Property
    - Part Document
      - Part Component Definition
        - Part Features
          - Extrude Features
            - Extrude Feature
    - Drawing Document
      - Sheets
        - Sheet
          - Drawing Views
            - Drawing View

What happens when we declare the variable?




Dim oExtrude As ExtrudeFeature
oExtrude.
Application
AttributeSets
ConsumeInputs
Definition
Delete
EndFaces
ExtendedName
Faces
FeatureDimensions
Dim oBalloon As Balloon
oBalloon.
Attached
AttributeSets
BalloonValueSets
Delete
GetBalloonType
GetReferenceKey
Layer
Leader
Parent
Dim oOcc As ComponentOccurrence
oOcc.
Appearance
AppearanceSourceType
Application
AssociativeForeignFilename
AttributeSets
BOMStructure
ChangeRowOfiAssemblyMember
ChangeRowOfiPartMember
Constraints
Dim oNote As GeneralNote
oNote.
BackgroundColor
Color
Delete
Fitted
FittedTextHeight
FittedTextWidth
FormattedText
GetReferenceKey
Height

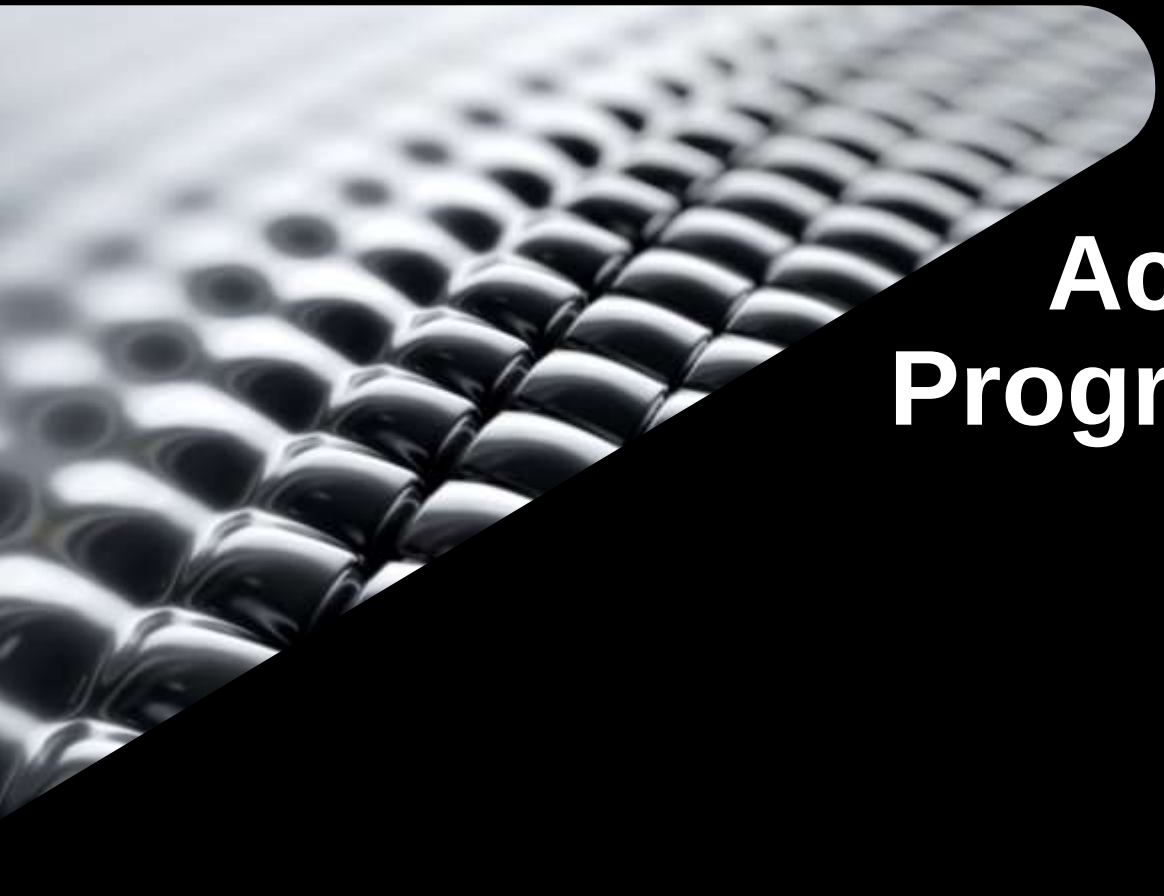
iLogic Editor IntelliSense Filters



iLogic

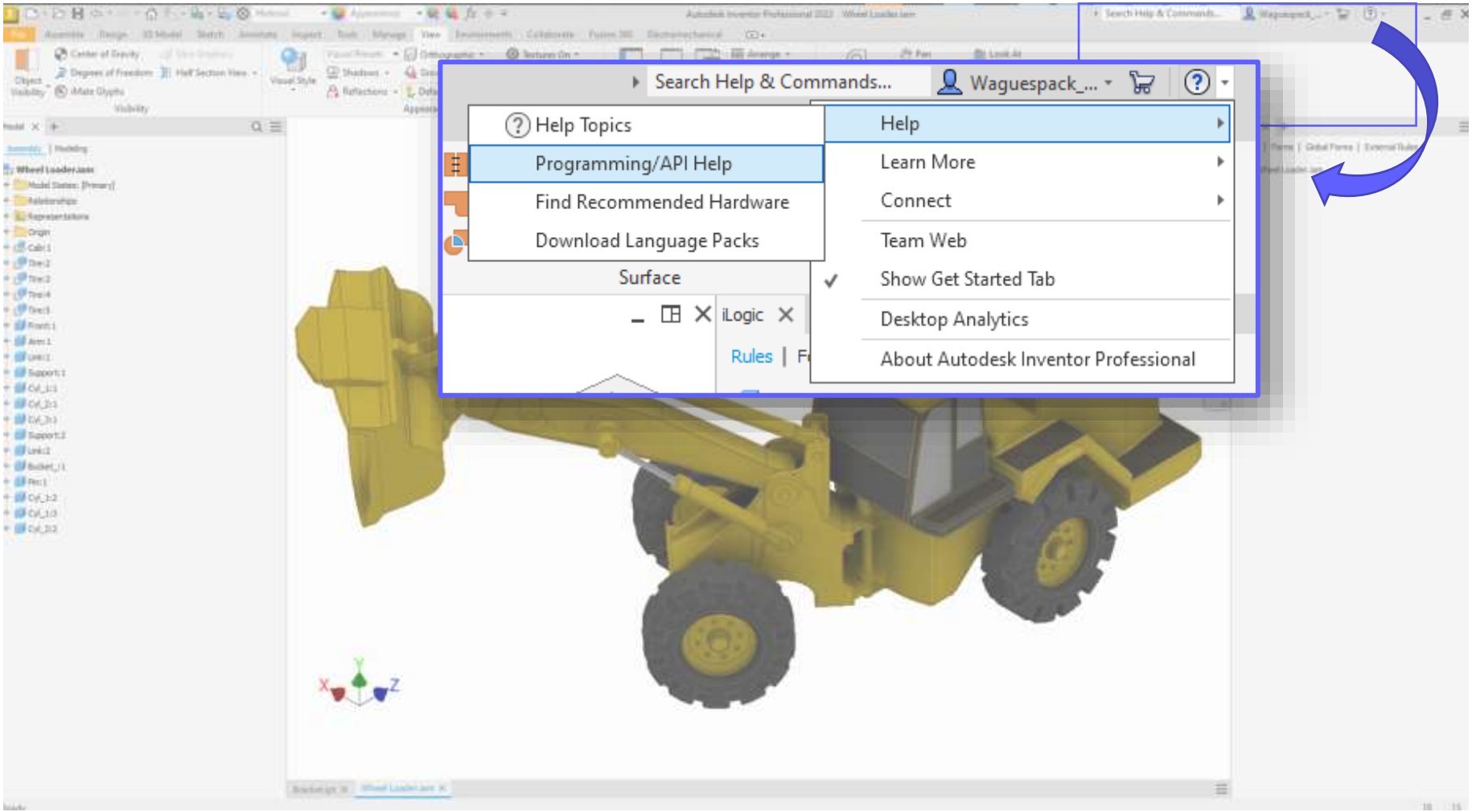
Inventor

Standard .Net



Access API and Programming Help Files

Tip #14


Search Help & Commands...
Waguespack_...
Help Topics
Programming/API Help
Find Recommended Hardware
Download Language Packs
Help
Learn More
Connect
Team Web
Show Get Started Tab
Desktop Analytics
About Autodesk Inventor Professional
Surface
iLogic
Rules

Autodesk Inventor 2023 API Help

Hide Locate Back Forward Print Options

Contents Index Search

- Using the Inventor Software Developer
- What's New in the Inventor API
- Inventor API User's Manual
- Inventor API Reference Manual
- Sample Programs

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.

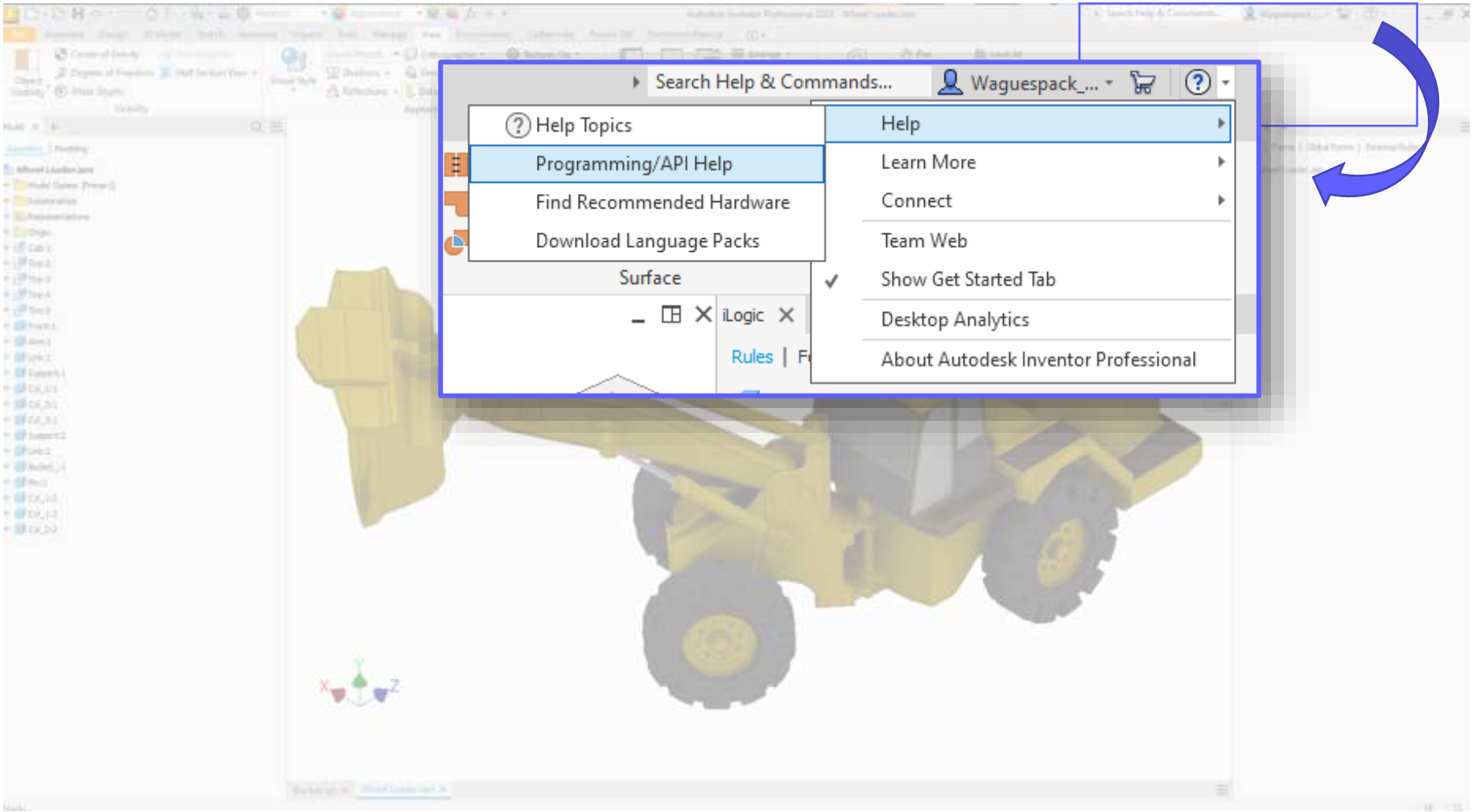
API Help

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.

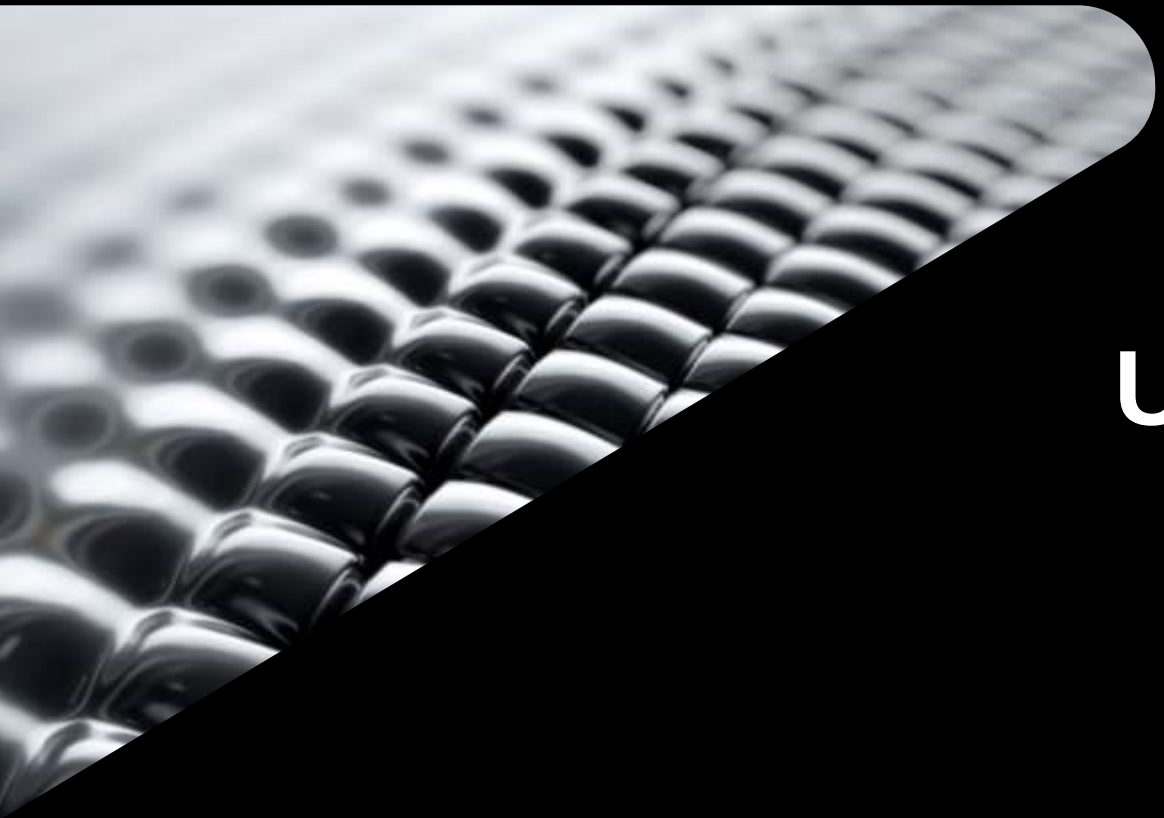
SDK Folder


Search Help & Commands...
Waguespack_...
Help Topics
Programming/API Help
Find Recommended Hardware
Download Language Packs
Help
Learn More
Connect
Team Web
Show Get Started Tab
Desktop Analytics
About Autodesk Inventor Professional
Surface
iLogic
Rules

AUTODESK UNIVERSITY

Objective:

Explore development and error handling techniques

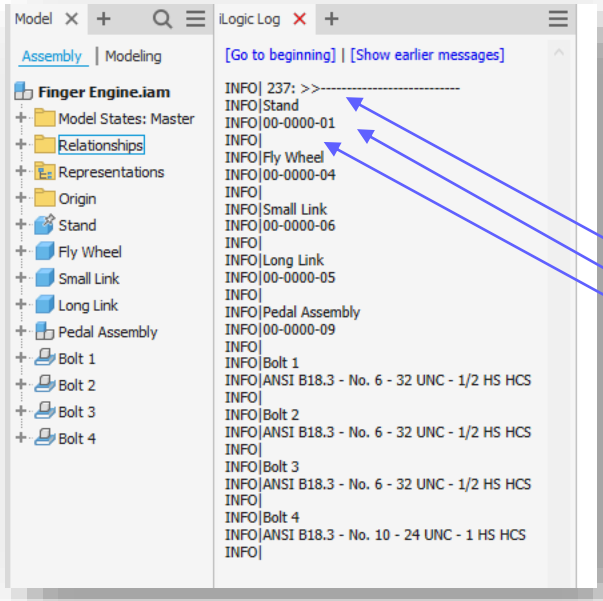


Use the iLogic Logger

Tip #15

What is the iLogic Logger?

