Autodesk University 2012

iLogic for Everyone: 5 iLogic Rules Everyone Can Use

Steve Olson – MESA

CP2104

The iLogic component in Inventor is very powerful in helping users automate segments of their design process. However, there are many ways iLogic can be used, even ways that are not specific to one design. This course is intended to show students five ways they can use iLogic that are not specific to any one design. This course will start with an introduction to iLogic. Then, it will address five scenarios that the students could be facing and the iLogic Rules that could be used to resolve the issues. The class will discuss topics such as Sheet Metal Extents, Plot Stamps, iProperties, Saving as PDF, and Exporting Bills of Material.

Learning Objectives

At the end of this class, you will be able to:

- Create Inventor Templates that Utilize External iLogic Rules
- Understand the concept of External iLogic Rules and how to create them
- Manipulate properties using iLogic Rules and iLogic Forms
- Use the Save As and Export Functions inside iLogic Rules

About the Speaker

Steve Olson is the Manager of Training Services for MESA (an Autodesk Gold Partner and an Autodesk Authorized Training Center). He is a Certified Professional in Inventor and has been training students in Inventor and other Autodesk Products for four years. Steve's industry experience comes from five years of service at Fleetwood Folding Trailers. As Manager of Training Services, Steve has created five courses taught at MESA. Steve has been using iLogic since Inventor 2010, and, as an Autodesk Certified Consultant, has been working with customers in their use of iLogic for three years. Steve has presented iLogic and other Inventor-related topics for Western Pennsylvania and Northeast Ohio Inventor User Group Meetings, MESA U, and MESA-sponsored events.
steve.olson@mesa-cad.com

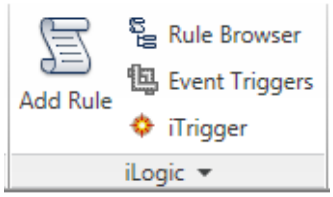
Introduction to iLogic

What is iLogic?

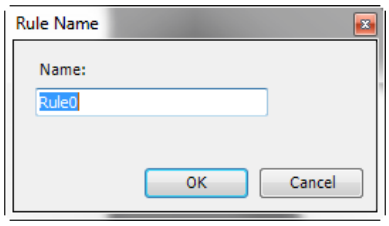
iLogic is a built-in component to Autodesk Inventor that allows users to create rules that are written in Visual Basic.Net, to automate segments of the design process. These rules are essentially small computer programs, that Inventor will run at certain times and situations. For example, a rule could be written that will cause Inventor to suppress or enable a feature based on the length of the part.

Getting Started with iLogic

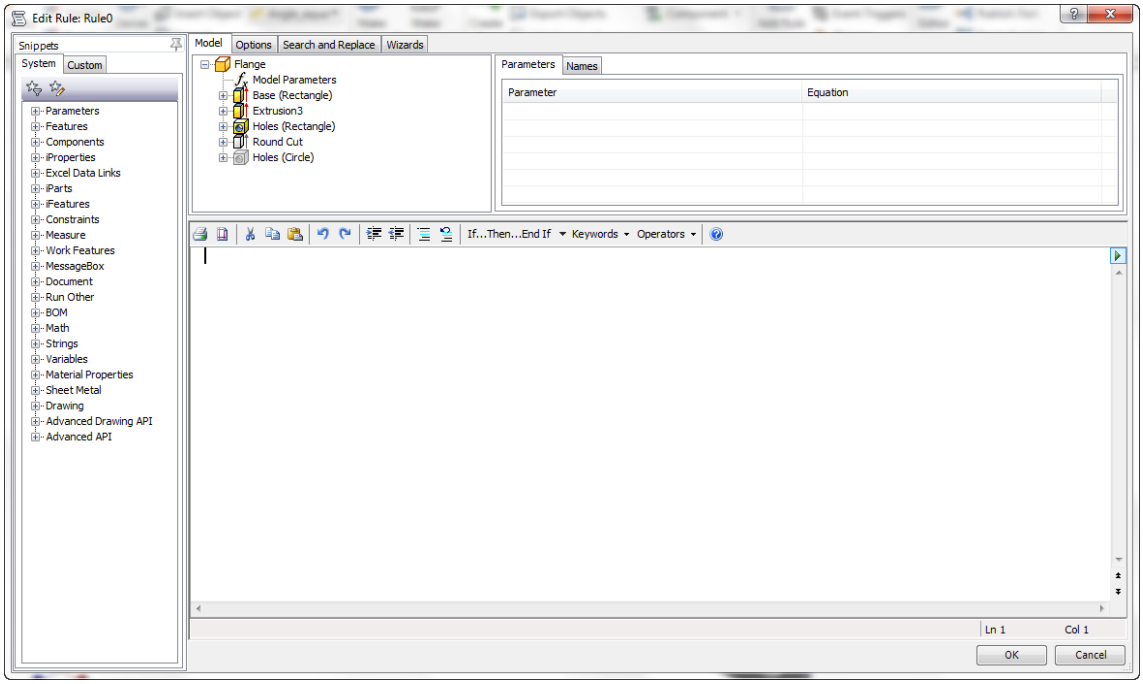
iLogic relies very heavily on two components of the 3D model. Obviously, the rules are very important (i.e. what they are and what they do). However, the rules are very dependent on the parameters of the model. In the computer programming world, programs manipulate variables. When dealing with iLogic, parameters are going be our variables, or what our rules manipulate. Since iLogic is dependent on parameters, it was necessary for Autodesk to make some changes to parameters in general. For the most part, the interface did not change. However, Autodesk has introduced two new types of parameters. Users still have the ability to create Numeric parameters. However, users now have the ability to create Text and True/False parameters. Text parameters are used for string values that the user can test and manipulate. True/False are used for Boolean operations. Along with this, Autodesk has introduced Multi-Value parameters, where the user can specify a restrictive list of values for the parameter. Before a user begins using iLogic, it would be best to start by creating the 3D model that the user has in mind. During the creation of the "base" model, the user should create the parameters that they anticipate using when the rules are written. When it comes to parameters, it is entirely possible to write rules based on the default model parameter names (i.e. d0, d1, etc) however, the best practice would be to use more descriptive names. This will help the user to keep the logic clear, and it will be easier for others to understand the intent of the rule. Once the model is created, and the user has designated the necessary parameters, it is time to begin writing the rules. The ability to add new rules can be obtained through the "Manage" tab of the ribbon.



Clicking on the "Add Rule" icon, will open a dialog box where the user is to designate the name of the rule.



After the user clicks “OK” Inventor will open the rule-authoring dialog.



This interface is also designed to make iLogic user friendly, and offer help to users that do not have any programming experience. The main pane is where the rule will be typed. The Snippets pane (left hand pane) is full of code snippets, or code templates. When the user finds a function they would like to add to the rule, a simple double click will add it with placeholders for various items the user needs to specify. For example if “IsActive” from the “Features” section of the tree is double clicked, iLogic would add `Feature.IsActive("featurename")` to the rule. In this case, `"featurename"` is a place holder where the user has to specify the name of the feature that they want to be affected by this rule. Across the top of the box, there is a pane with several tabs that give you access to the names of components, features, and parameters. Along with that, there are other tools and utilities that aid in the creation and editing of rules.

Rule #1: Sheet Metal Extents

Scenario: To create an external iLogic rule that will place the sheet metal extents of our component into the Description iProperty of the part file.

1. Start a Sheet Metal.ipt file and add the following parameters.

   Sheet Metal Extents Parameters

Name	Type	Value(s)
LengthExtents	Numeric	1 in
WidthExtents	Numeric	1 in

2. Once the parameters are created it is important to mark them for Export, and format the Custom Property so the values will be displayed properly in the description. It is also recommended that you export and format the Thickness parameter because it will be included in the description as well.
3. Create the following Sheet Metal Extents Rule as an External Rule.

