IM500024

Discover the flexibility of iLogic in Inventor to improve your processes

Pascale Brassat (Dipl.-Ing.(FH)
Claudius Peters Projects GmbH

Lernziele

- identify daily tasks if they could be improved with iLogic
- solve simple problems with iLogic
- search for solutions based on iLogic to improve your daily work
- to combine a lot of different iLogic based ideas to optimize more complex tasks step by step

Description

How many times you're working in Inventor and asking yourself, if it's possible to improve your daily work by automate repeating tasks or organize complicated jobs. iLogic is a very powerful and flexible programming technique in Inventor and could improve most of your daily processes. I want to show with a couple of examples how many different processes could be improved by using this technology. e.g. building up a library of parts, using iLogic forms, store knowledge into iLogic, using vb.net knowledge inside of iLogic… In our company at Claudius Peters we're reducing a lot of time consuming tasks to a minimum by using iLogic. Also we're storing knowledge and rules into iLogic programs to improve our internal process quality.

Speaker

Very short summary.
Working at Claudius Peters Projects GmbH.
Since 2007 I'm using Inventor and since version 2010 I'm using iLogic. Technical draftsman and afterwards university degree in mechanical engineering and informatics.

General Information

iLogic within Inventor enables the automation of many processes. I would like to briefly show how diverse the possibilities are so that the reader can develop their own ideas.
Briefly in advance. Except for a few restrictions, because iLogic does not offer access to the product areas within Inventor (frame generator, etc.), there are hardly any limits to your own ideas.

Before even considering using iLogic. What is that, actually?

iLogic is a fully integrated programming environment with access to almost all objects within Inventor. It is somewhat comparable to VBA, only more convenient in terms of access to these objects.

Compared to VBA, iLogic is a little restricted when it comes to the design of forms, but this can also be managed if required by using external programs (e.g. using .net in combination with iLogic).

iLogic is already on board when installing Inventor and has been since version 2011. (In version 2010 it can still be installed using an add-in)

Since then, the editor has been expanded again and again and iLogic has been optimized a bit overall, but the programs from 2011 can still be run today as they did then.

The programming environment is very simple and clear and therefore offers a good environment for the first steps, especially for beginners. Even those switching from VBA will find themselves at home right away. Especially since iLogic also speaks a VB dialect and therefore the programming looks almost identical in terms of language.

ILogic can be reached via the ribbon: Manage / iLogic / iLogic Browser as soon as a template or file is opened.

Which topics should be considered for a solution via iLogic?

Of course, this question is not so easy to answer, since there are as many different tasks as there are sand by the sea, but here are 3 exemplary ideas in which areas tasks can be considered:

1. Tasks that are often recurring and that can be automated. Assuming you save 5 out of 6 minutes of work per run on a task and this run has to be repeated very often per day, then that would be a perfect exercise for iLogic.

Note: It is mostly not the big programs that increase satisfaction or speed up the processes, but rather the small things that often eat up time. Addressing these issues ultimately frees up the time that is needed to complete large-scale tasks.

2. Libraries can be created wonderfully with iLogic. Configurable components, for example. All features, parameters, etc. can be controlled via iLogic. Also across assemblies and

directly by selection from a browser. It is therefore very easy to display the most diverse forms, including adapting the parts lists, etc.

3. The third category concerns the safeguarding of know-how within a company. In iLogic, things can be implemented very well that only certain users or employees know. This can be about technical regulations as well as the processing of certain processes. Suppose you only work on a process once every few weeks, but in order for this process to be completed, a manual has to be retrieved every time and this process takes so much longer than actually necessary. Then this process could be implemented in iLogic. Or certain rules that technically specify how a component should look (e.g. if a certain parameter has a certain value, another parameter cannot be greater than ...).

Information:

First of all, an important piece of information:
There are thousands of problems that need to be solved and that can be solved 😊 That's why I say: take courage. For beginners in particular, the following applies: Do not be put off by discussions about programming styles etc. It is important to have the confidence and get started.

Small Examples:

Create 5 views based on a main view

Prerequisite: A drawing file (idw, dwg) in Inventor is open and a view is placed (ideally the front view)

Then create within the document (if the rule is only to be used within the document) or as an external rule (if the rule is to be used across documents and you do not want to copy the rule every time), a rule with the name like him they like it (e.g. CreateDrawingViews)

Then paste the content below there.