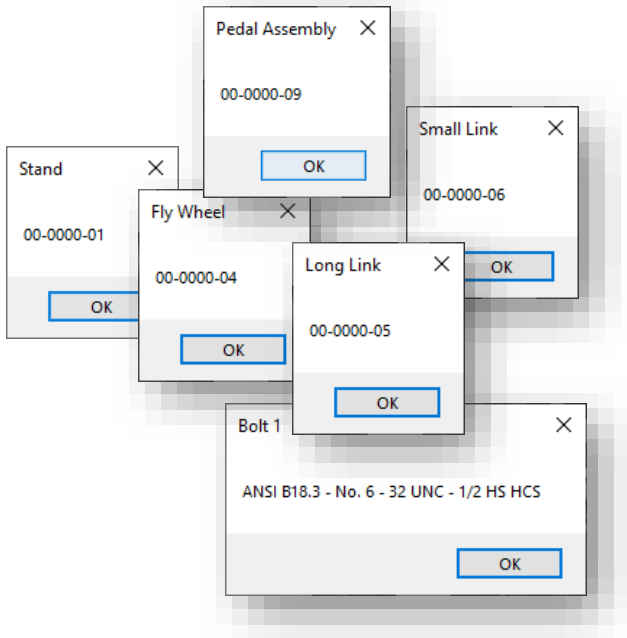
- The iLogic Logger is a tool that we can use to view or report information from our iLogic rules
- This could be model information, errors, etc
- In this example a rule writes out the occurrence name and part number to the logger




```
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

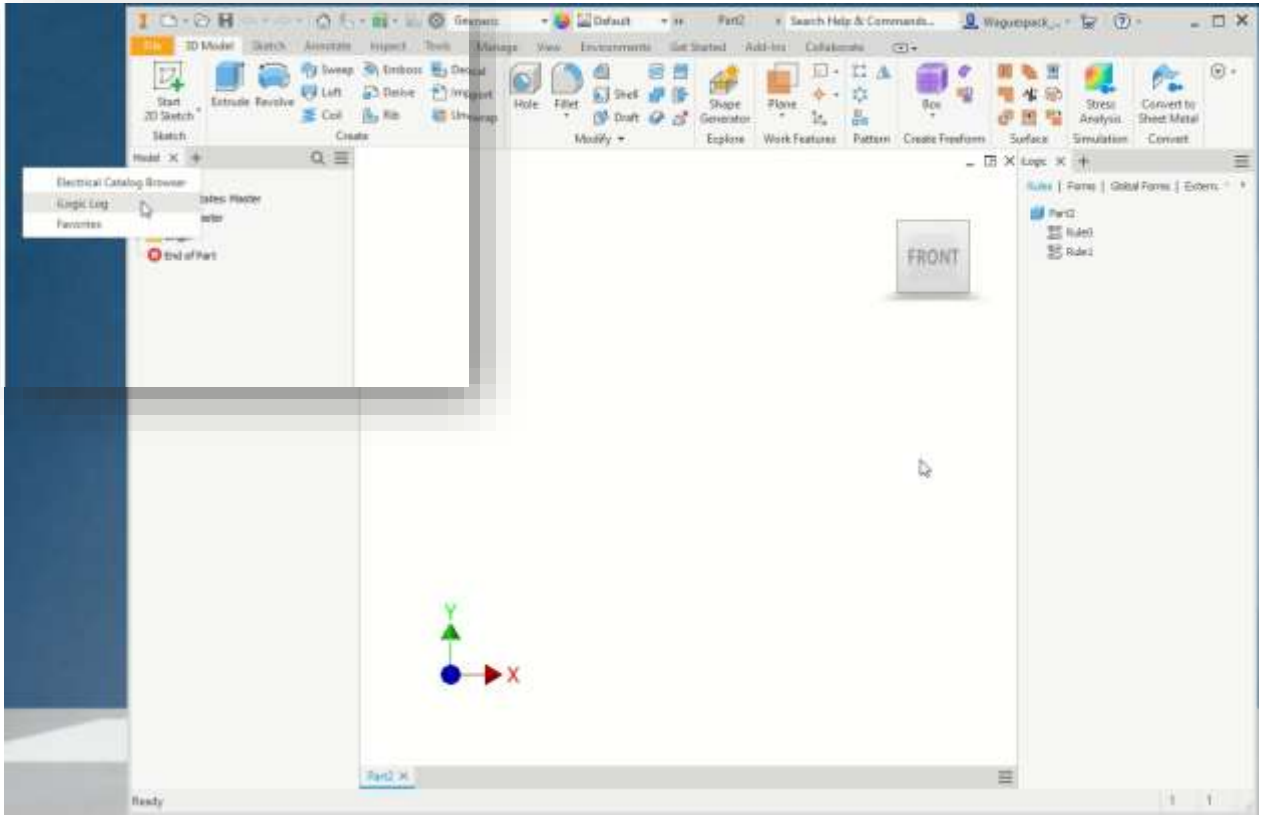



```
1   Dim oDoc As AssemblyDocument
2   oDoc = ThisDoc.Document
3
4   Dim oOcc As ComponentOccurrence
5   For Each oOcc In oDoc.ComponentDefinition.Occurrences
6
7       oPn = iProperties.Value(oOcc.Name, "Project", "Part Number")
8       MessageBox.Show(oPn, oOcc.Name)
9
10  Next
```

Accessing the iLogic Logger

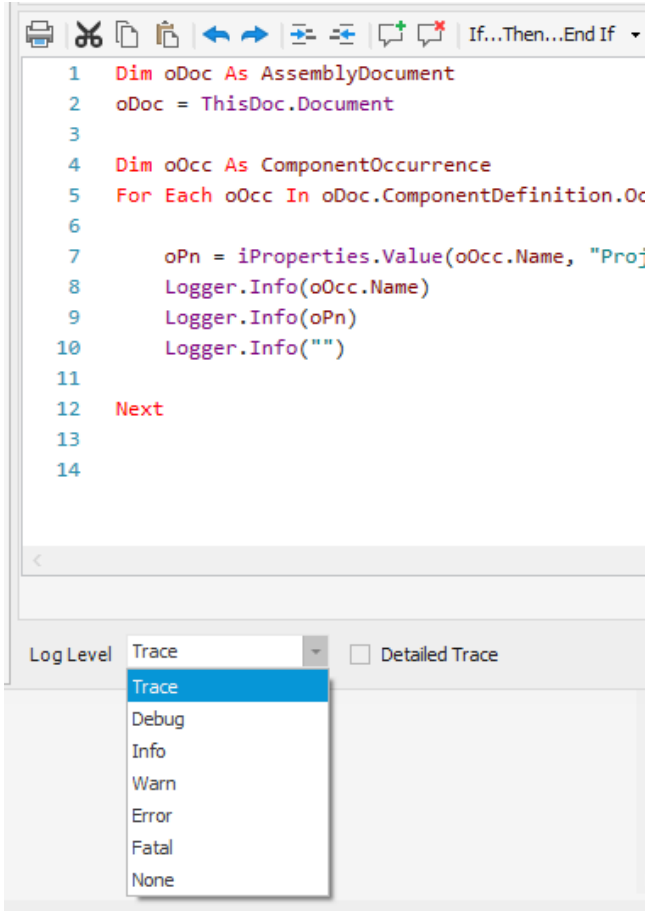
- We access the iLogic logger using the + next the standard Model browser

- The Logger can be docked to a location and position of your liking



Logger Levels

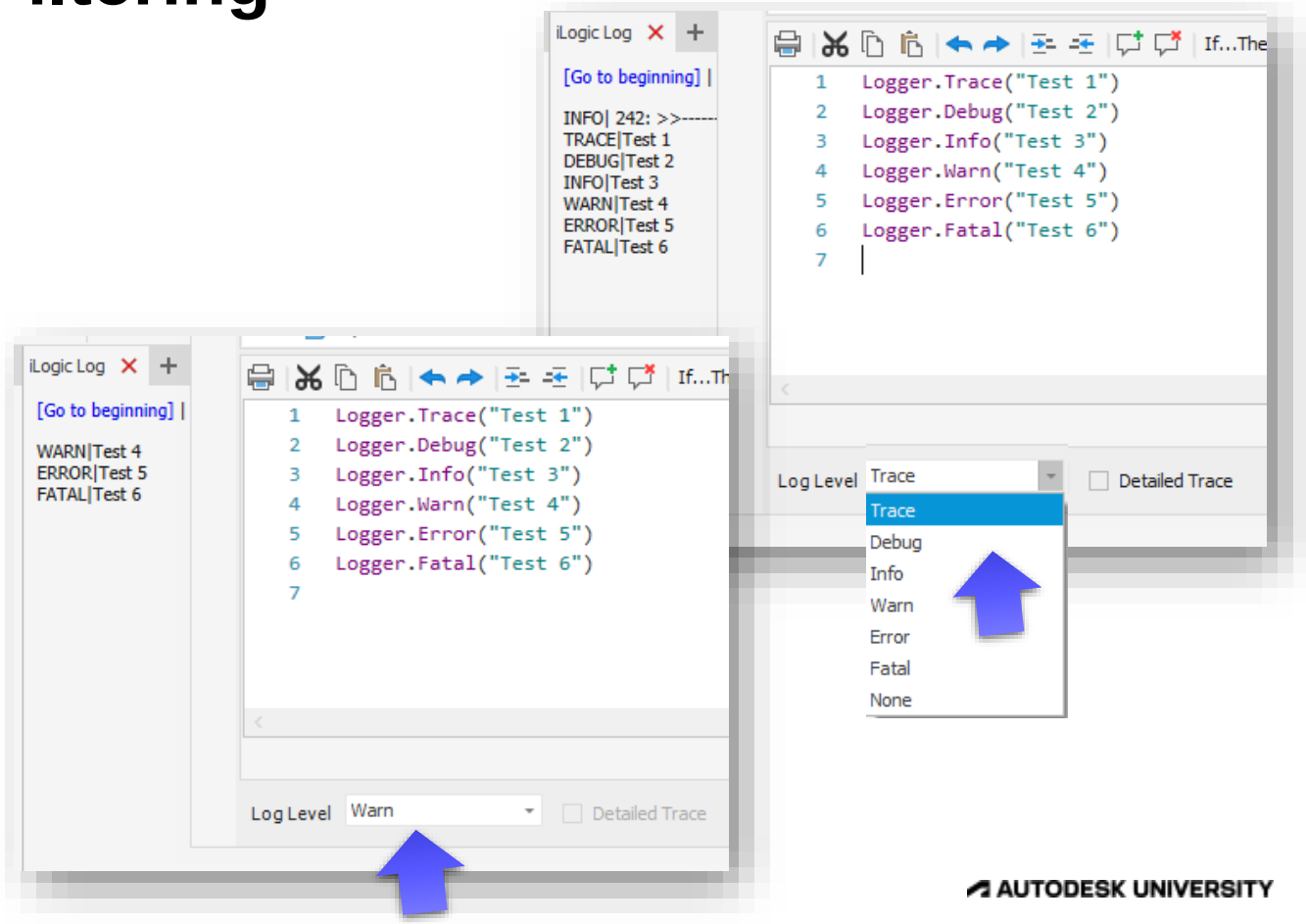
- The logger offers multiple levels of logging accessed via the Log Level drop down in the iLogic code editor



Logger Level Filtering

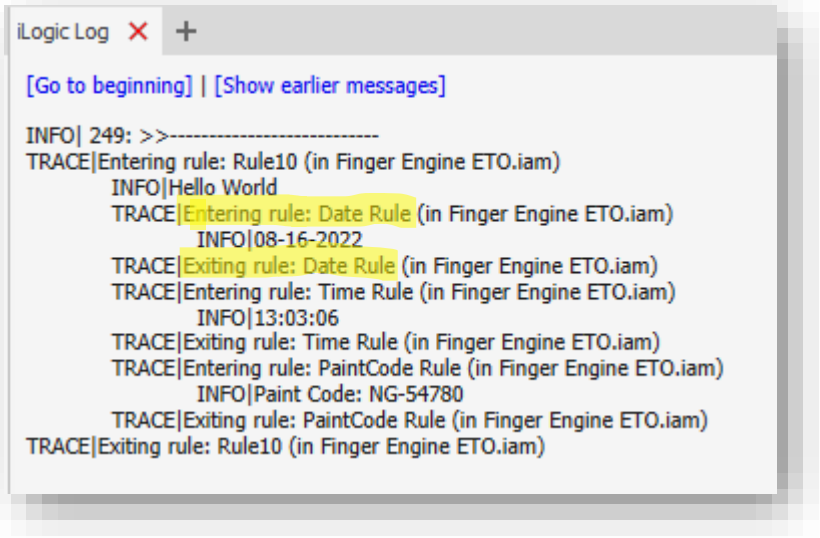
- The levels are cascading in nature, meaning that if you set the Log Level to the highest level (Trace) you will see all other levels.

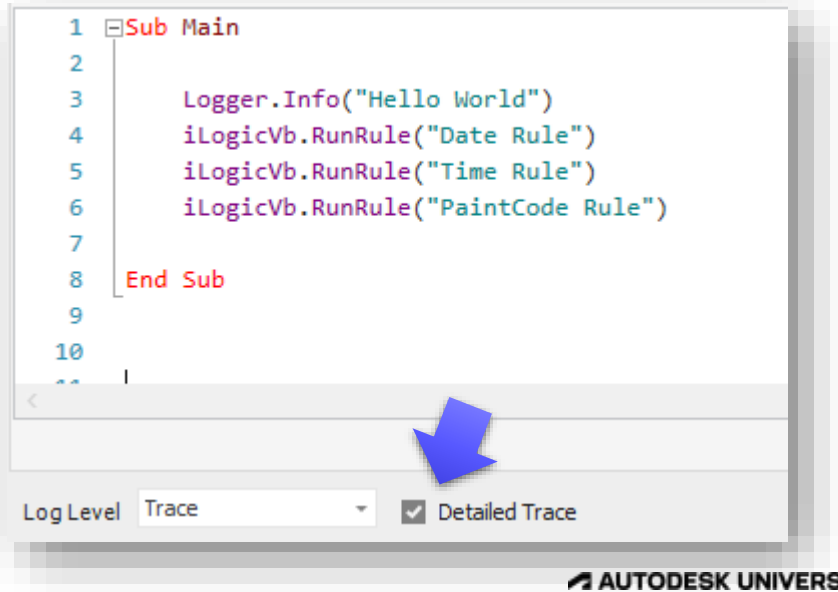
- If you were to set it to a lower level (such as **Warn**) you would only see levels at or below that level

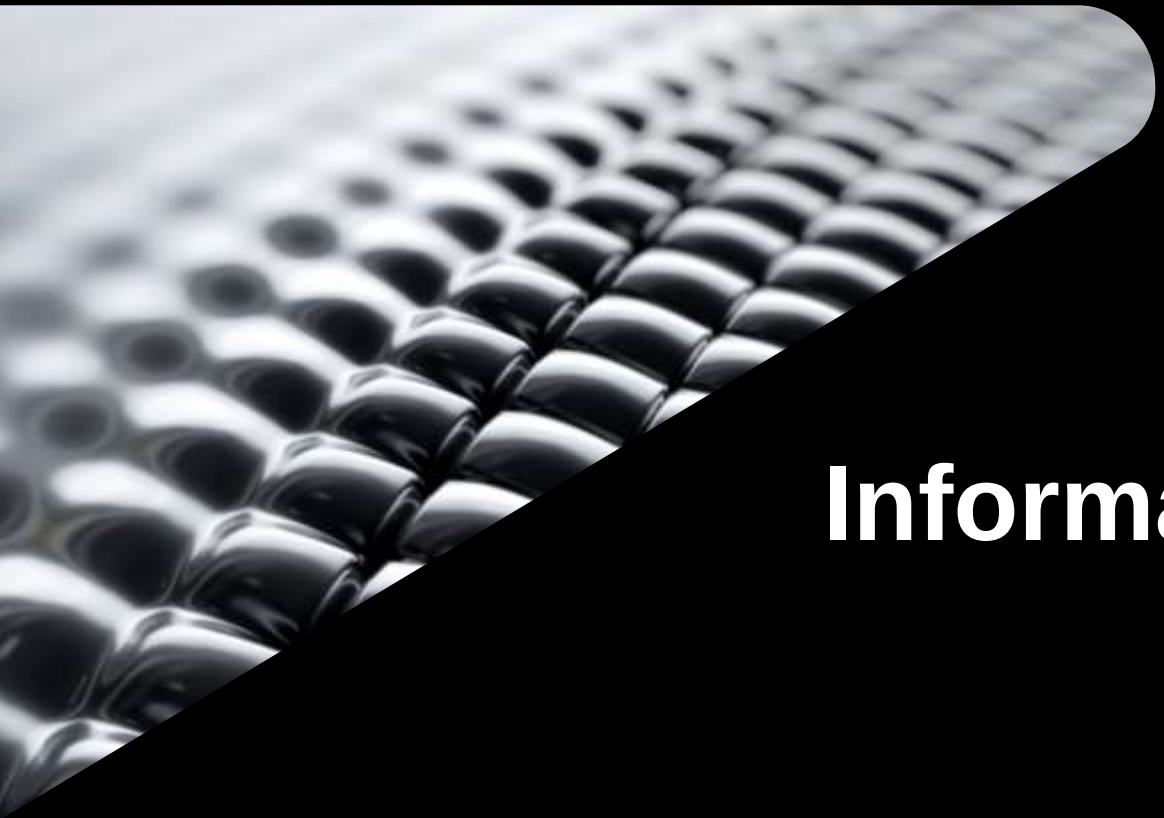


Detailed Trace

- 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





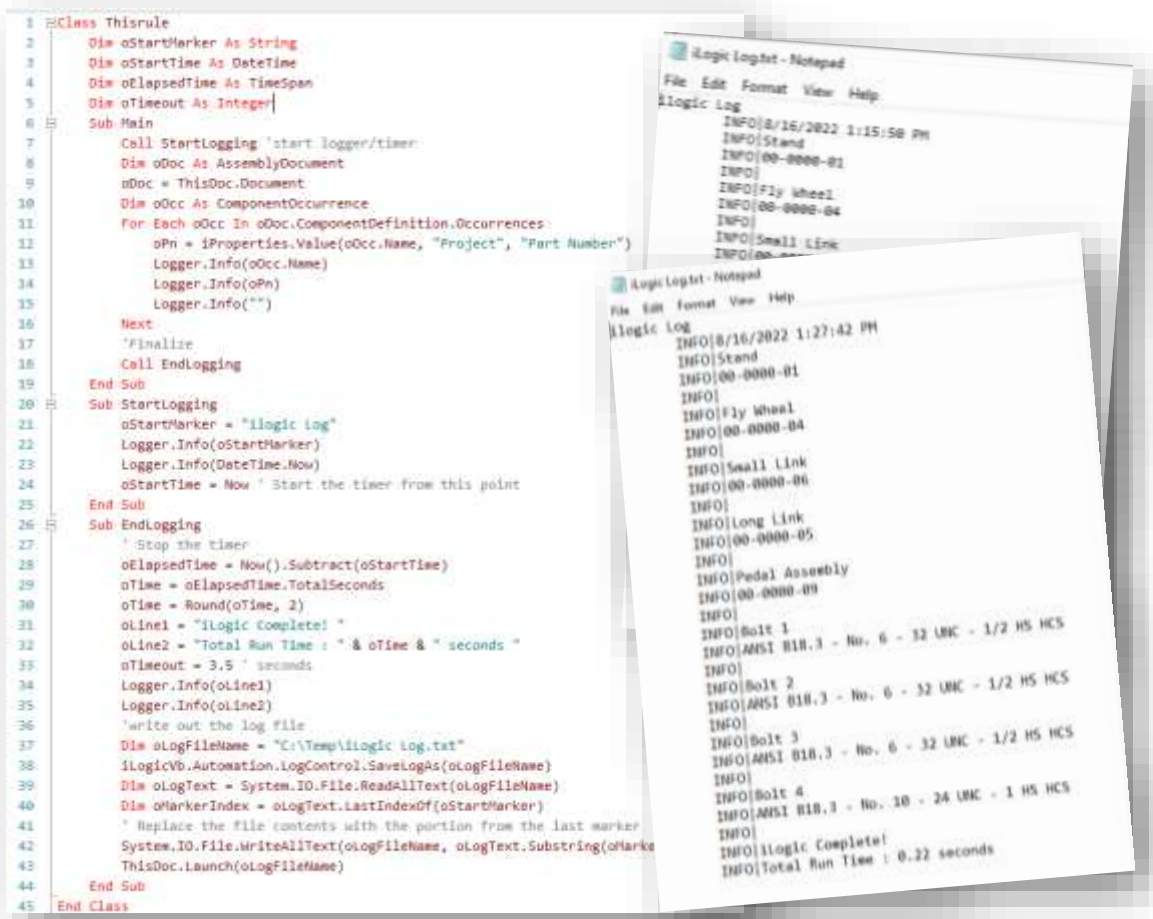


Write Logger Information to a File

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



```
Class Thisrule
    Dim oStartMarker As String
    Dim oStartTime As DateTime
    Dim oElapsedTime As TimeSpan
    Dim oTimeout As Integer
    Sub Main
        Call StartLogging 'start logger/timer
        Dim oDoc As AssemblyDocument
        oDoc = ThisDoc.Document
        Dim oOcc As ComponentOccurrence
        For Each oOcc In oDoc.ComponentDefinition.Occurrences
            oPn = iProperties.Value(oOcc.Name, "Project", "Part Number")
            Logger.Info(oOcc.Name)
            Logger.Info(oPn)
            Logger.Info("")
        Next
        'Finalize
        Call EndLogging
    End Sub
    Sub StartLogging
        oStartMarker = "iLogic Log"
        Logger.Info(oStartMarker)
        Logger.Info(DateTime.Now)
        oStartTime = Now ' Start the timer from this point
    End Sub
    Sub EndLogging
        ' Stop the timer
        oElapsedTime = Now().Subtract(oStartTime)
        oTime = oElapsedTime.TotalSeconds
        oTime = Round(oTime, 2)
        oLine1 = "iLogic Complete! "
        oLine2 = "Total Run Time : " & oTime & " seconds "
        oTimeout = 3.5 ' seconds
        Logger.Info(oLine1)
        Logger.Info(oLine2)
        'write out the log file
        Dim oLogFileName = "C:\Temp\iLogic Log.txt"
        iLogicVb.Automation.LogControl.SaveLogAs(oLogFileName)
        Dim oLogText = System.IO.File.ReadAllText(oLogFileName)
        Dim oMarkerIndex = oLogText.LastIndexOf(oStartMarker)
        ' Replace the file contents with the portion from the last marker
        System.IO.File.WriteAllText(oLogFileName, oLogText.Substring(oMarke
        ThisDoc.Launch(oLogFileName)
    End Sub
End Class
```

iLogic Log.txt - Notepad

File Edit Format View Help

```
iLogic Log
        INFO|8/16/2022 1:15:58 PM
        INFO|Stand
        INFO|00-0000-01
        INFO|
        INFO|Fly Wheel
        INFO|00-0000-04
        INFO|
        INFO|Small Link
```

iLogic 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|00-0000-06
        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))
```

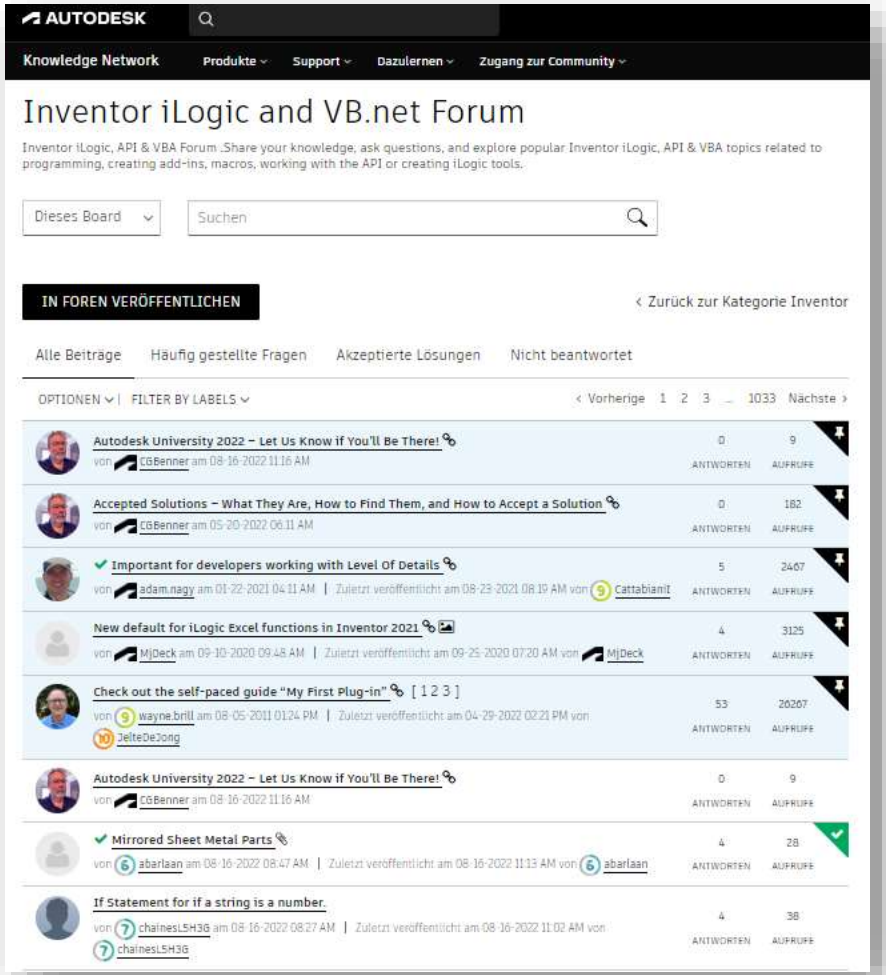
AUTODESK
Mike Deck
Software Developer
Autodesk, Inc.

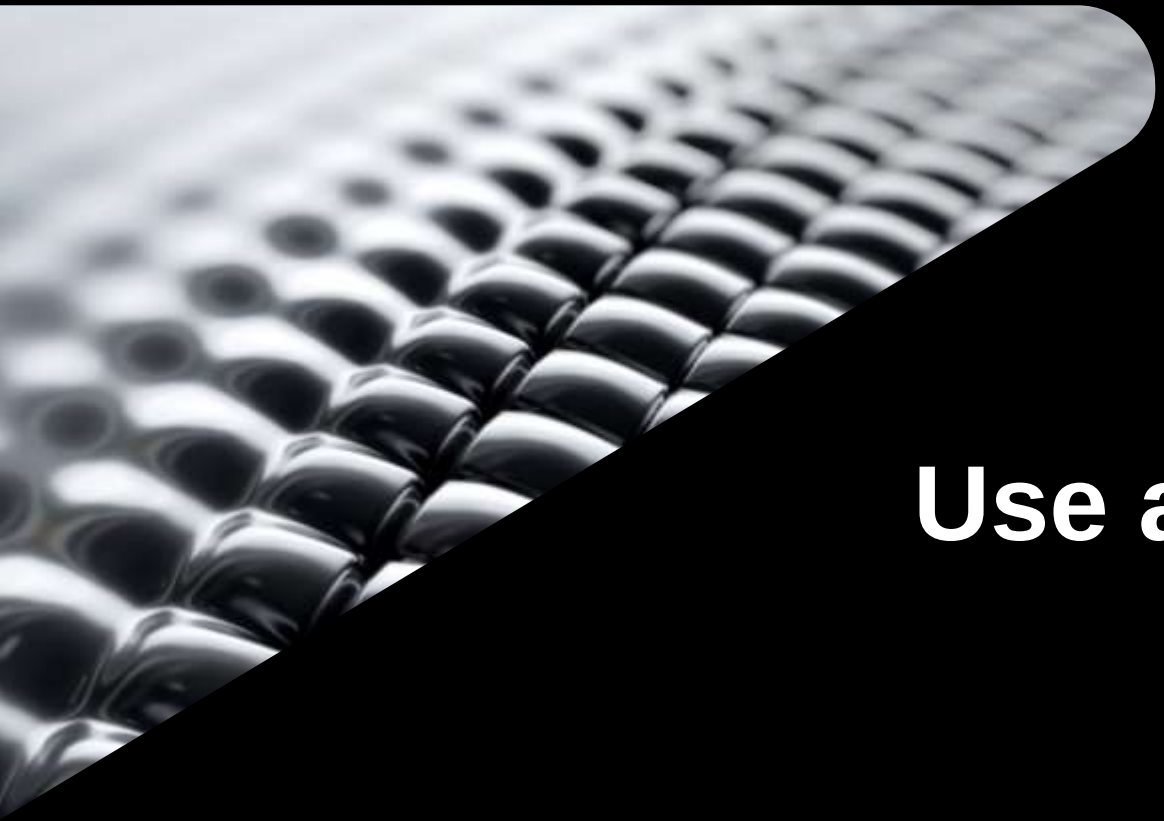
Inventor iLogic and VB.net Forum

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

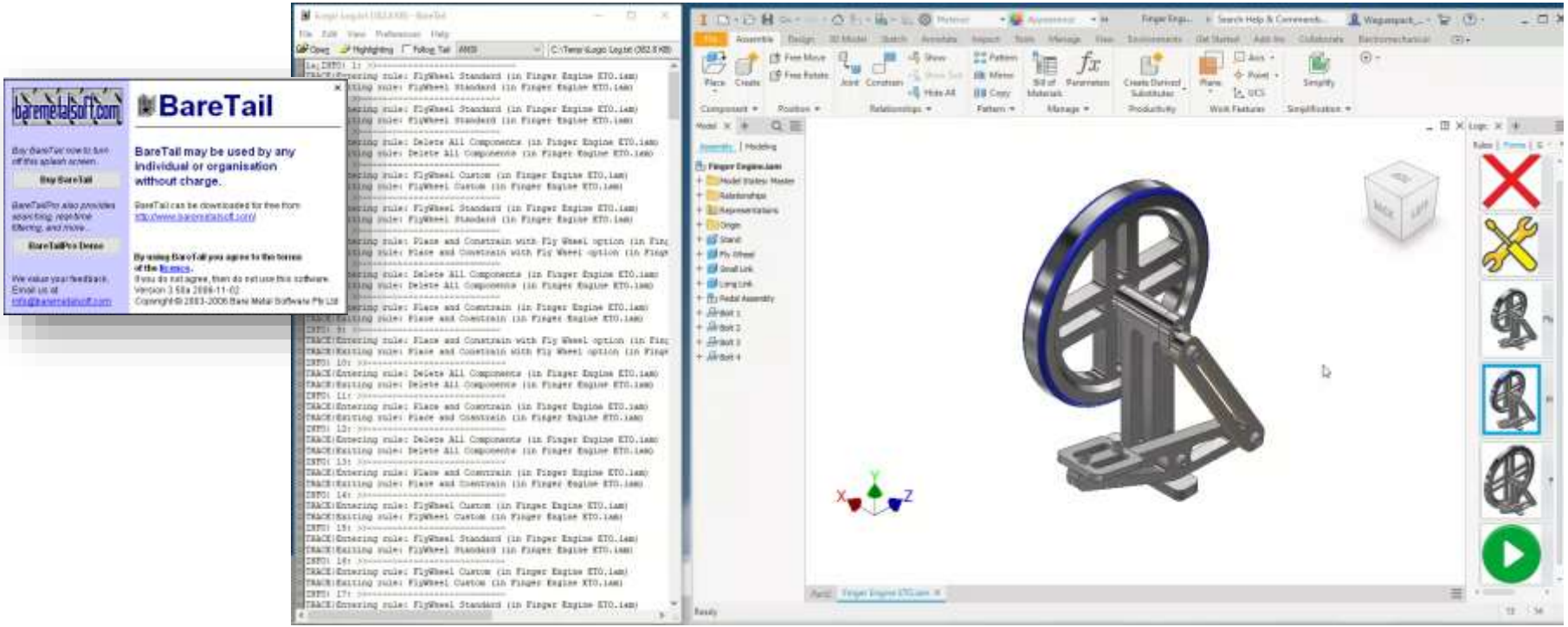




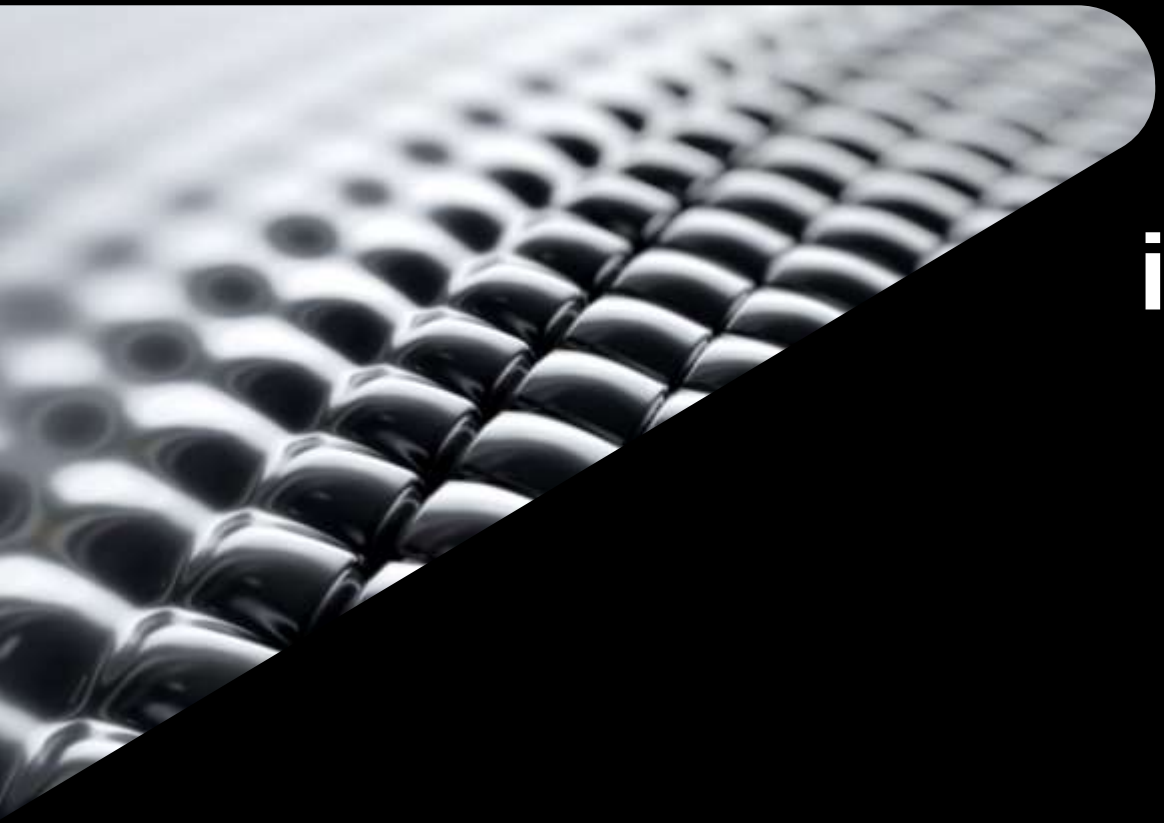
Use a Log Monitor

Tip #18

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



- Watch log outputs real time

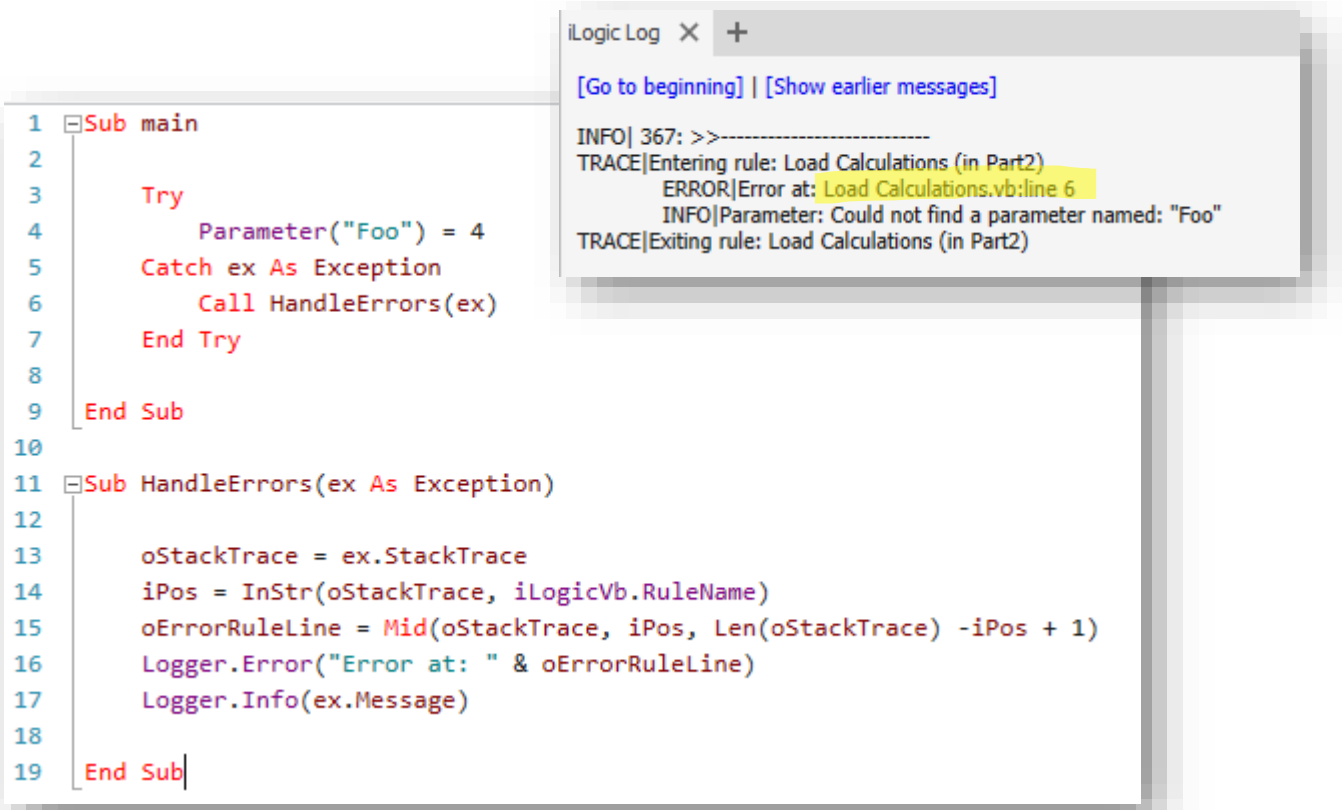


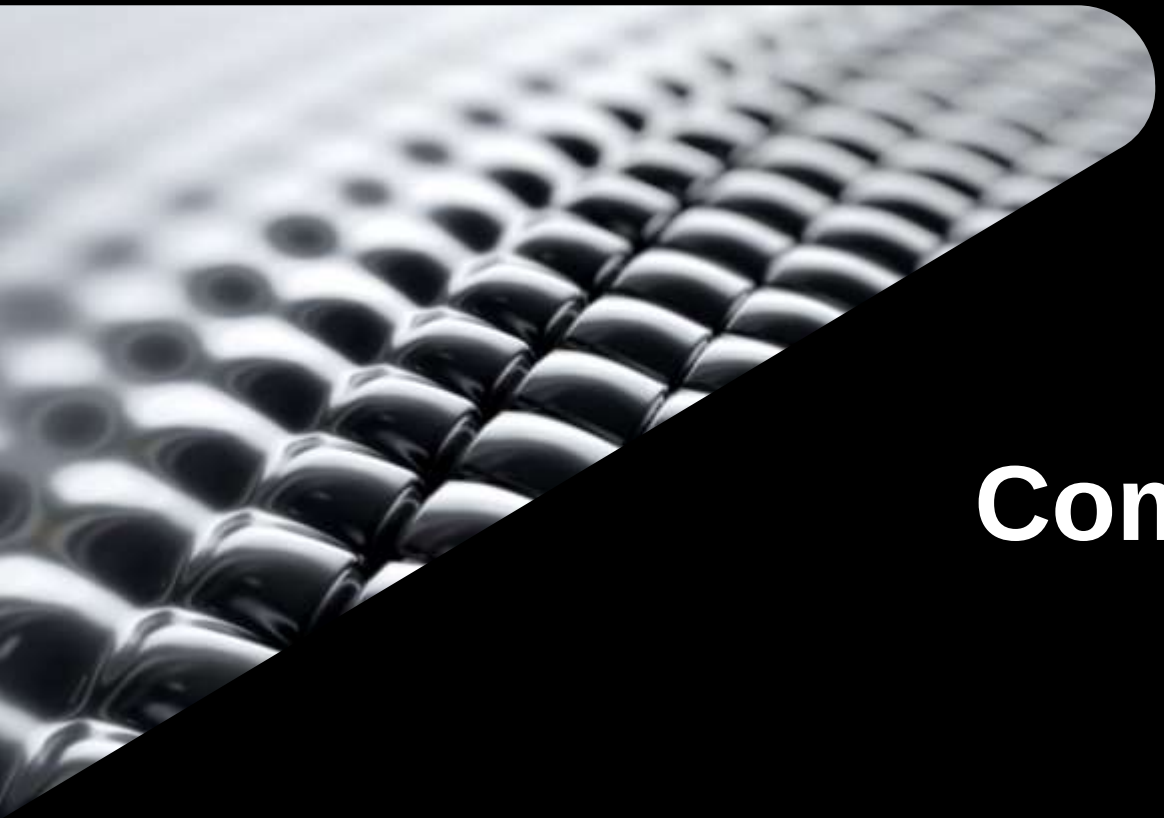
Use Try Catch Statements with the iLogic Logger to catch Exceptions

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



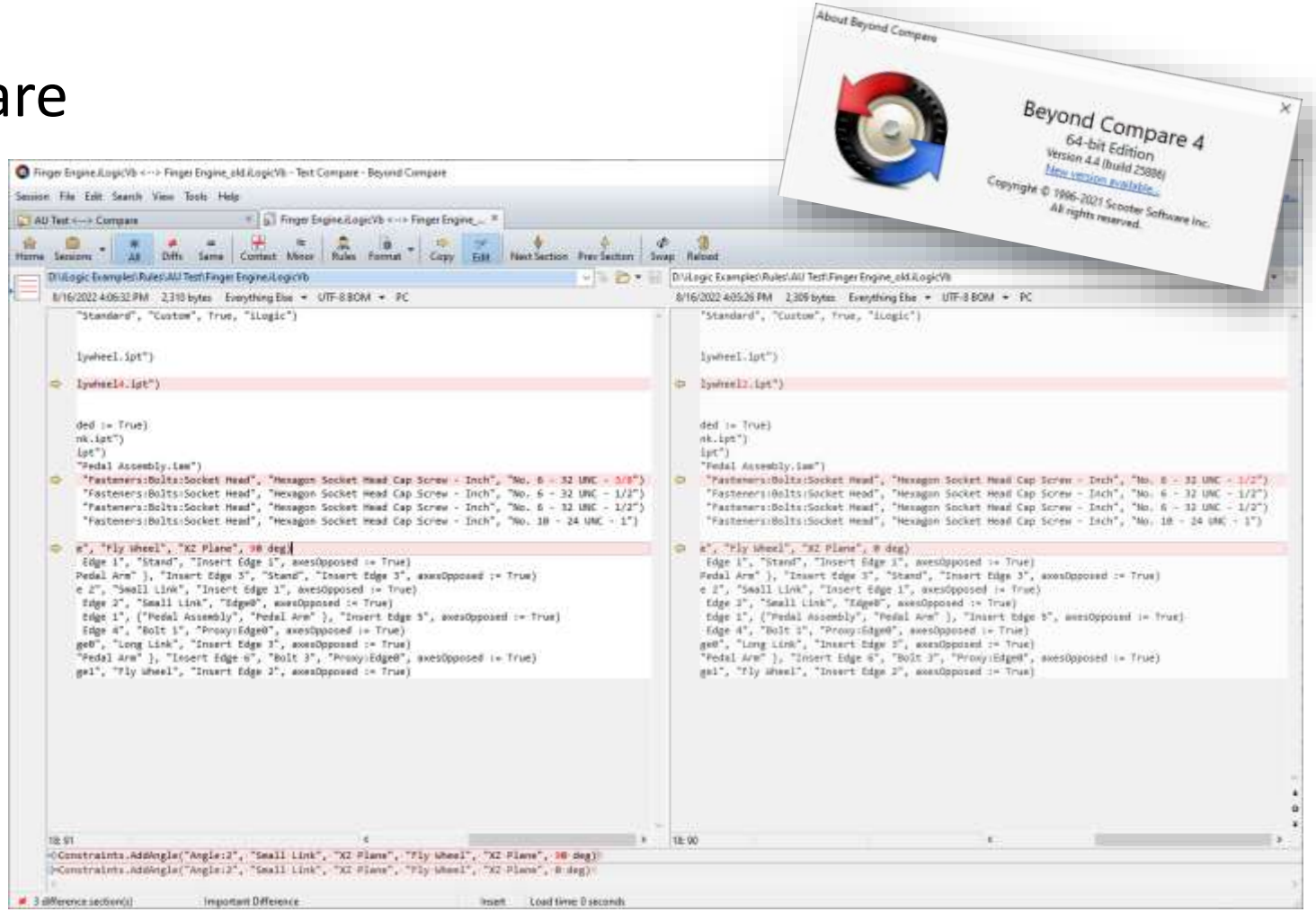


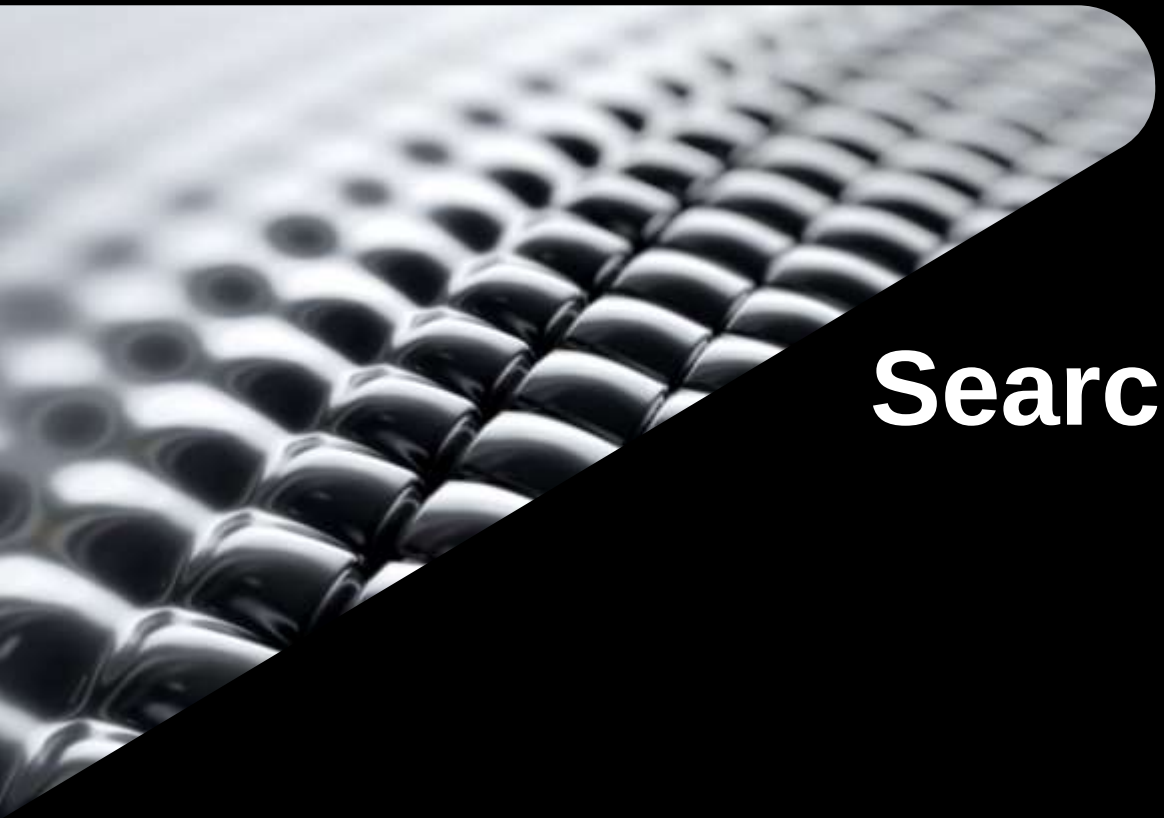
Use a Code Comparison Tool

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.



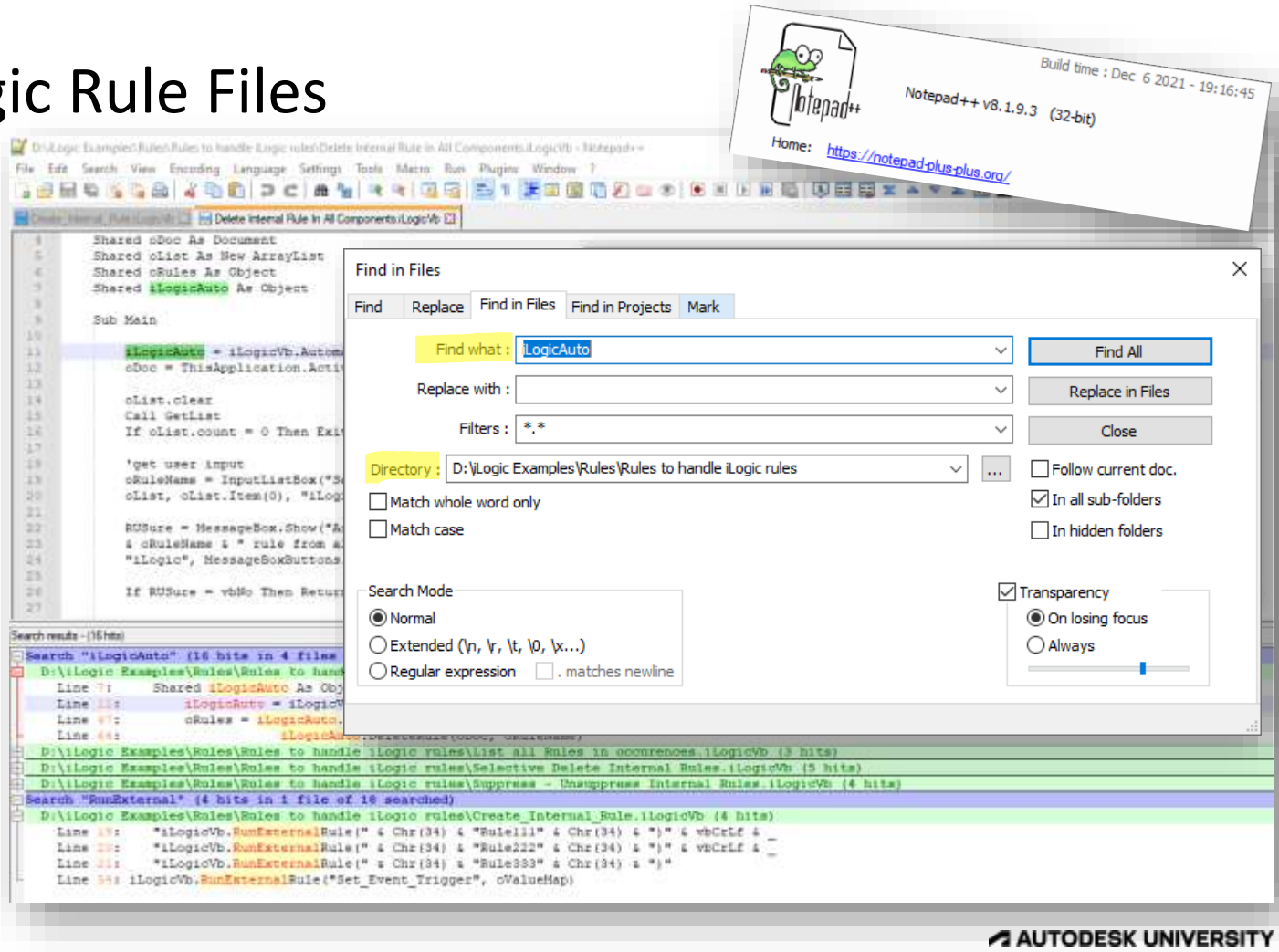


Search Within Your iLogic Rules

Tip #21

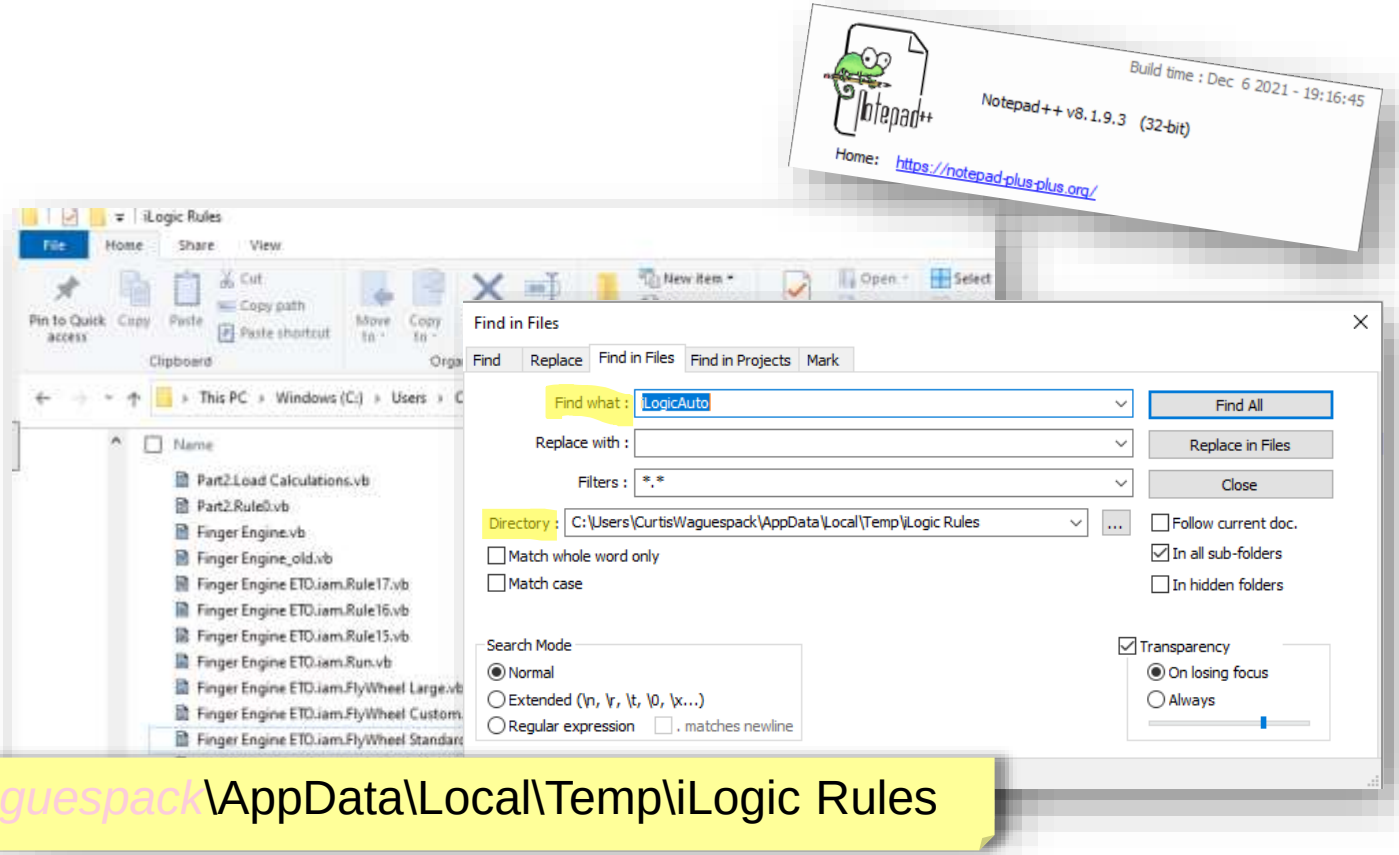
Search in iLogic Rule Files

- 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



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

AUTODESK UNIVERSITY

Objective

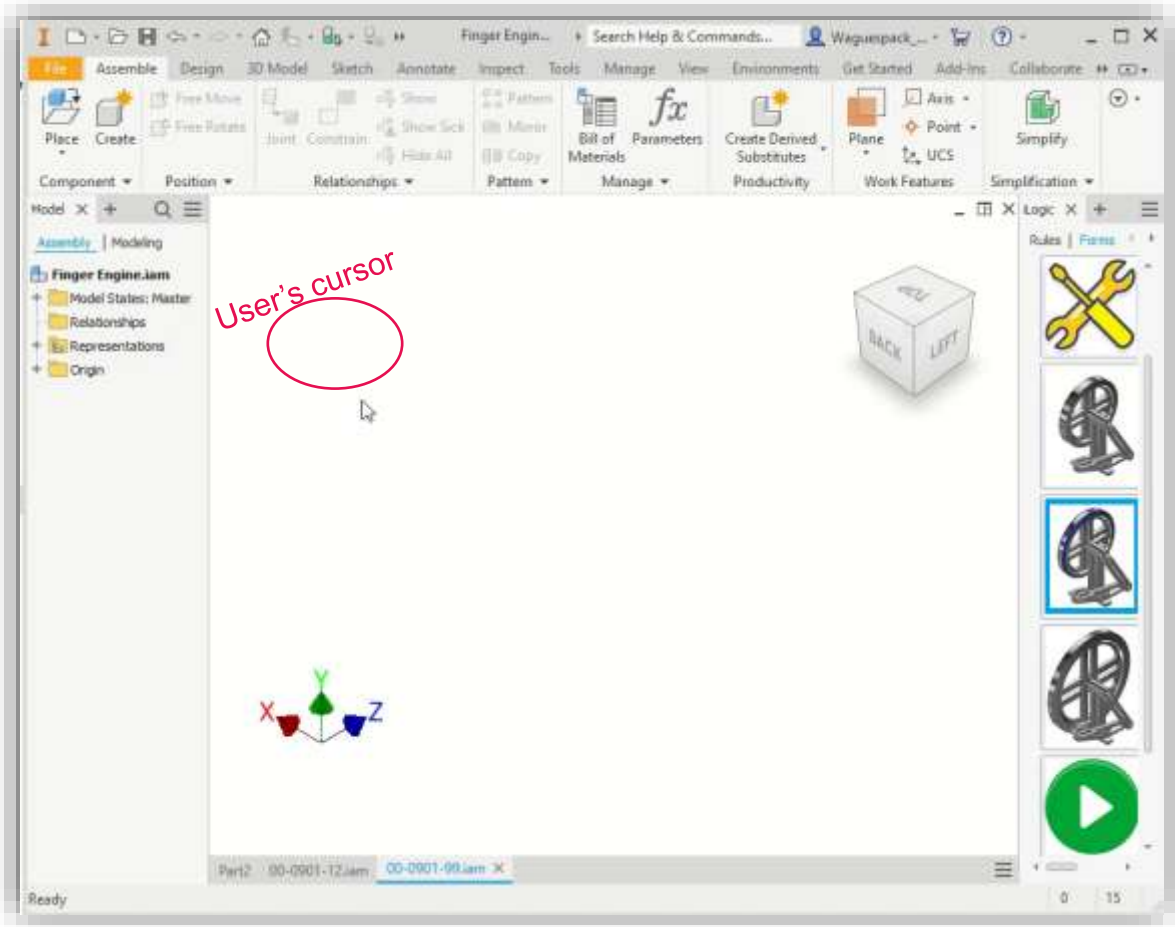
Create a Better Interface Between Your iLogic Automation and the Users Who Employ It

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 augments:
  - Message text “Model created successfully…”
  - a “Timeout” value of 2.5 seconds

Timed Message rule

```
Sub Main
    'set default
    oTimeout = 1.5 ' seconds
    oMsg = "Hello World"

    'get value map argument from other rule
    If Not RuleArguments("Timeout") = "" Then oTimeout = RuleArguments("Timeout")
    If Not RuleArguments("Message") = "" Then oMsg = RuleArguments("Message")

    Call TimedMessage(oMsg, oTimeout)

End Sub


Sub TimedMessage(oMsg As String, oTimeout As Double)

Dim oForm As New Form() With { .Enabled = True }
System.Threading.Tasks.Task.Delay(TimeSpan.FromSeconds(oTimeout)).ContinueWith(Sub(t)
oForm.Close()
End Sub ,
System.Threading.Tasks.TaskScheduler.FromCurrentSynchronizationContext())

MessageBox.Show(oForm, String.Format(oMsg, oTimeout), "iLogic")

End Sub
```

Calling Rule

```
InventorVb.DocumentUpdate()
ThisApplication.ActiveView.Fit

Dim oValueMap As Inventor.NameValueMap = ThisApplication.TransientObje CreateNameValueMap()
oValueMap.Add("Message", "Model created successfully." & vbLf & vbLf &  Preparing to run motion test rule.")
oValueMap.Add("Timeout", "2.5")
iLogicVb.RunExternalRule("Timed Message", oValueMap)

iLogicVb.RunRule("Run")
```

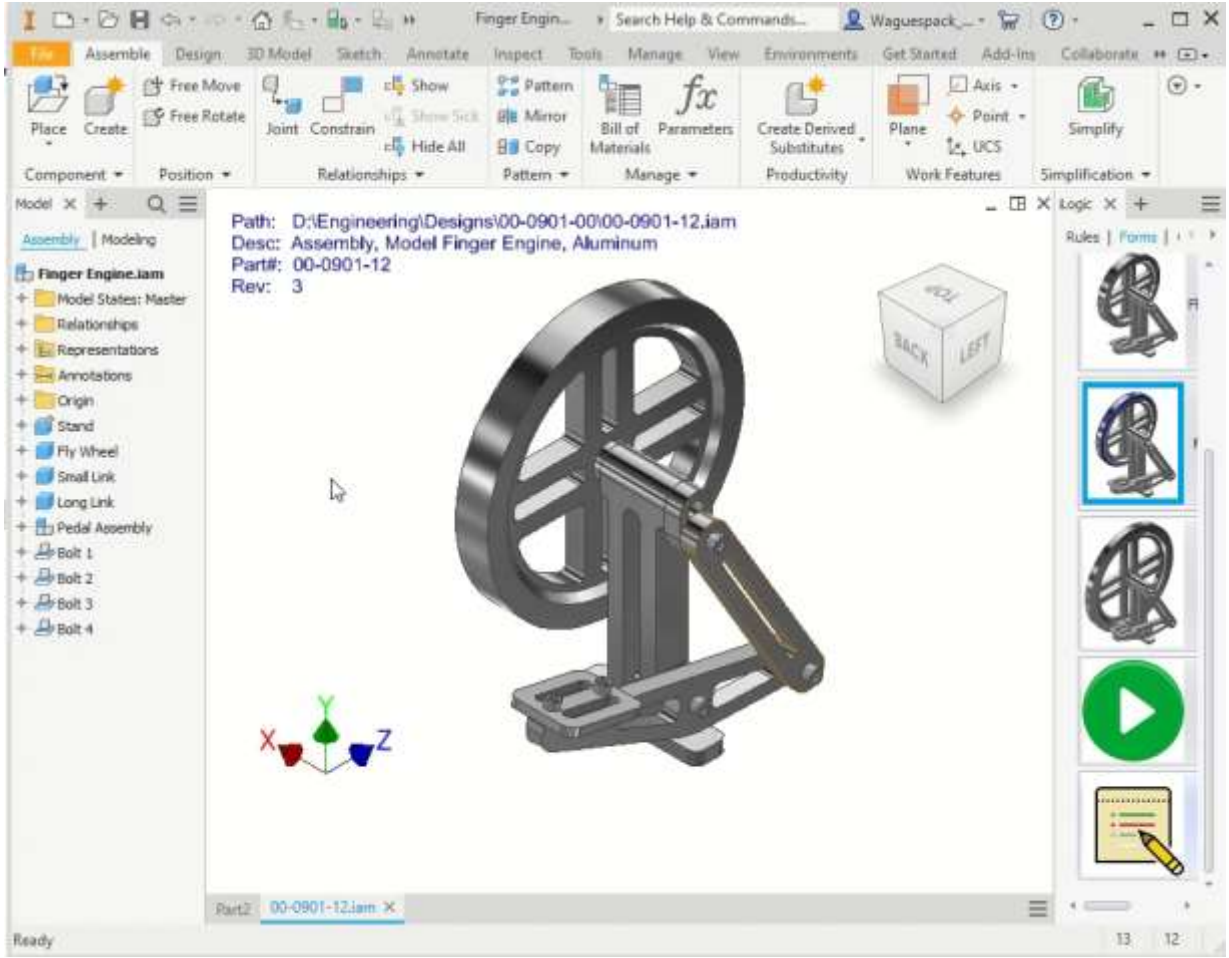
Display Information in the graphics area

Tip #23

Annotation Information

Tip provided by **Andrew Humiston**

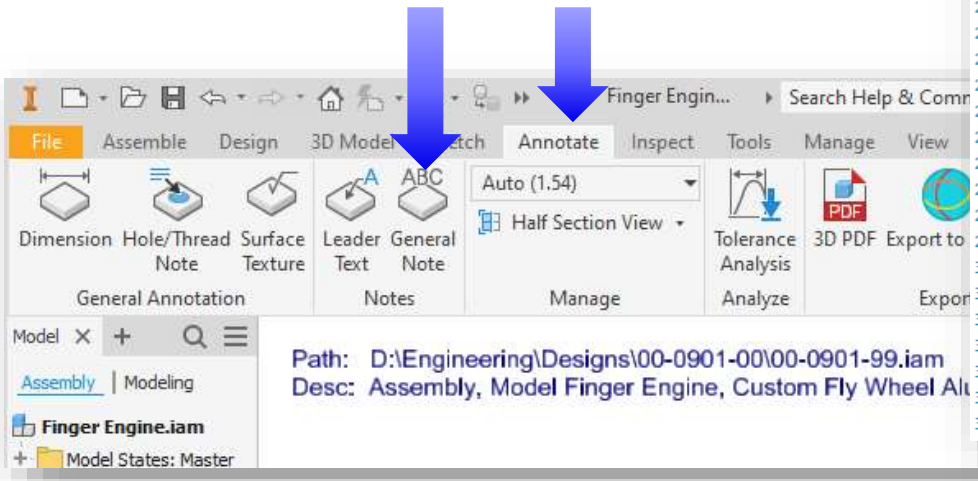
- Use a General Annotation Note to display file information to your users.

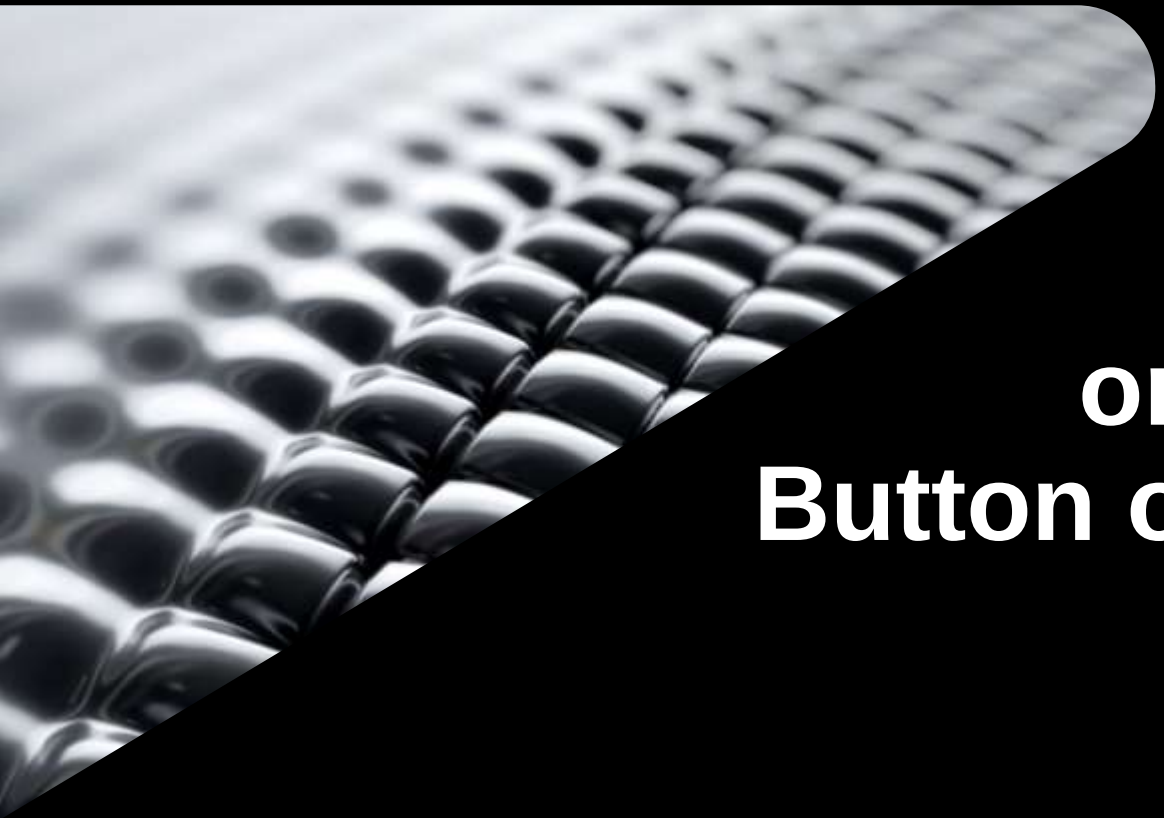


Annotation Note

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

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

    'get general notes collection
    sNotes = oDoc.ComponentDefinition.ModelAnnotations.ModelGeneralNotes