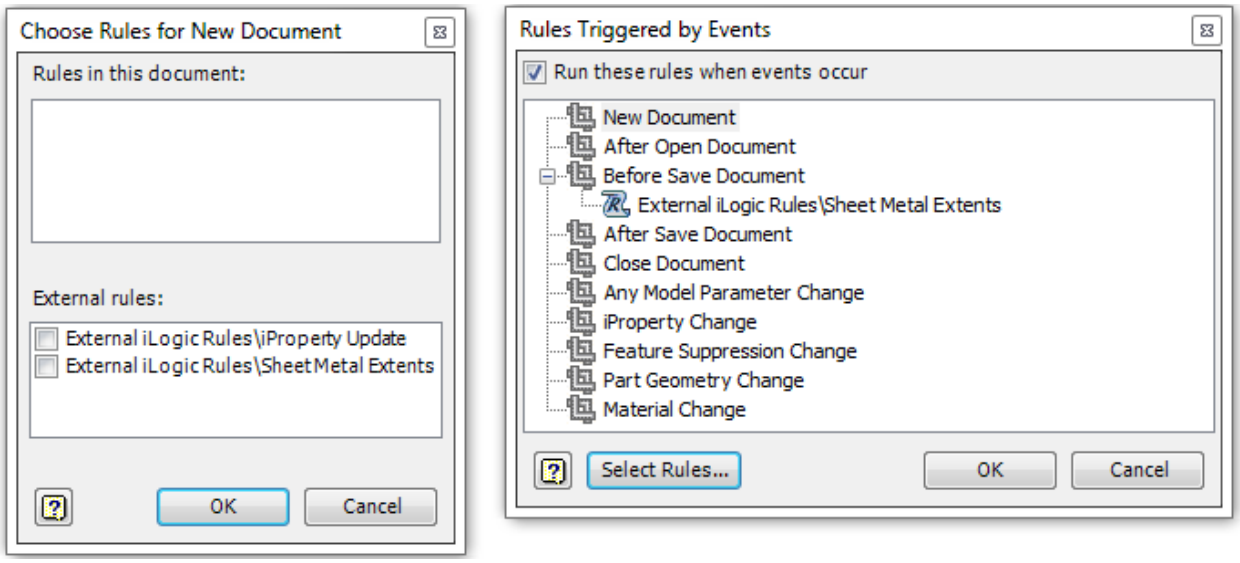
   Sheet Metal Extents Rule

```
'Sets LengthExtents and WidthExtents Parameters to the Sheet Metal
Extents
LengthExtents = SheetMetal.FlatExtentsLength
WidthExtents = SheetMetal.FlatExtentsWidth
```

   Sheet Metal Extents Rule Explanation

   These two lines take the length and width extends of the components and places their values in the LengthExtents and WidthExtents parameters.

4. Set the Event Triggers for the Sheet Metal Extents Rule. On the Manage tab of the Ribbon, select Event Triggers icon.
5. Select “Before Save Document” then the “Select Rules…” button. Check the box for “External iLogic Rules\Sheet Metal Extents” then Click OK.



6. Repeat Step 5 for the Any Model Parameter Change, iProperty Change, Freature Suppression Change, Part Geometry Change, and Material Change events.
7. In the iProperty dialog, key the following formula into the Description field “=PL <Thickness> x <WidthExtents> x <LengthExtents>.”
8. Save the file as a template. You may get errors when saving the file, it is because there is nothing to flatten and iLogic is trying to get the flat extents of an empty part file.

Now that this is set up, any file started from this template will automatically populate the Description iProperty with the flat extends. What about existing sheet metal files, we can use this external rule to perform the same function in those files, it is just a matter of making sure that the older files have the LenghtExtents and WidthExtents parameters, the parameters marked for Export and formatted properly, and the Event Triggers set. Some of this can be handled through an iLogic rule. For the others, a message box can be displayed reminding the user to perform those operations. The following Sheet Metal Legacy Utility is the rule that will perform those functions for the user.

Sheet Metal Legacy Utility

```
'Adds the LengthExtents and WidthExtents Parameters by reading an
XML file
iLogicVb.Automation.ParametersXmlLoad(ThisDoc.Document,
C:\Projects\iLogic Parts\Sheet Metal Extents-params.xml")

'Sets the Description iProperty to Pull the Custom iProperties
iProperties.Value("Project", "Description") = "=PL <Thickness> x
<WidthExtents> x <LengthExtents>"

'Displays a dialog box reminding the user to Export and format
parameters and set Event Triggers
```

```
i = MessageBox.Show("The Extents Parameters have been added.  Please
mark them for export and format them appropriately.  Then set the
triggers for the Sheet Metal Extents Rule.", "Sheet Metal Legacy
Utility", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1)
```

Sheet Metal Legacy Utility Explanation

The first line adds the LengthExtents and WidthExtents Parameters by reading in an XML file. The second line sets the part description. Inventor will pull the iProperty value when it sees the <> symbols around the iProperty name. In this case, Thickness, WidthExtents, and Length Extents. Then the rule displays a dialog that reminds the user to Export and Format the parameters, as well as, setting the Event Triggers for the Sheet Metal Extents Rule in this file. Recommended Event Triggers for this rule include: Before Save Document, Any Model Parameter Change, iProperty Change, Feature Suppression Change, Part Geometry Change, and Material Change.

Rule #2: IDW Plot Stamp

Scenario: To use iLogic to generate a Plot Stamp on an IDW file, and clear it off once plotting is done.

Plotting with a Plot Stamp is an interesting rule because iLogic is really good at manipulating properties and geometry, but it is not really suited for creating or placing items. So the real trick here is how to get a plot stamp on a drawing in the first place. This can be done by having a border that contains the properties that need to be shown in the plot stamp. When the properties are blank, it will appear as if we removed the plot stamp. Which works great for any file started from a template that includes that border, but what about older files? iLogic is capable of reading drawing resources from a specified file. Therefore, any old files can be prompted to read a border containing the plot stamp properties from a drawing template.

It is worth noting that for this rule to work, a drawing template with a border containing the plot stamp properties needs to be created. In the following code, it will be necessary to change the line declaring the template to match the path and file name of the template being used in your environment.

Plot With Plot Stamp

```
'This sets the Custom iProperties to the appropriate values
iProperties.Value("Custom", "PS_Name") =
ThisApplication.GeneralOptions.UserName
iProperties.Value("Custom", "PS_Date") = Now.ToShortDateString
iProperties.Value("Custom", "PS_Time") = Now.ToShortTimeString

'This declares a template to pull drawing resources from
ThisDrawing.ResourceFileName = "C:\CP2104 iLogic For Everyone\Inventor
Support\Templates\iLogic Templates\IDW with Plot Stamp.idw"

'This is a loop that replaces the border on all sheets
NoSheets = ThisApplication.ActiveDocument.Sheets.Count

For i = 1 To NoSheets

ThisSheet = "Sheet:" & i

ActiveSheet = ThisDrawing.Sheet(ThisSheet)
ActiveSheet.Border = "PS Border"

Next i

InventorVb.DocumentUpdate()

'Printing Commands
Dim oCtrlDef As ControlDefinition
```

```
oCtrlDef =
ThisApplication.CommandManager.ControlDefinitions.Item("AppFilePrintCmd
")
oCtrlDef.Execute

'Clears Plot Stamp Values
iProperties.Value("Custom", "PS_Name") = " "
iProperties.Value("Custom", "PS_Date") = " "
iProperties.Value("Custom", "PS_Time") = " "

'Updates the View
iLogicVb.UpdateWhenDone = True
```

Explanation of Plot with Plot Stamp Rule

The first three statements create the PS_Name, PS_Date, and PS_Time Custom iProperties, and sets them to the appropriate information pulled from the system. The statement in the next section identifies that the rule wants to pull drawing resources from a particular file. This rule will direct iLogic to pull a border from the named file. The next group of statements, finds the total number of sheets in the file, and creates a loop that will cause the border on each sheet in the drawing to be changed to the border specified. The statements listed after 'Printing Commands, cause the Printing Dialog to be initiated. The user is able to set the printing options to match their needs. The last four statements, blank out the values of the Custom iProperties, and cause the document to update.

It is important to note that the previous rule will work for Inventor 2012 and 2013. Starting with Inventor 2012, if a rule refers to a Custom iProperty that does not exist in the file, iLogic will create the Custom iProperty. Inventor 2011 and prior releases would error on any statement referring to non-existent Custom iProperties. Therefore, you may need to use the following rule to get add the necessary Custom iProperties.

Add Custom iProperites (Inventor 2011 and previous)

```
Sub Main()
  FindOrCreateProperty("PS_Name", "Username")
  FindOrCreateProperty("PS_Date", "1/1/2011")
  FindOrCreateProperty("PS_Time", "12:00 AM")
  'FindOrCreateProperty("another new prop", 0.0)
End Sub

Function FindOrCreateProperty(propName As String, propValue As Object)
     ' Get the current document.
     oDoc = ThisDoc.Document
    ' Get the user defined (custom) property set
    Dim propSet As PropertySet = oDoc.PropertySets.Item("Inventor User
Defined Properties")
    Dim invProperty As Inventor.Property
```

```
    ' See if the property alreay exists.  If it does, don't change the
value.
    For Each invProperty In propSet
      If (invProperty.Name = propName) Then Return invProperty
    Next
    ' Create the property
    invProperty = propSet.Add(propValue, propName)
End Function
```

Add Custom iProperties Explanation

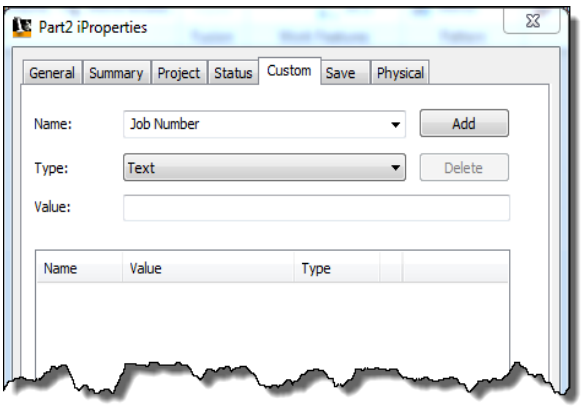
The first portion of this rule lists all the properties that the user wants to create in this rule. Note, there is a template line left in there so the user can see the syntax that needs to be used. The second section is a loop that will check to see if the iProperty exists, and if it doesn't the iProperty will be created. This rule was found during a Google search and is written in Straight VB Code and this rule would need to be identified as such.

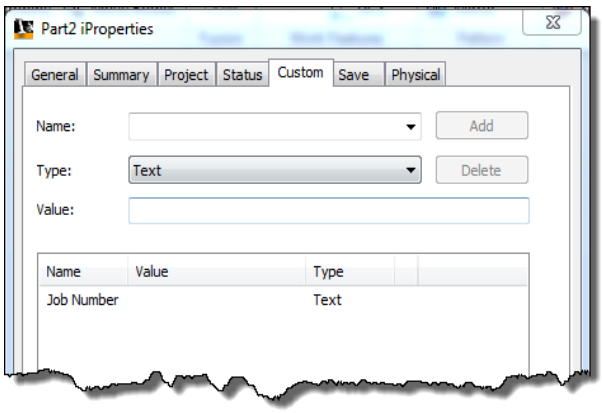
Rule #3: iProperty Form

Scenario: To create a Global Form that will give us access to the iProperties that are the most important to us. We will also assume that there are some legacy files that may not have certain Custom iProperties, so we will create a rule that will add those iProperties to older files, and add that rule to our Global Form.

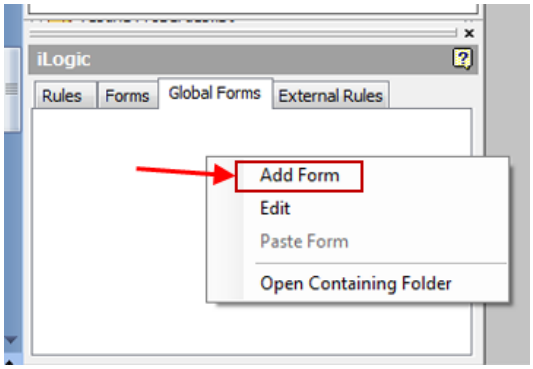
Creating the Form

1. Start an Inventor part file.
2. For this example, we will create two Custom iProperties (Job Number and Customer). Go to the iProperites dialog, and select the "Custom" tab of the dialog.
3. Create the Job Number Custom iProperty by typing Job Number in the Name Field, do not worry about the Value field. Then click the "Add" button.





4. Repeat Step 3 to create a Customer Custom iProperty.
5. Close the iProperties Dialog.
6. In the iLogic Browser, click on the "Global Forms" tab, then right click and choose "Add Form."

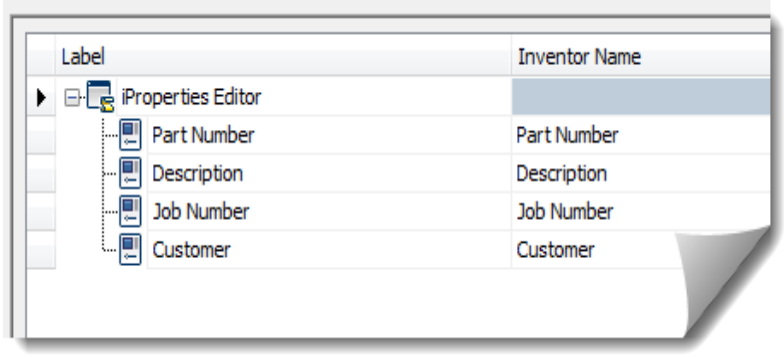


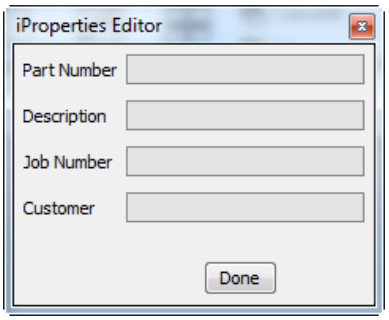
This will open the Form Editor and a preview of the form.

7. In the Form Editor, under Label change the name of the form from "Form 1" to "iProperties Editor."

8. In the browser on the left side of the Form Editor dialog, click on the iProperties tab. Click and drag the Part Number iProperty from the browser to the main window of the dialog. This will add the Part Number iProperty to the form.
9. Repeat Step 9 to add the Description, Job Number and Customer iProperties to the form. (Note: The Customer and Job Number Custom iProperties will be listed under the Custom branch of the browser.)

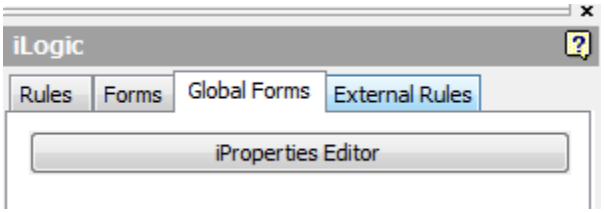
The Form Editor and Form Preview should look like the following images.



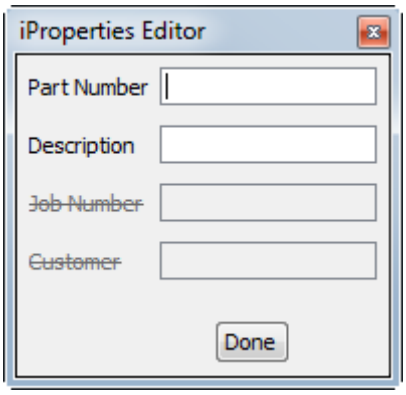


10. Click “OK” in the Form Editor dialog to close the dialog.

Now there should be an iProperties Editor button in the Global Forms tab of the iLogic Browser.



Clicking on the iProperties Editor button will display the form and allow the user to edit those properties. Since the form is a Global Form it will be available in all files. However, if the form is accessed from a file that does not contain the Job Number or Customer iProperites, those fields will be grayed out when the form is accessed (see the following image).

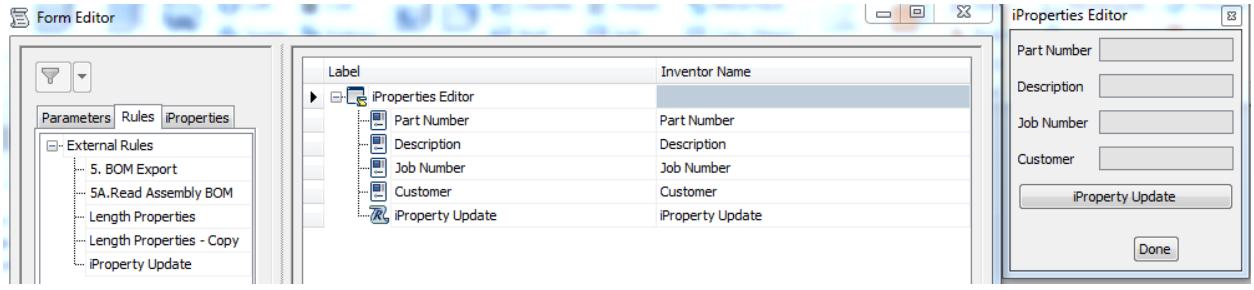


This can be remedied by creating an External Rule that will add those properties to the active file. A button can be added to the form that will run the rule.

iProperty Update Rule

```
iProperties.Value("Custom", "Job Number") = ""
iProperties.Value("Custom", "Customer") = ""
```

Once the rule is created, go to the Global Forms tab, and right click on the iProperties Editor button. Then choose Edit. This will open the Form Editor dialog, and allow the user to add elements to the form. In the browser, click the Rules tab, and drag and drop the iProperty Update rule into the main window of the dialog. The Form Editor and preview should look like the following image.



Now if the user accesses the form from a file that does not contain those properties, clicking the iProperties Update button will run the rule, and allow the user to edit the Job Number and Customer Custom iProperties.

It is important to note that if you have a form similar to this in Inventor 2012, the form would have to be closed and reopened for the new iProperties to be editable.

A lot of CAD Managers want to make sure their users are filling in certain iProperties. iLogic can also be used to test if certain iProperties have been filled in. An External iLogic rule can be created that test the length of the string of each mandatory iProperty. The rule can then set to run when the file is saved, and if iProperties are not filled in, the rule can then inform the user

with a verbal cue, display an error message, and then display the form so the user can fill in the iProperties.

iProperty Check Rule

```
EmptyProperties = 0

If Len(iProperties.Value("Project", "Part Number")) = 0 Then
    EmptyProperties = EmptyProperties + 1
End If

If Len(iProperties.Value("Project", "Description")) = 0 Then
    EmptyProperties = EmptyProperties + 1
End If

If Len(iProperties.Value("Custom", "Job Number")) = 0 Then
    EmptyProperties = EmptyProperties + 1
End If

If Len(iProperties.Value("Custom", "Customer")) = 0 Then
    EmptyProperties = EmptyProperties + 1
End If

If EmptyProperties > 0 Then
    '___Use windows voice command_____________
    Dim objSPVoice,colVoices
    objSPVoice = CreateObject("SAPI.SpVoice")
    objSPVoice.Speak ("Not all i Properties have been filled out")

    i = MessageBox.Show("Not all iProperties have been filled out.
Please enter the appropriate information.", "iProperties Missing",
MessageBoxButtons.OK, MessageBoxIcon.Hand,
MessageBoxDefaultButton.Button1)

    iLogicForm.ShowGlobal("iProperties Editor")
End If
```

Rule #4: Save Copy As PDF

Scenario: To create a rule that will save a PDF copy of our drawing to a directory somewhere on our network.

Create PDF

```
'Gets the Workspace Path
WorkspacePath= ThisDoc.WorkspacePath()

'Gets the Length of the WorkspacePath String
WorkspacePathLength = Len(WorkspacePath)

'Gets just the Path of the file
PathOnly = ThisDoc.Path

'Removes the Workspace Path from FullPath
DirectoryPath = Strings.Right(PathOnly, PathOnly.Length-
WorkspacePathLength)

'Sets the Dirctory that the PDF should be saved in
PDFPath = "C:\PDFs\"  & DirectoryPath

'Checks to see if that directory exists, if not, it is created
If(Not System.IO.Directory.Exists(PDFPath)) Then
    System.IO.Directory.CreateDirectory(PDFPath)
End If

'Saves the PDF in the desired location
ThisDoc.Document.SaveAs(PDFPath & "\" & ThisDoc.FileName(False) &
".pdf" , True)
```

Create PDF Rule Explanation

The first three lines determine the workspace path, the length of the workspace path, and the path of the document being worked on. The next line removes the workspace portion from the path of the file. The next line puts a “C:\PDFs\” prefix on the path of the file. The rule then checks to see if the directory that the file will be saved in exists. If the path does not exist, then the directory is created. The last statement saves a PDF version of the file.

Advanced Create PDF

```
Sub Main ()
FileName = ThisDoc.FileName(True) 'with extension

FileExtension = Right(FileName, 3)

If FileExtension = "idw" Then
    Save_As_PDF
Else If FileExtension = "dwg" Then
```

```
    Save_As_PDF
Else
    ErrorMessage
End If
End Sub

Sub Save_As_PDF
'Gets the Workspace Path
WorkspacePath= ThisDoc.WorkspacePath()

'Gets the Length of the WorkspacePath String
WorkspacePathLength = Len(WorkspacePath)

'Gets just the Path of the file
PathOnly = ThisDoc.Path

'Removes the Workspace Path from FullPath
DirectoryPath = Strings.Right(PathOnly, PathOnly.Length-
WorkspacePathLength)

'Sets the Dirctory that the PDF should be saved in
PDFPath = "C:\PDFs\"  & DirectoryPath

'Checks to see if that directory exists, if not, it is created
If(Not System.IO.Directory.Exists(PDFPath)) Then
    System.IO.Directory.CreateDirectory(PDFPath)
End If

'Saves the PDF in the desired location
ThisDoc.Document.SaveAs(PDFPath & "\" & ThisDoc.FileName(False) &
(".pdf") , True)
End Sub

Sub ErrorMessage
i = MessageBox.Show("This is not a drawing file.  No PDF will be
created.", "Create PDF", MessageBoxButtons.OK, MessageBoxIcon.Hand,
MessageBoxDefaultButton.Button1)
End Sub
```

Rule #5: Export Bill of Material

Scenario: To create an iLogic Rule that will export the Bill of Material of an assembly as an Excel file, then have a drawing file read certain information from the exported Bill of Material.

BOM Export

```
'Get the Filename of the file without the extension
FileName = ThisDoc.FileName(False) 'without extension

'Sets the path and filename of where the Exported BOM should be
saved
BOMFile = "C:\BOMs\" & FileName & ".xls"

'Exports the BOM to the desired location
ThisBOM.Export("Structured", BOMFile, kMicrosoftExcelFormat)
'kMicrosoftAccessFormat            = Microsoft Access
'kMicrosoftExcelFormat            = Microsoft Excel
'kdBASEIIIFormat                  = dBASE III
'kdBASEIVFormat                   = dBASE IV
'kTextFileTabDelimitedFormat      = Text File Tab Delimited
'kTextFileCommaDelimitedFormat    = Text File Comma Delimited
'kUnicodeTextFileTabDelimitedFormat    = Unicode Text File Tab
Delimited
'kUnicodeTextFileCommaDelimitedFormat    = Unicode Text File Comma
Delimited
```

BOM Export Explanation

The first line retrieves the name of the file without the extension. The second line creates a path and filename string for the Exported BOM. The last statement exports the BOM to the desired location.

The next rule is designed to read an existing Excel spreadsheet and populate Custom iProperites that will fill in information in a sketch symbol. The sketch symbol's purpose is to display the item number and quantity of a part in reference to the Bill of Material of a specified assembly.

Read Assembly BOM

```
'Gets name of reference assembly from iProperties and identifies the
spreadsheet to be read
BOMFile = "C:\BOMs\" & iProperties.Value("Custom", "Assembly Name")
& ".xls"

'Gets the name of the model represented in the oldest view on the
drawing
modelName =
IO.Path.GetFileName(ThisDrawing.ModelDocument.FullFileName)
```

```
modelName = Strings.Left(modelName, modelName.Length - 4)

'Reads information from the specified spreadsheet and places values
in properties
i = GoExcel.FindRow(BOMFile, "Sheet1", "Part Number", "=",
modelName)

iProperties.Value("Custom", "Item") =
GoExcel.CurrentRowValue("Item")

iProperties.Value("Custom", "QTY") = GoExcel.CurrentRowValue("QTY")

'Updates the view
iLogicVb.UpdateWhenDone = True
```

Read Assembly BOM Explanation

The first line identifies the path to the Excel file to be read by combining the contents of a Custom iProperty with “C:\BOMs\” and “.xls.” The second and third statements get the file name, without the extension, of the part represented in the oldest view on the drawing. The last group of statements read the Excel file and retrieves information regarding the part referenced on the drawing, specifically Item Number and Quantity.