If the rule is executed, you as the user will be asked to select a view. If you have done that, then everything around the selected view will automatically be displayed
additional basic views created. (Projection method 1)

An underscore at the end always means a line break within iLogic.
In part, this makes legibility a little more difficult, but it is necessary to use the space accordingly ...

```
If ThisApplication.ActiveDocument.DocumentType = _
        DocumentTypeEnum.kDrawingDocumentObject Then

Dim oDrwg As DrawingDocument
oDrwg = ThisApplication.ActiveDocument

Dim oView As DrawingView

oView = _
        ThisApplication.CommandManager.Pick(SelectionFilterEnum.kDrawingViewFilter, _
```

Only start the procedure if it is a Drawing Document

```
        "Please select a drawing view")

If oView Is Nothing Then
        Exit Sub 'In case of "esc is hit during the pick command"
End If

Dim oAction = ThisApplication.TransactionManager.StartTransaction(oDrwg,
        "Create 6 Views")

Dim oTG As TransientGeometry
oTG = ThisApplication.TransientGeometry

Dim Point1 As Point2d = oTG.CreatePoint2d(oView.Position.X - 50, oView.Position.Y)

Dim RightView As DrawingView = oDrwg.ActiveSheet.DrawingViews.AddProjectedView(oView,
_
                Point1, DrawingViewStyleEnum.kFromBaseDrawingViewStyle, oView.Scale)

Point1 = _
oTG.CreatePoint2d(oView.Position.X - (oView.Width / 2) -(RightView.Width / 2) -3, _
        oView.Position.Y)
        RightView.Position = Point1

Dim Point2 As Point2d = oTG.CreatePoint2d(oView.Position.X + 50, oView.Position.Y)

Dim LeftView As DrawingView = _
        oDrwg.ActiveSheet.DrawingViews.AddProjectedView(oView, Point2, _
        DrawingViewStyleEnum.kFromBaseDrawingViewStyle, oView.Scale)

Point2 = _
oTG.CreatePoint2d(oView.Position.X + (oView.Width / 2) + (LeftView.Width / 2) + 3, _
        oView.Position.Y)
        LeftView.Position = Point2

Dim Point3 As Point2d = oTG.CreatePoint2d(oView.Position.X, oView.Position.Y - 50)

Dim TopView As DrawingView = _
        oDrwg.ActiveSheet.DrawingViews.AddProjectedView(oView, Point3, _
        DrawingViewStyleEnum.kFromBaseDrawingViewStyle, oView.Scale)

Point3 = oTG.CreatePoint2d(oView.Position.X, _
        oView.Position.Y - (oView.Height / 2) -(TopView.Height / 2) -3)

TopView.Position = Point3

Dim Point4 As Point2d = oTG.CreatePoint2d(oView.Position.X, oView.Position.Y + 50)

Dim BottomView As DrawingView = _
        oDrwg.ActiveSheet.DrawingViews.AddProjectedView(oView, Point4, _
        DrawingViewStyleEnum.kFromBaseDrawingViewStyle, oView.Scale)

Point4 = oTG.CreatePoint2d(oView.Position.X, _
        oView.Position.Y + (oView.Height / 2) + (BottomView.Height / 2) + 3)

BottomView.Position = Point4

Dim Point5 As Point2d = oTG.CreatePoint2d(LeftView.Position.X + 50, _
        LeftView.Position.Y)
```

Query to select a specific view.

Transaction? For using Undo Command on the action

Positioning and creating a new view

```
Dim BackView As DrawingView = _
        oDrwg.ActiveSheet.DrawingViews.AddProjectedView(LeftView, _
        Point5, DrawingViewStyleEnum.kFromBaseDrawingViewStyle, oView.Scale)

Point5 = _
        oTG.CreatePoint2d(LeftView.Position.X + _
        (LeftView.Width / 2) + (BackView.Width / 2) + 3, _
        LeftView.Position.Y)

BackView.Position = Point5

oAction.End

End If
```

Combination of several solids / solid bodies into a single one:

```
If ThisDoc.Document.DocumentType = DocumentTypeEnum.kPartDocumentObject Then

Dim Teil As PartDocument = ThisDoc.Document

If Teil.ComponentDefinition.SurfaceBodies.Count   > 1 Then

        Dim Aktion As Transaction = _
        ThisApplication.TransactionManager.StartTransaction(Teil,"Combine
All")

        Dim oToolBodies As ObjectCollection = _
        ThisApplication.TransientObjects.CreateObjectCollection

        Dim i As Integer = 1

        For Each Element As SurfaceBody In _
                Teil.ComponentDefinition.SurfaceBodies

                If i > 1 Then
                        oToolBodies.Add(Element)
                End If
                i = i+1
        Next

Dim oCombine As CombineFeature

oCombine = _
        Teil.ComponentDefinition.Features.CombineFeatures.Add( _
        Teil.ComponentDefinition.SurfaceBodies.Item(1), _
        oToolBodies, PartFeatureOperationEnum.kJoinOperation, False)

        Aktion.End

        End If

End If
```

Cycling through all solids in a component.