    Dim oInfoNote As ModelGeneralNote
    Dim oNoteDef As ModelGeneralNoteDefinition

    Try : sNotes.Item("Information Note").delete : Catch : End Try
    dFontSize = 0.12
    oNoteDef = sNotes.CreateDefinition("", True, kUpperLeftQuadrant)
    oNoteDef.Text.Size = dFontSize
    oInfoNote = sNotes.Add(oNoteDef)
    oInfoNote.Name = "Information Note"

    'get existing note text
    sString = sNotes.Item(1).Definition.Text.FormattedText

    sValue = oDoc.FullFileName
    sText = sText & "Path:    " & sValue & "<Br/>"

    sName = "Description"
    sValue = iProperties.Value("Project", sName)
    sText = sText & "Desc:   " & sValue & "<Br/>"

    sFormattedText = FormatText(sText, dFontSize)
    sNotes.Item(1).Definition.Text.FormattedText = sFormattedText
End Sub

Function FormatText(sString1 As String, dFontSize As Double)

    sFormattedText = "<StyleOverride " & "FontSize='" & dFontSize & "' " & _
    ">" & sString1 & "</StyleOverride>" & "<Br/>"
    Return sFormattedText
End Function
```

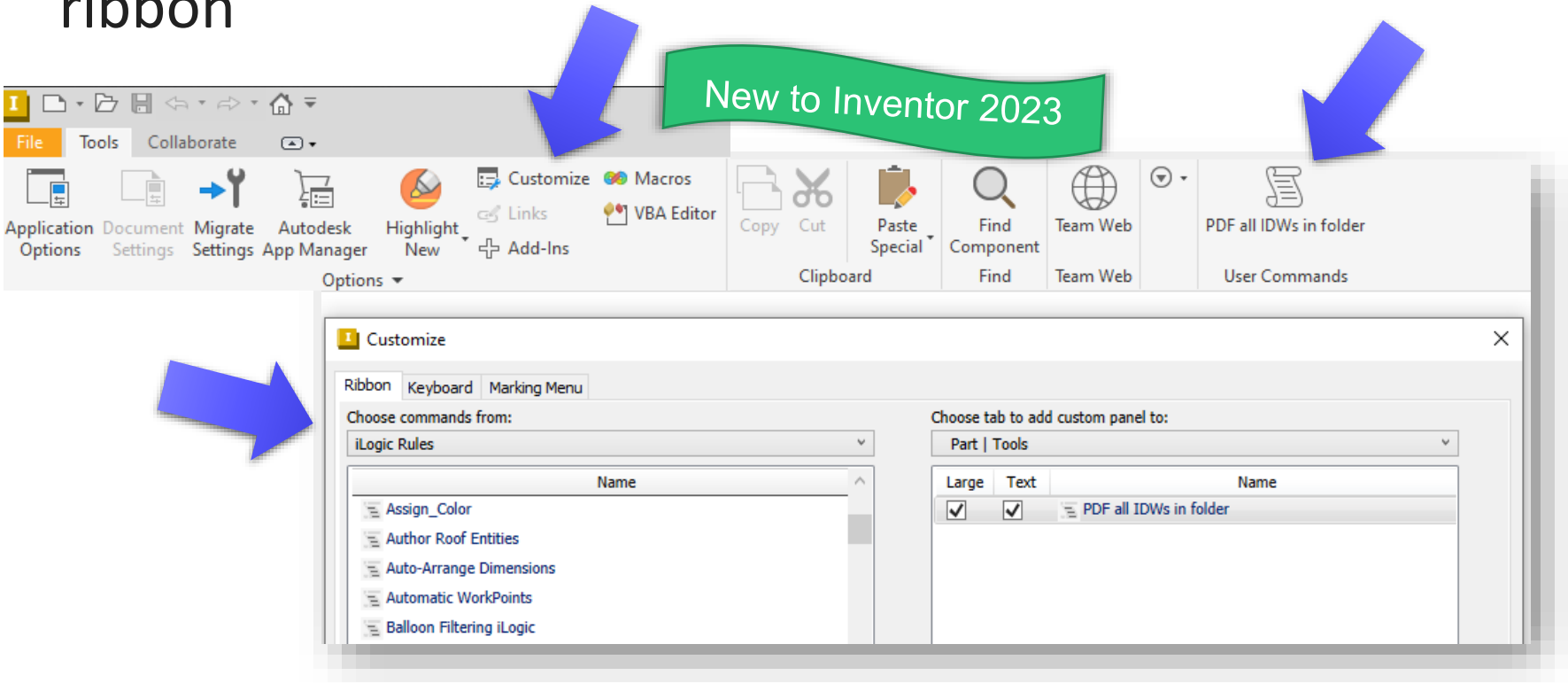




External Rule or Global Form Button on the Ribbon

Tip #24

Add buttons for External Rules and Global Forms to the ribbon

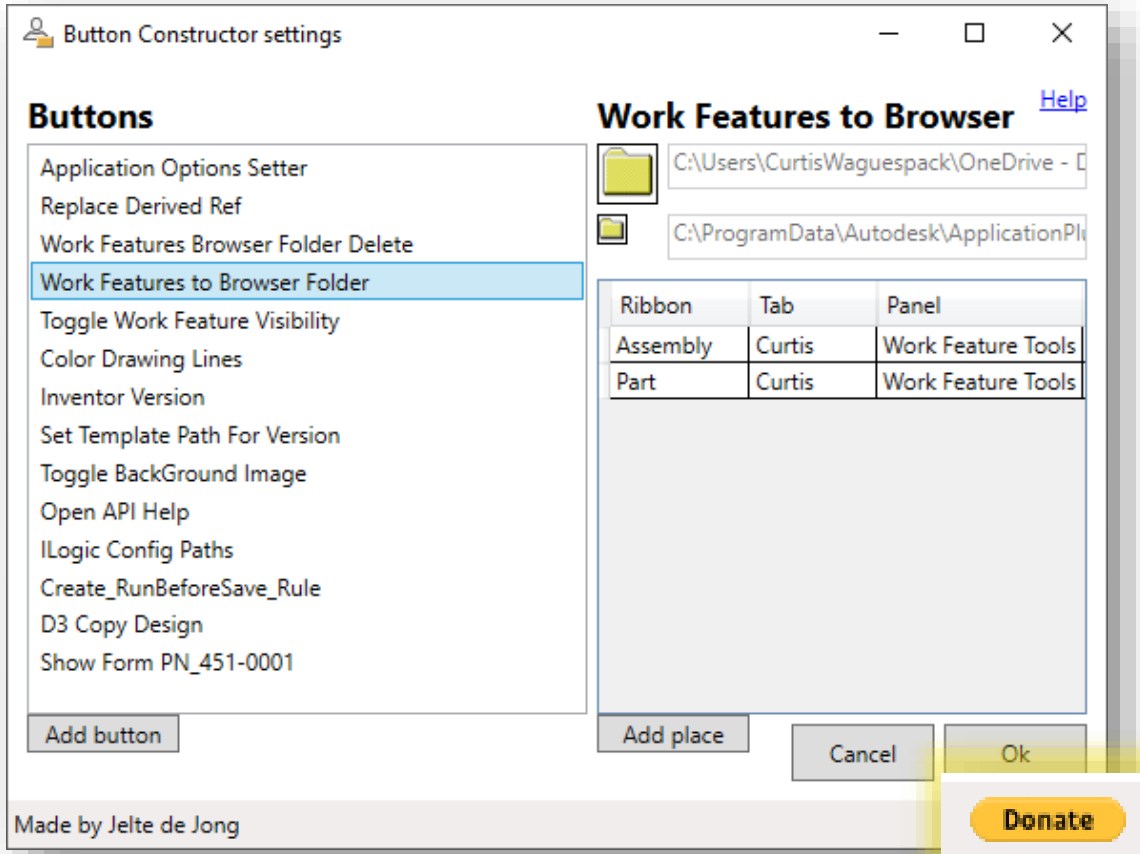


Button Constructor Add in

Get it at from the Autodesk App Store

Created by: Jelte de Jong

www.hjalte.nl



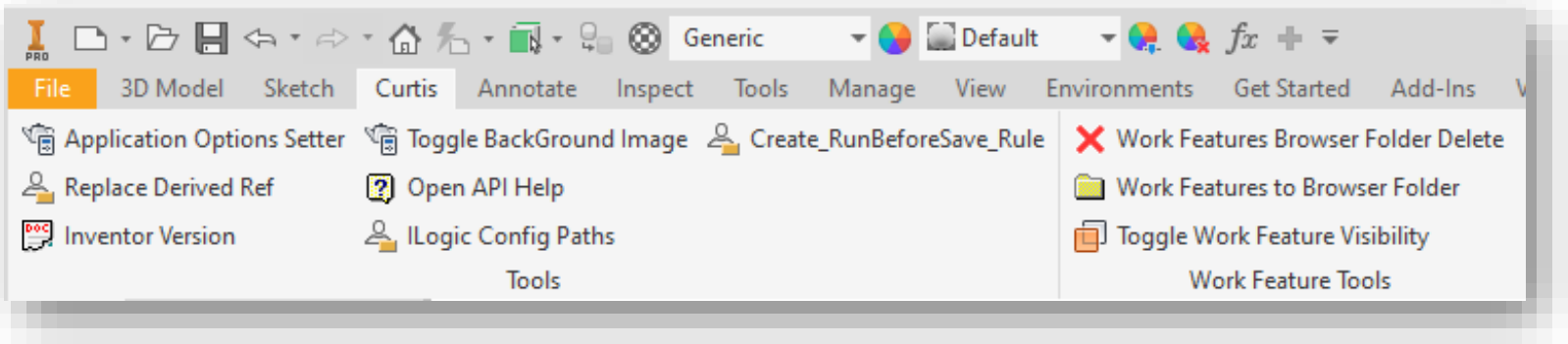
Button Constructor Add in

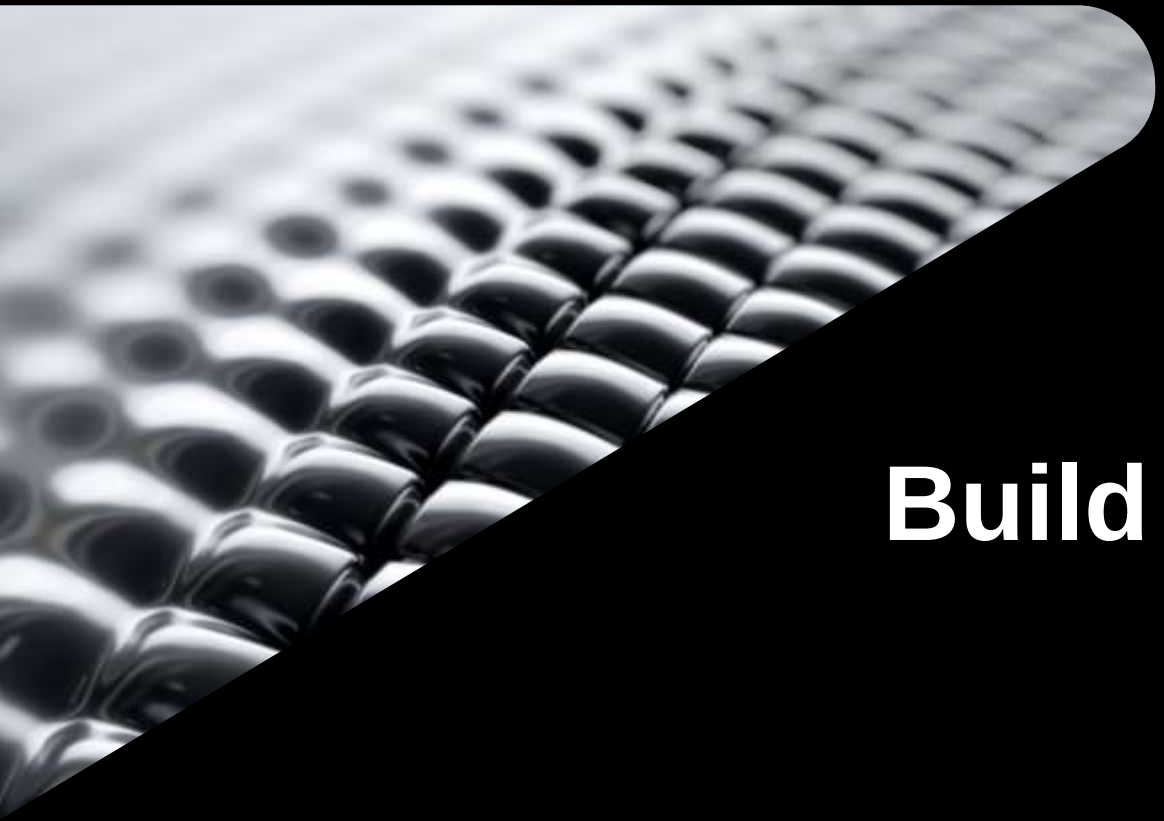
Get it at from the
Autodesk App Store

Created by: Jelte de Jong

Example:

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



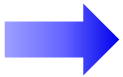


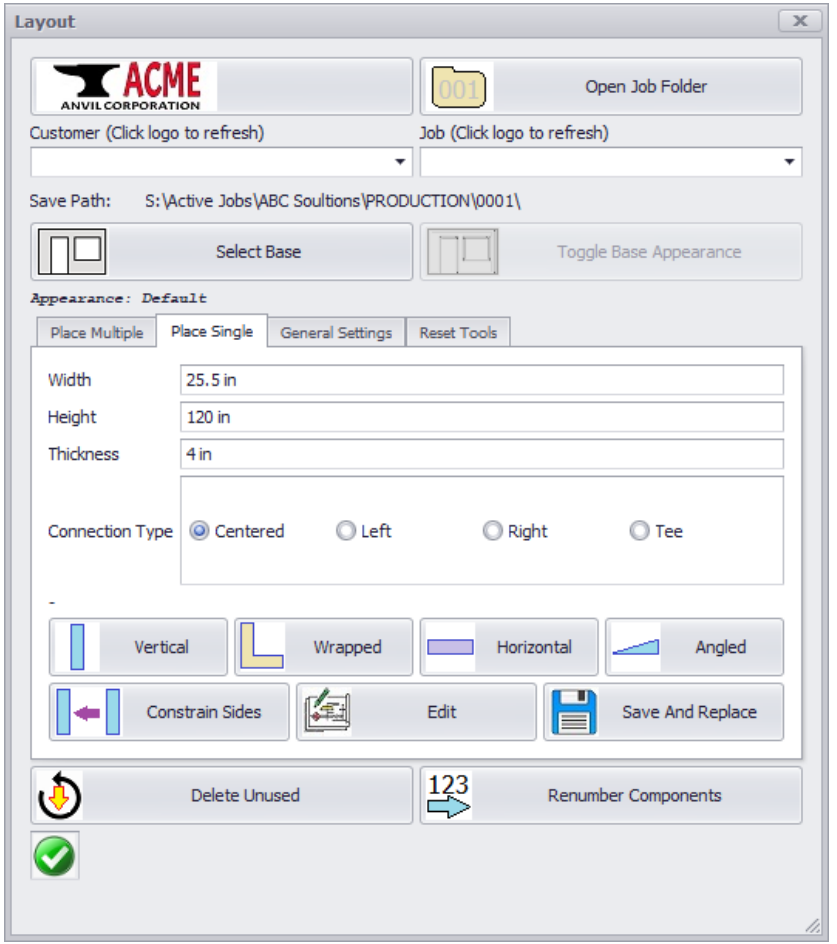
Build Better Forms

Tip #25

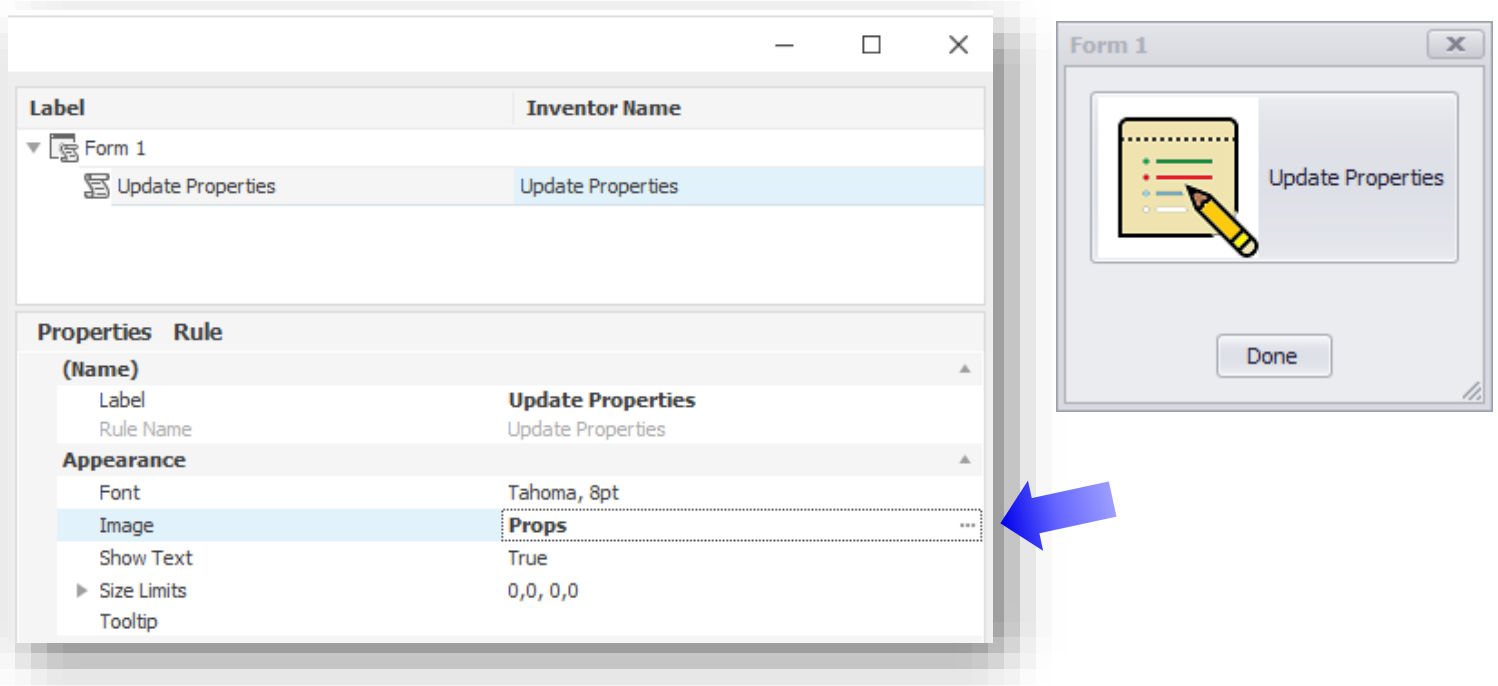
Make it more visual

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



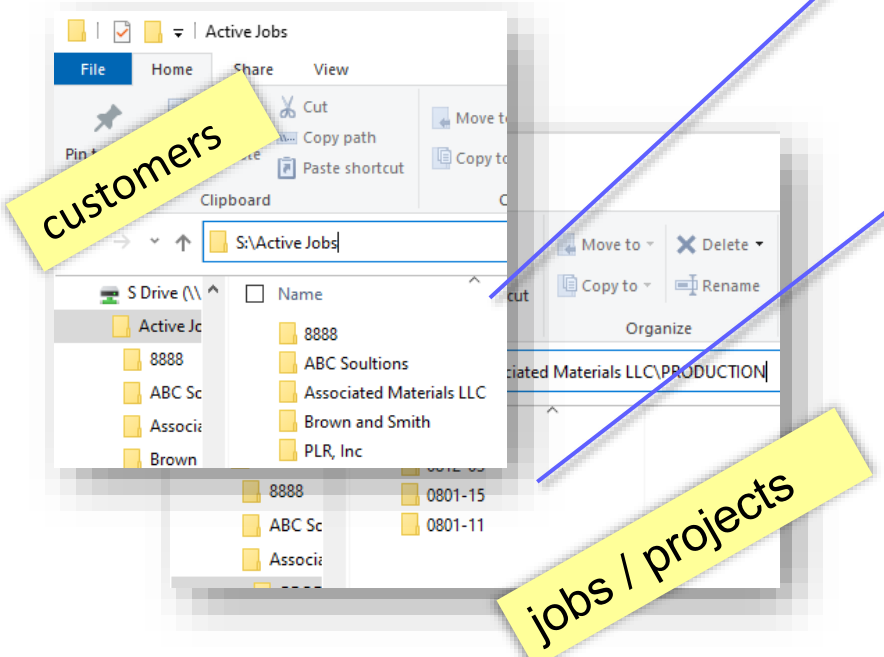


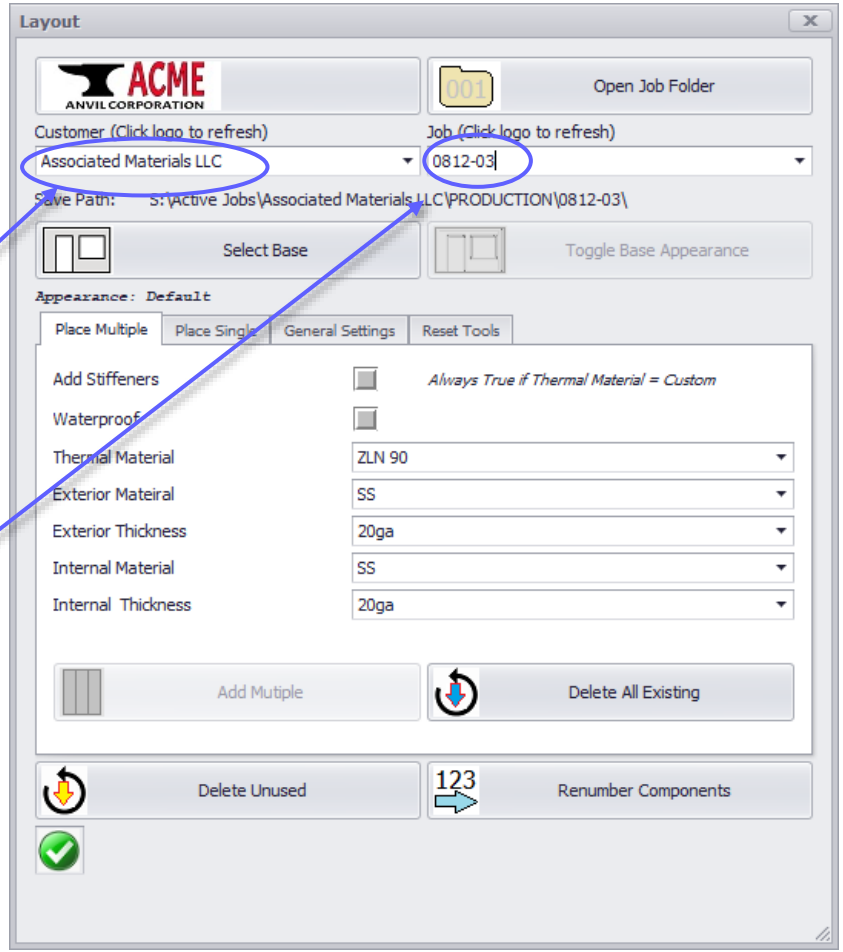
Add icons to the buttons



Make it more interactive

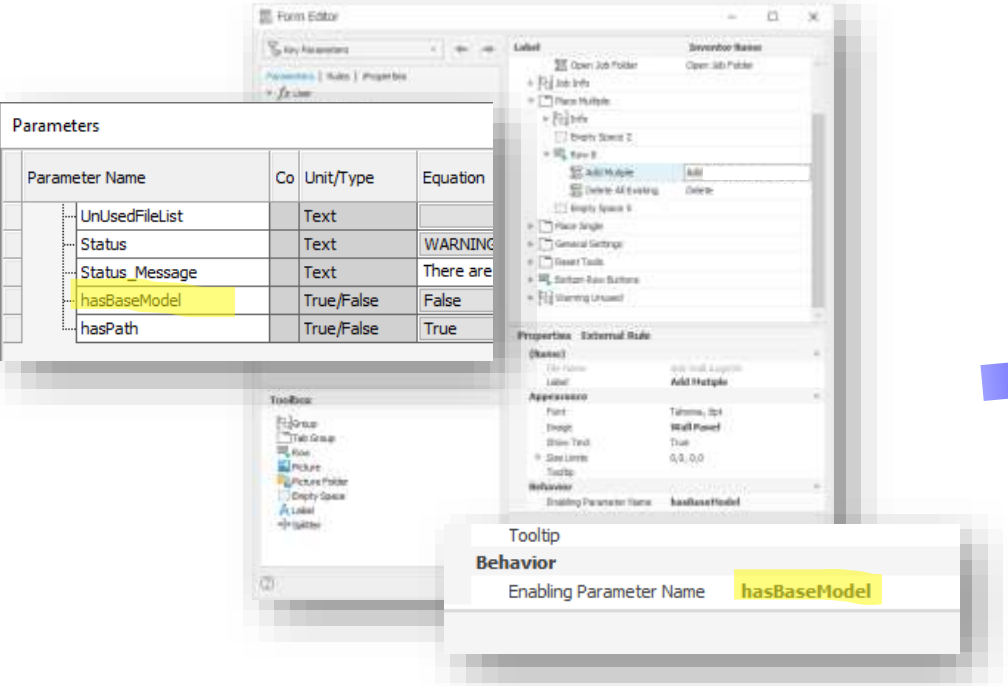
- Button for a rule that gets customer and job folders from directory, and populates the lists
- Prevents errors by disallowing invalid input

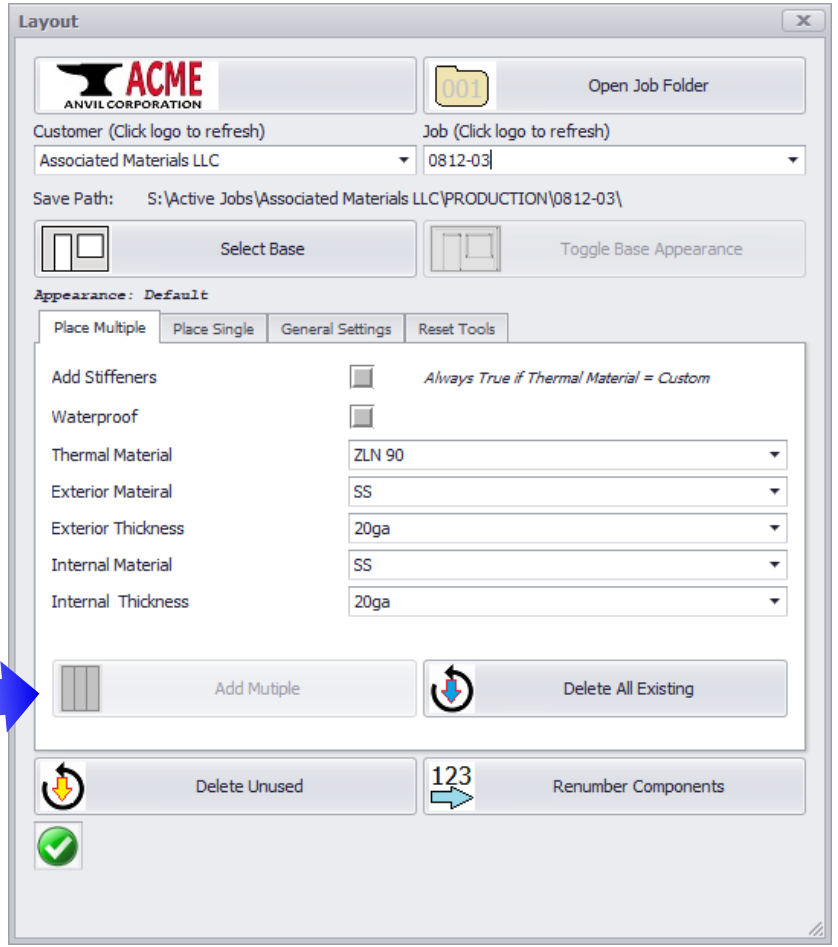




Make it more interactive

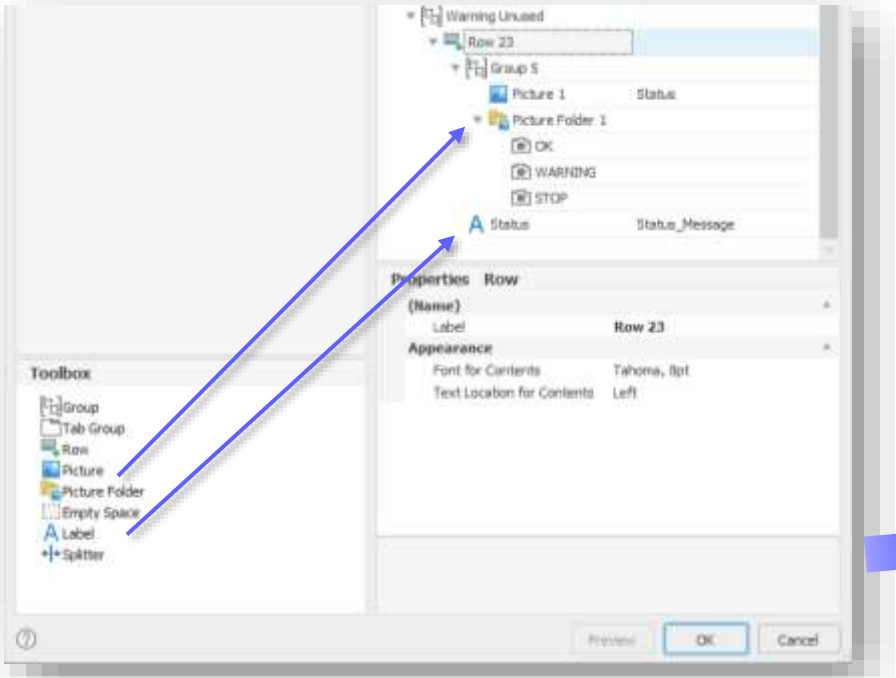
- Uses a True / False parameter as an Enabling Parameter to enable/disable the button
- Prevents errors by disallowing invalid options

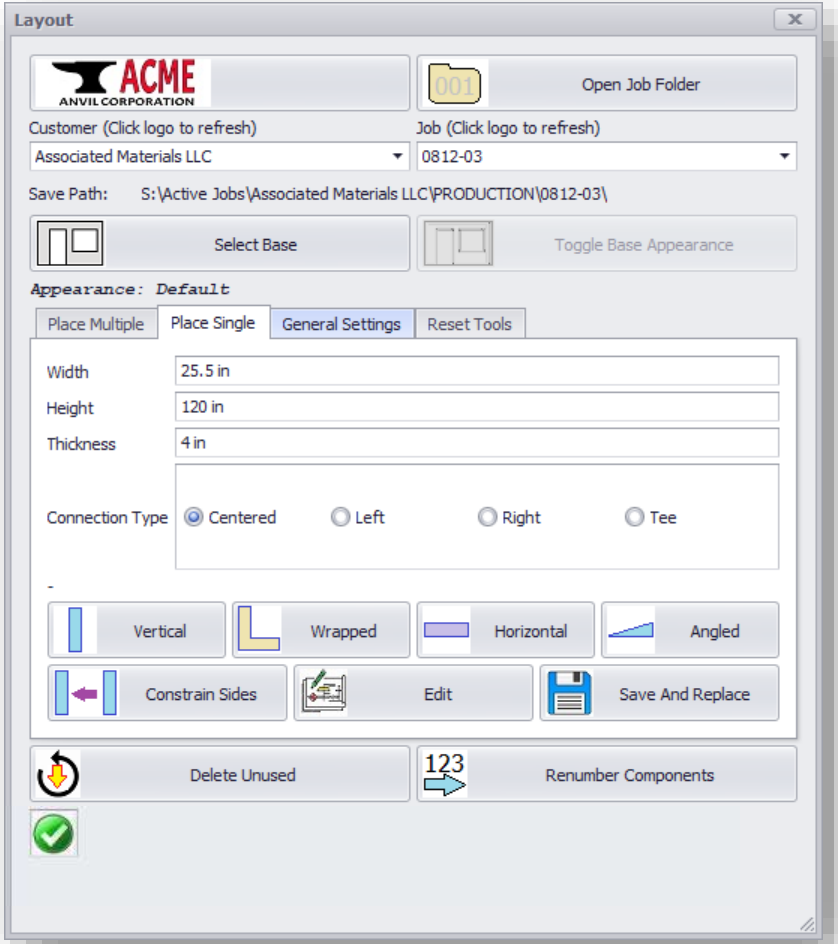




Make it more interactive

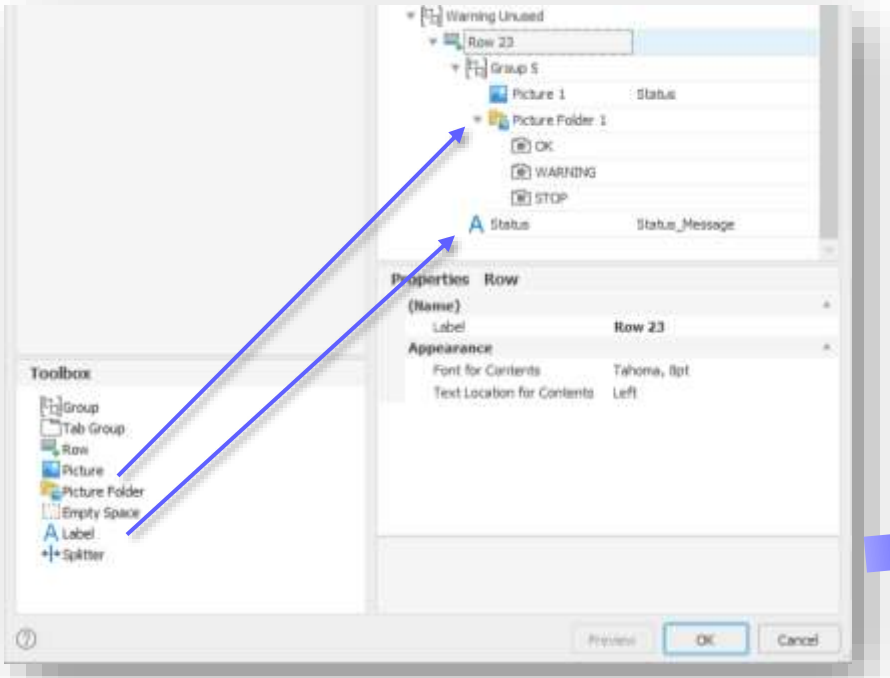
- Uses Picture Folders and labels to communicate interactively with the user

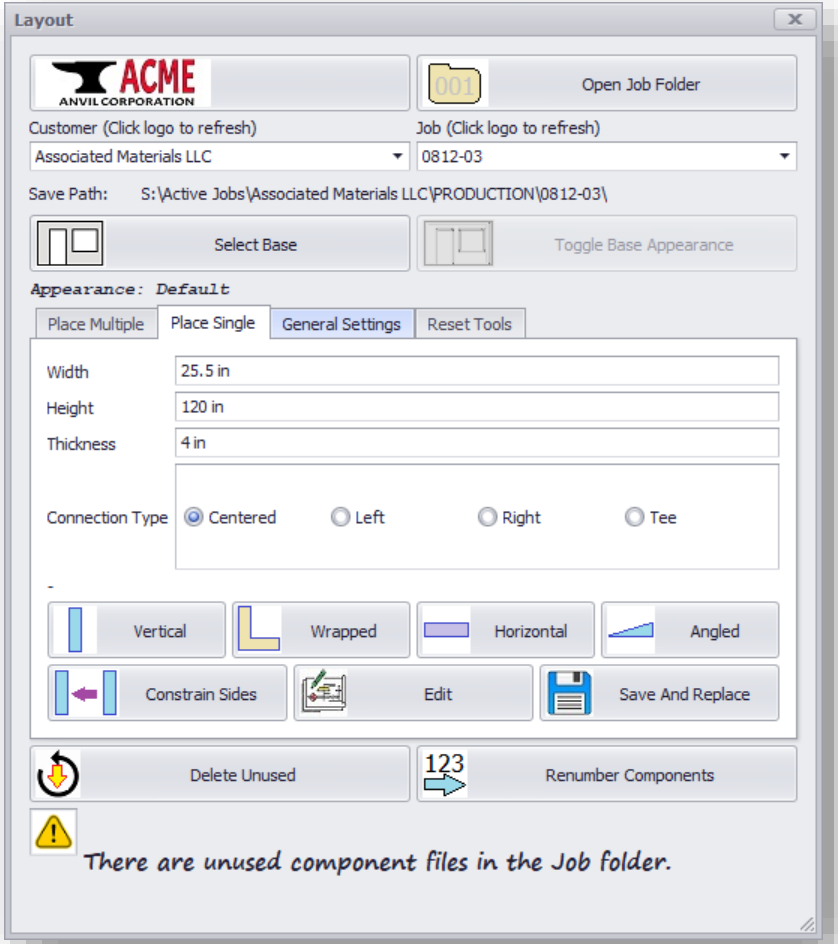




Make it more interactive

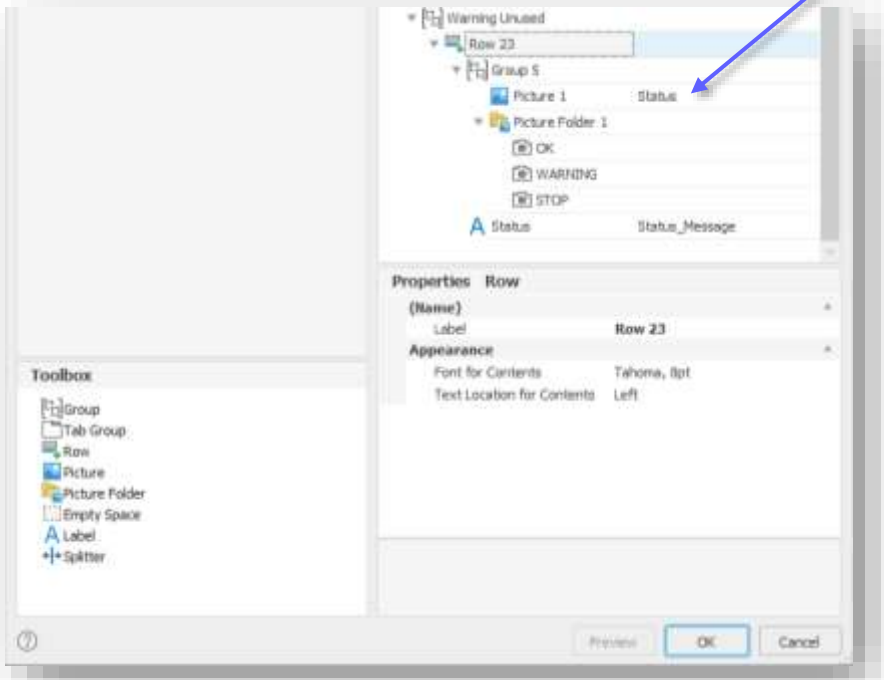
- Uses Picture Folders and labels to communicate interactively with the user

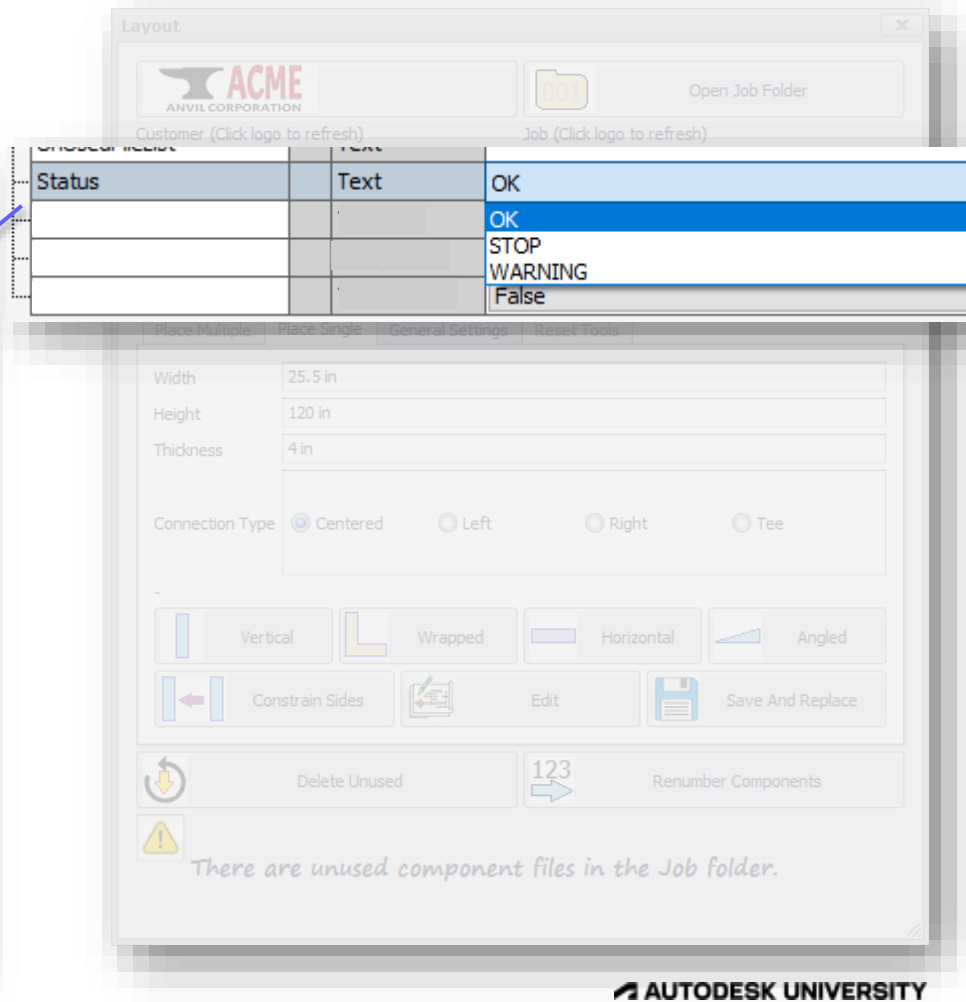




Make it more interactive

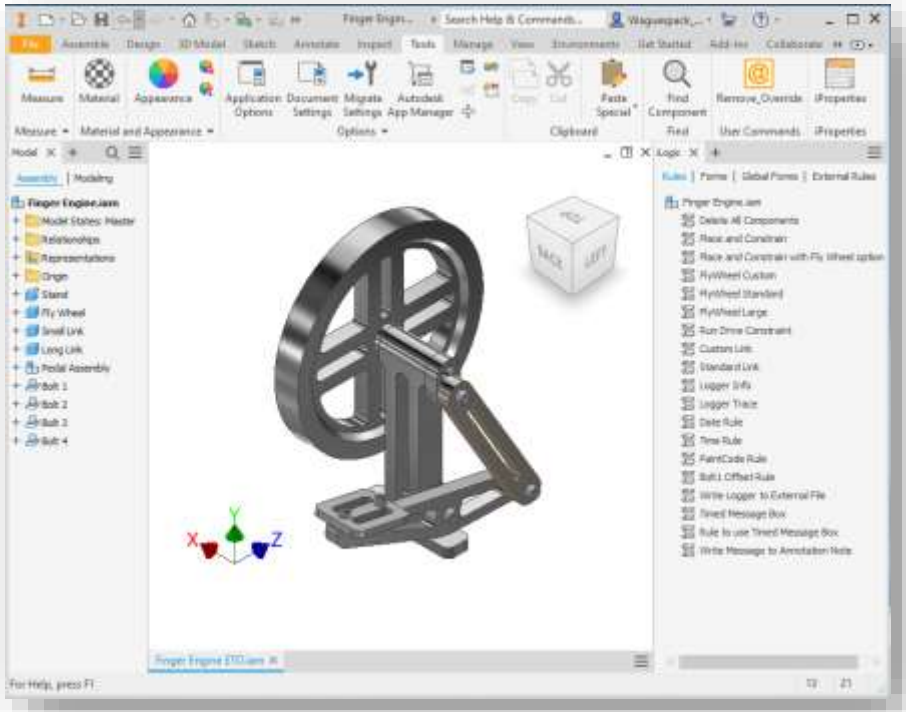
- Uses Picture Folders and labels to communicate interactively with the user






FINISH LINE IN SIGHT

Supplemental Information



MFG501293 › Code Examples

Name

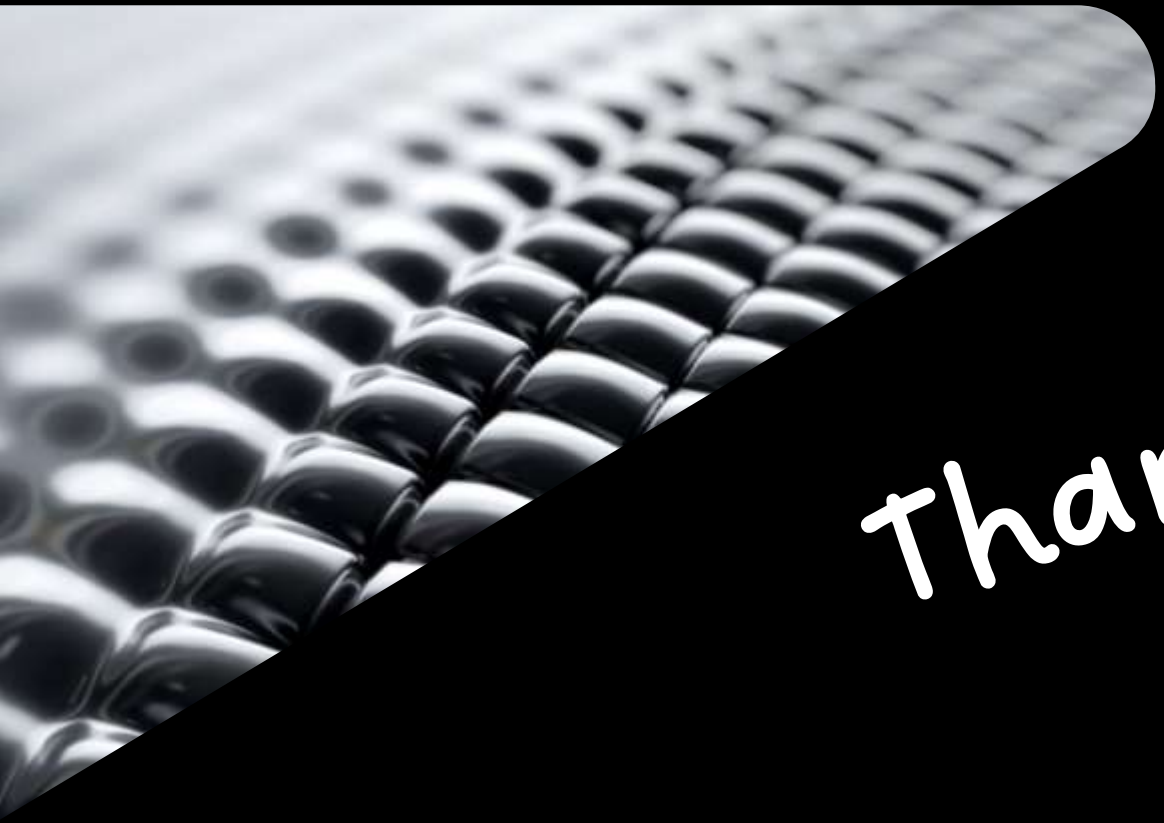
- Bin Location Class.iLogicVb
- Bin Location Function.iLogicVb
- Bin Location Sub.iLogicVb
- Bolt1 Offset Rule.iLogicVb
- Bolts Boolean Parameters with Not Version.iLogicVb
- Bolts If Then Version.iLogicVb
- Boolean Parameters with Not.iLogicVb
- Custom Link.iLogicVb
- Date Rule.iLogicVb
- Delete All Components.iLogicVb
- FlyWheel Custom.iLogicVb
- FlyWheel Large.iLogicVb
- FlyWheel Standard.iLogicVb
- Get Customer and Job Lists.iLogicVb
- Logger Info.iLogicVb
- Logger Trace.iLogicVb
- Open Jobs Folder.iLogicVb
- Outlining 1.iLogicVb
- Outlining 2.iLogicVb
- PaintCode Rule.iLogicVb
- Place and Constrain with Fly Wheel option.iLogicVb
- Place and Constrain.iLogicVb
- Planned and Optomized Rule Example.iLogicVb
- Rule to use Timed Message Box.iLogicVb
- Run Drive Constraint.iLogicVb
- Standard Link.iLogicVb
- Time Rule.iLogicVb
- Timed Message Box.iLogicVb
- Write Logger to External File.iLogicVb
- Write Message to Annotation Note.iLogicVb
- Write to Log.iLogicVb

MFG501293 › Models

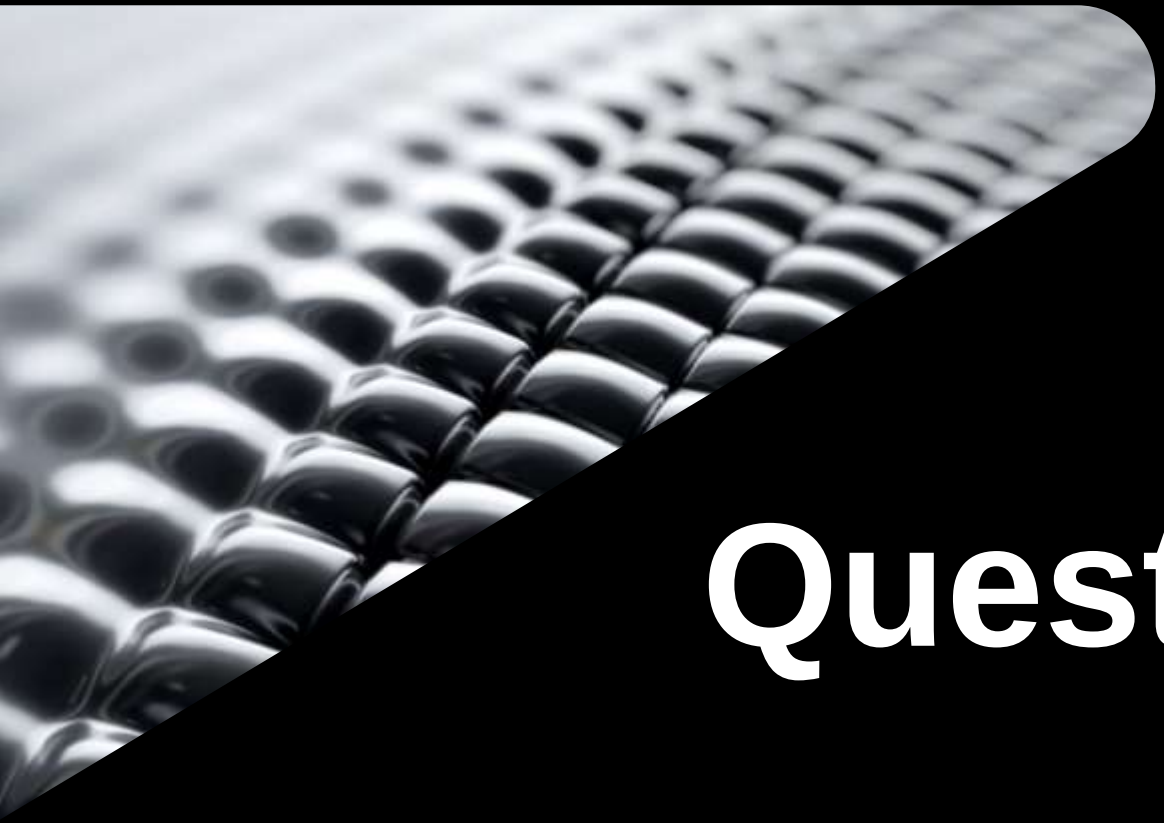
Name

- Components
- Finger Engine ETO.iam
- Finger Engine.iam

Inventor 2022 files



Thank you!



Questions?