Adding the file name of a selected component to the clipboard:

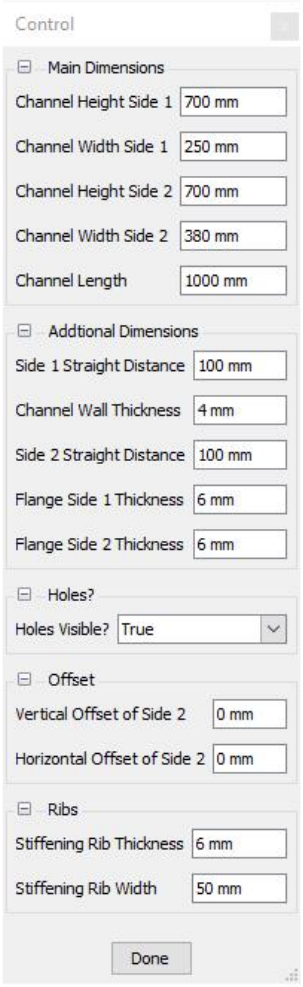
```
If ThisApplication.ActiveDocument.DocumentType = _
DocumentTypeEnum.kAssemblyDocumentObject Then _
'This should only work in assembly documents.

        Dim oAsm As AssemblyDocument
        oAsm = ThisApplication.ActiveDocument

        Dim oOcc As ComponentOccurrence

        oOcc = _
ThisApplication.CommandManager.Pick( _
SelectionFilterEnum.kAssemblyOccurrenceFilter, _
"Please select a component")

        If oOcc Is Nothing Then
                Exit Sub _
        'In case of "esc is hit during the pick command"
        End If

        Dim oDoc As Document = oOcc.Definition.Document

        'MsgBox(oDoc.FullFileName) 'Show fullfilename in messagebox

        'InputBox("This is the selected Occurrence Fullfilename:", _
 "Filename", oDoc.FullFileName) 'Show the Fullfilename in a Dialogbox TextBox

        'Process.Start(System.IO.Path.GetDirectoryName _
(oDoc.FullFileName)) 'Open containing folder in windows explorer

        My.Computer.Clipboard.SetText(oDoc.FullFileName) _
        'Copy the fullfilename to the clipboard.

End If
```

Possible use cases

Library parts and Know-How:

Example:

It is a transition piece for a square duct.
The iLogic Form, which controls the transition piece, can be seen on the left.
The use of the iLogic Forms alone ensures that users do not have to navigate to the parameter menu in order to know which parameters should be changed, but can see this directly.
A laborious search through the parameter list is no longer necessary.



Programmauszug;

```
dim_L = dim_L

If dim_L < 300 Then
    Parameter("dim_L") = 300
End If

If dim_L > 3200 Then
    Parameter("dim_L") = 3200
End If

If dim_L < 800 Then
    Feature.IsActive("Rib_1") = False
    Feature.IsActive("Rib_2") = False
    Feature.IsActive("Rib_3") = False

    'Egal
    Parameter("d151") = 120
    Parameter("d195") = 130
    Parameter("d196") = 140
ElseIf dim_L >= 800 And dim_L < 1600 Then
    Feature.IsActive("Rib_1") = True
    Feature.IsActive("Rib_2") = False
    Feature.IsActive("Rib_3") = False

    Parameter("d151") = -Math.Floor(dim_L/2 - (dim_Rib_t/2))

    'Egal
    Parameter("d195") = -130
    Parameter("d196") = -140
ElseIf dim_L >= 1600 And dim_L < 2400 Then
    Feature.IsActive("Rib_1") = True
    Feature.IsActive("Rib_2") = True
    Feature.IsActive("Rib_3") = False
```

In combination with the iLogic rule option, automatic mode on, this line means that if the parameter dim_L is changed, this rule will be executed.

Know How: Channel must not be longer than 3200 and not shorter than 300

If the length has a certain value, then certain things should happen (ribs shown or hidden etc.)

.
.
.

Calling an external program with command line arguments:

```
Shell("F:\Programme\ProgrammA.exe Document=""C:\Dings.ipt""", _
AppWinStyle.NormalFocus, True)
```

iLogic Forms for starting programs:

Teaching the ribbon new buttons only works with add-ins. Therefore, the possibility to quickly add access to programs or functions via iLogic Forms and to provide them with appealing icons is very useful: