



Autodesk® Inventor® iLogic™ or Inventor API: Which Is the Right Tool for Me?

Wayne Brill – Autodesk

CP2544 This class will compare the capabilities of iLogic with the Inventor API. We will cover the advantages/disadvantages of using iLogic and the advantages/disadvantages of using the Inventor API. You will see how iLogic and the Inventor API are able to automate Inventor behavior. You will discover the skill sets you need to use iLogic or the Inventor API using VB.NET. In this session you will gain an understanding of which tool to use to automate common tasks. Either iLogic or the Inventor API, or maybe both!

Learning Objectives

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

- Understand how iLogic is used
- Understand the basics of how the Inventor API is used
- Be able to determine when to use iLogic or the Inventor API

About the Speaker

Wayne has been a member of Autodesk Developer Technical Services supporting Autodesk Developer Network members for ten years. Wayne provides API support Autodesk® Inventor®, for AutoCAD®, AutoCAD Mechanical, AutoCAD OEM, and RealDWG™.

wayne.brill@autodesk.com

Introduction

In my role as an Inventor API consultant with Developer Technical Services at Autodesk I have had the good fortune to be able to help many people with the Inventor API. In the last several years I have also supported iLogic. This utility built right into Inventor is allowing many companies to automate the configurations of their designs and save time and money. This session will get you up to speed on these technologies.

I believe most people who attend this class will agree that they are in one of these four groups. (Perhaps with some overlap)

1. An Inventor user who knows and uses iLogic but is unfamiliar with the Inventor API.
2. A Programmer who knows the Inventor API but does not work with iLogic.
3. A Programmer who has not worked with the Inventor API and has not used iLogic either.
4. An Inventor user who does not use iLogic or the Inventor API.

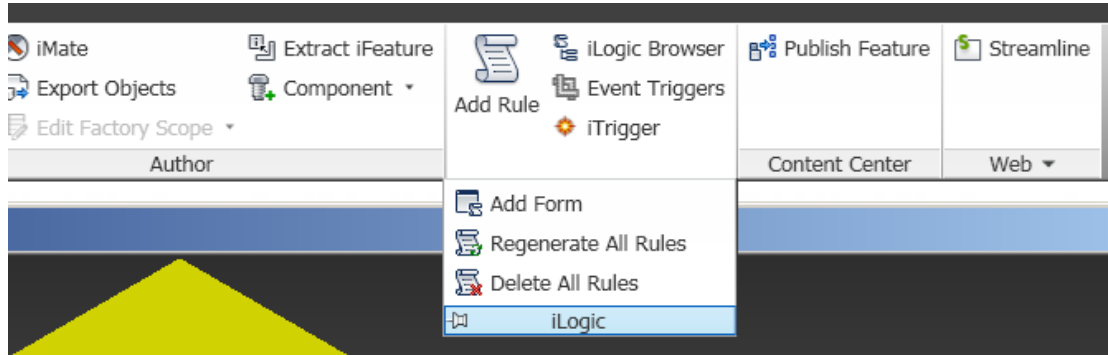
Because of the different types of people attending this session, all of the topics in this document may not be that helpful to you. For example, if you are a power user and proficient with iLogic, you can skip the parts that are intended to get someone up to speed on that part of Inventor. If you know the Inventor API then you can skip the parts of this document that focus on the API.

Contents:

[iLogic and Inventor API comparison](#)
[Inventor API Introduction](#)
[iLogic Introduction](#)
[Using the Inventor API in iLogic rules](#)
[iLogic API](#)
[Additional Resources](#)

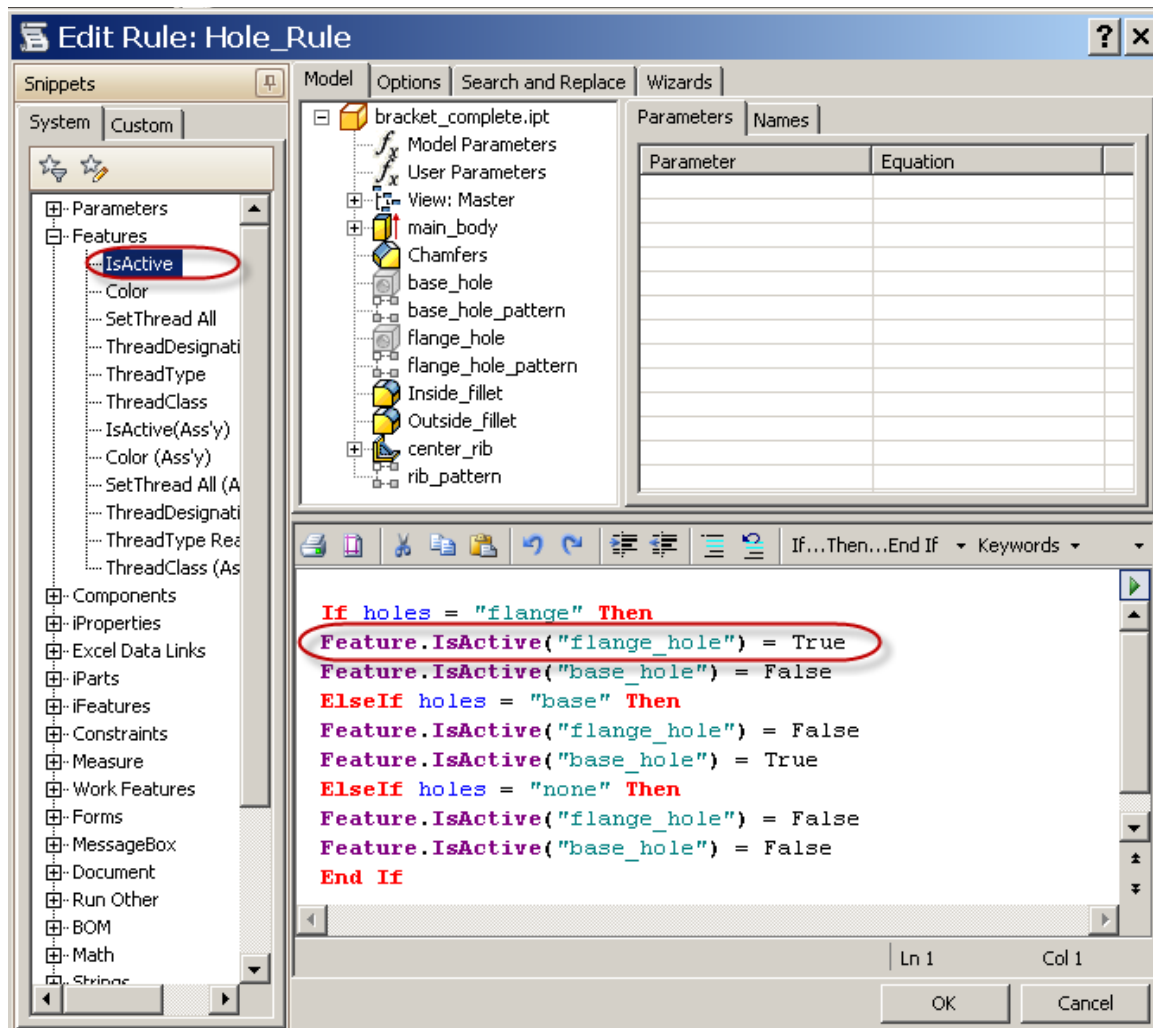
iLogic and Inventor API comparison

iLogic is designed to allow you to create different configurations of your model or drawing. The configuration of the model changes when the value of a parameter changes. iLogic ships with Inventor and if the iLogic AddIn is loaded, you will see the iLogic panel on the Manage Tab:

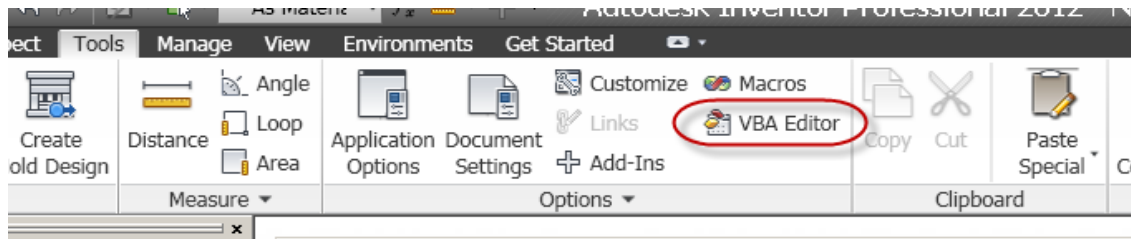


You use items on this panel to access iLogic functionality. Add Rule will open up the Edit Rule dialog. In the Edit Rule dialog you create the rules by selecting functions and making changes to the code that is inserted into the Edit Rule dialog. You can also enter code by typing directly in the rule area. iLogic rules are saved with the document and the rules run automatically when the value of Inventor parameters change. iLogic also supports External Rules and Global forms that are not saved in the document. iLogic rules can also run straight VB.NET code.

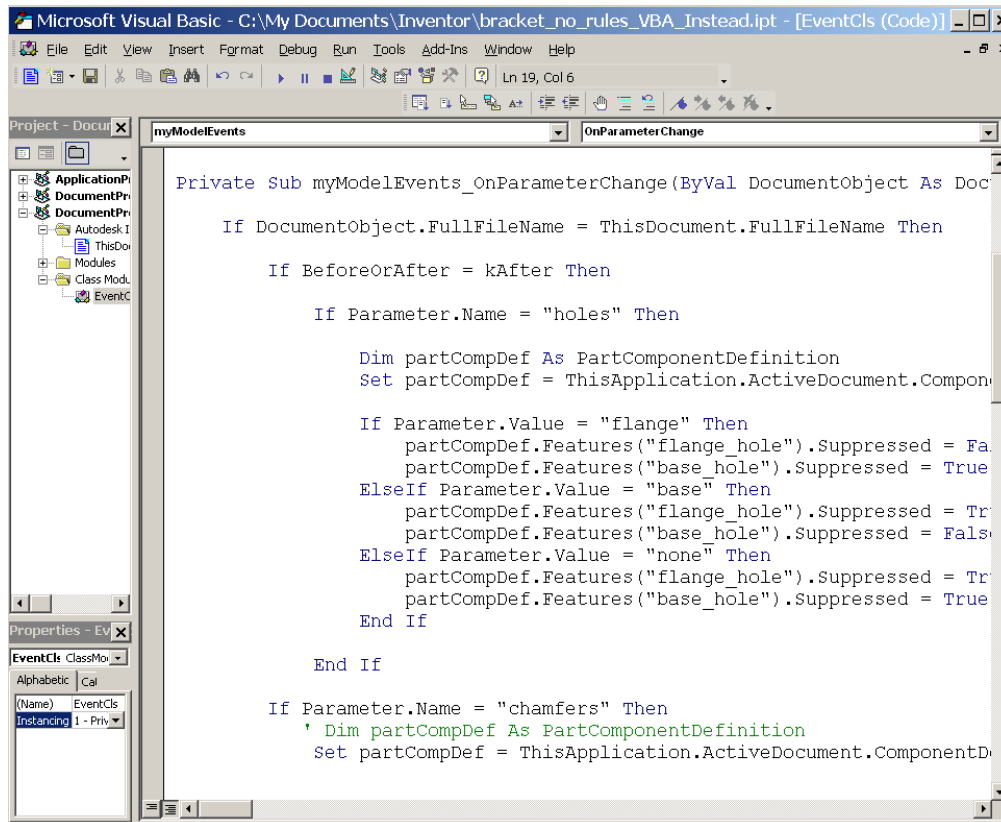
The rule shown in this screenshot of the Edit Rule dialog suppresses or unsuppresses features based on the value of a parameter named holes. Notice how the iLogic rule uses the iLogic function "Feature.IsActive"



The Inventor API is also installed with Inventor. The quickest way to access the Inventor API is by using the VBA editor. You can launch the VBA integrated development environment (IDE) from the Tools Tab.



When you use the Inventor API you use Sub Procedures and functions instead of iLogic rules. These procedures would need to be run by the user or be called by an event. Using the Parameter changed event will be needed to get similar functionality as an iLogic rule that runs when a parameter is changed. In this screenshot of the VBA IDE you can see code that will suppress or unsuppress features. The iLogic rule shown above is also suppressing and unsuppressing features.



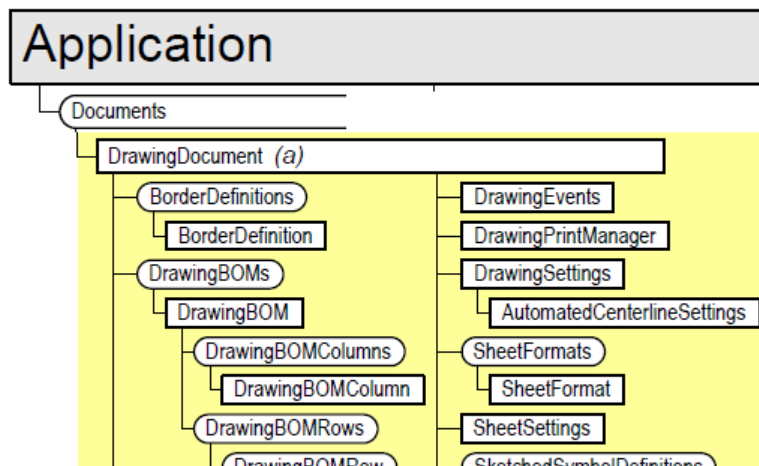
VBA IDE showing code using the Inventor API

In the code window of the VBA editor notice the code is using the Inventor API directly. It gets the PartComponentDefinition of the ActiveDocument then uses the Features collection and sets the Suppressed property of the feature to true or false. (Remember the iLogic rule used "Feature.IsActive") From this you can get the idea that iLogic is doing a lot of things in the background that you don't actually see in the rule. This is one reason why iLogic is less complicated to use than using the Inventor API directly.

Inventor API Introduction

The Inventor API is a COM API that is used from many different environments such as VB.NET, C#, C++. If you take a look at the Inventor API object model you will see that many of the objects in the Inventor API are organized to follow the main document types in Inventor. (Parts, Assemblies, Drawings). Other sections in the API are used to create user interfaces and other objects that are not related to a specific document type. You can download a PDF that contains a diagram of the Inventor API Object Model from this URL-

<http://www.autodesk.com/developinventor> (You will find other resources available for the Inventor API as well)



ObjectModel screenshot of DrawingDocument object

The Object Model PDF is also available from the Inventor SDK. (Software Development Kit) The SDK will be in a directory similar to this:

..\Inventor 2012\SDK\DeveloperTools\Docs

The SDK ships with Inventor however separate msi files need to be run to install it. These are DeveloperTools.msi and UserTools.msi. The location of the installer is different for Windows XP and Windows Vista or Windows 7.

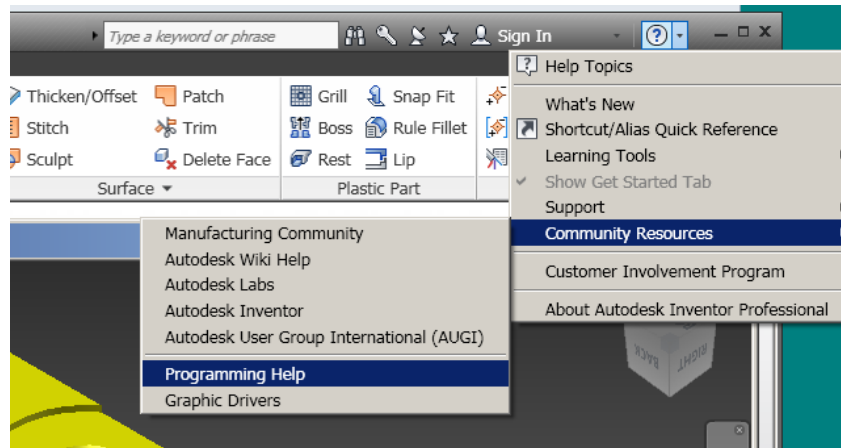
Windows XP: C:\Program Files\Autodesk\Inventor 2012\SDK\DeveloperTools.msi

Windows Vista or 7: C:\Users\Public\Documents\Autodesk\Inventor 2012\SDK\DeveloperTools.msi

For more information on the Inventor API SDK see this file:

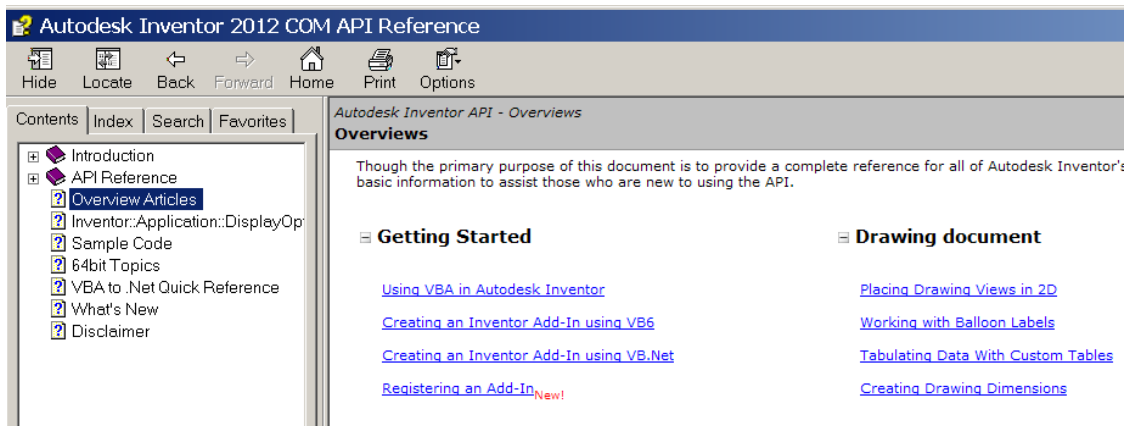
C:\Program Files\Autodesk\Inventor 2012\SDK\SDK_Readme.htm

Another great source for getting up to speed with the Inventor API is the COM Help documentation. You can open this file from Inventor ribbon as seen in the following screenshot:



How to open the Inventor API help

The Overview Articles topic is a great starting point:

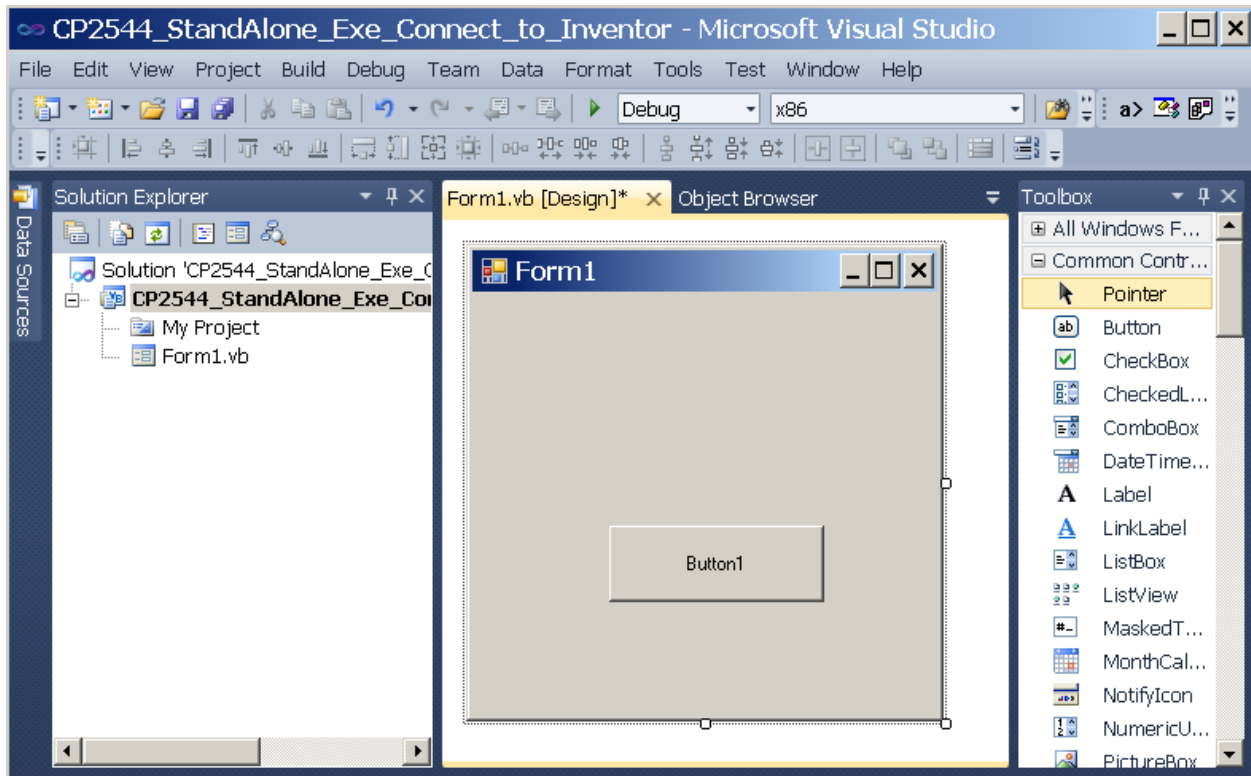


Overview Articles in the Inventor API help

You can use any language that supports COM automation such as VB.NET, C# and C++ to work with the Inventor API. However the quickest way to see how the Inventor API works is to use Visual Basic for Applications (VBA) which is installed with Inventor. VB.NET is also a good choice because of its ease of use and because it is in many cases it is straightforward to copy code from VBA to VB.NET. (VBA may not be available in future releases of Inventor so it is a good idea to plan on using another environment for production code). You can use the express versions of Visual Studio to work with the Inventor API. (Free download) In this session we will focus on using VB.NET and VBA. (Using an AddIn will probably be what you would use for your production code).

VB.NET Example

To try out the Inventor API you can use Visual Studio and create a windows program that connects to Inventor from out of process. Here is a screenshot of Visual Studio 2010 that has a “Windows Forms Application” (created by the VS Wizard). You add the button by placing the button from the toolbox. When this program is run, it will connect to a running session of Inventor. When the button is clicked it will run code that we will add below.

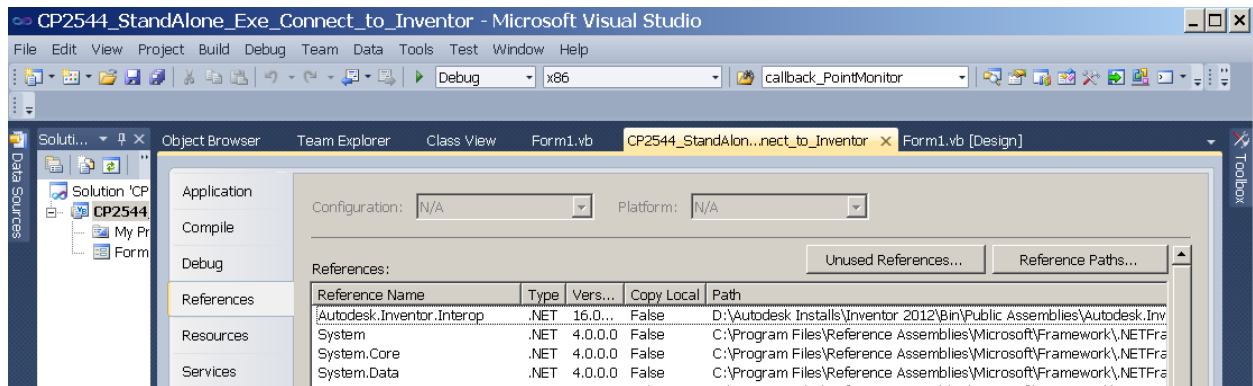


Visual Studio 2010 Windows Forms Application

After the project is created you need to add the Inventor API to the program, to do this bring up properties (right click on the project in the Solution Explorer) and go to “Add...” on the References tab. Navigate to this location: “C:\Program Files\Autodesk\Inventor 2012\Bin\Public Assemblies” and select this file: “Autodesk.Inventor.Interop.dll” (this is on XP, on Win7 the directory will be different)

Here is a screenshot after adding the reference to “Autodesk.Inventor.Interop.dll”. Notice the version is 16.0, this is the Inventor 2012 COM interop file.

Autodesk® Inventor® iLogic™ or Inventor API: Which Is the Right Tool for Me?



Inventor COM Interop is referenced

Now we can go ahead and use the Inventor API to do something. In this code below we start or connect to a session of Inventor when the form is created. Notice the “Imports Inventor” before the Class Form1. This brings in the Inventor namespace. A variable named `_InvApp` is declared as an `Inventor.Application` we will use this variable of the top level API object to do some work when the button is clicked. (Code for this farther below) Notice here that the ProgID - “Inventor.Application” is used with the `GetActiveObject` method. The Windows registry is searched using this ProgID. The ProgID is added to the registry when Inventor is installed. The “Try Catch” block is the .NET way to do error handling. If there is an error then we use `CreateInstance` to start up a new session of Inventor. When we start Inventor this way we need to shut it down or the Inventor exe will still be running. In the `FormClosed` event the `Quit` method is called on the Inventor Application.

```
Imports System
Imports System.Type
Imports System.Activator
Imports System.Runtime.InteropServices
Imports Inventor

Public Class Form1

    Dim _invApp As Inventor.Application
    Dim _started As Boolean

Public Sub New()

    ' This call is required by the designer.
    InitializeComponent()

    ' Add any initialization after the InitializeComponent() call.
    Try
        _invApp = Marshal.GetActiveObject("Inventor.Application")

    Catch ex As Exception
        Try
            Dim oInvAppType As Type = _
```

```

        GetTypeFromProgID("Inventor.Application")

        _invApp = CreateInstance(oInvAppType)
        _invApp.Visible = True

        'Note: if you shut down the Inventor session that was started
        'this(way) there is still an Inventor.exe running. We will
use
        'this Boolean to test whether or not the Inventor App will
        'need to be shut down.
        _started = True

    Catch ex2 As Exception
        MsgBox(ex2.ToString())
        MsgBox("Unable to get or start Inventor")
    End Try
End Try

End Sub

Private Sub Form1_FormClosed( _
    ByVal sender As Object, ByVal e As FormClosedEventArgs) _
    Handles Me.FormClosed

    If _started Then
        _invApp.Quit()
    End If

    _invApp = Nothing

End Sub

```

Now let's take a look at code for the button. Notice how it uses the `_InvApp` variable created above. (It was instantiated when the form was loaded). First an "If then" statement is used to ensure there is at least one open document. Then we display a message box with the number of open documents by using the Documents collection and the count property. We also display the name of the active document using the FullFileName property of the ActiveDocument.

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Try
        If _invApp.Documents.Count = 0 Then
            System.Windows.Forms.MessageBox.Show _
                ("Need to open a document")

            Return
        End If
        'Get then number of open documents and display it
        Dim iNumOfDocs As Double = _invApp.Documents.Count
        System.Windows.Forms.MessageBox.Show _
            ("Number of open documents = " & iNumOfDocs)
    End Try
End Sub

```

```

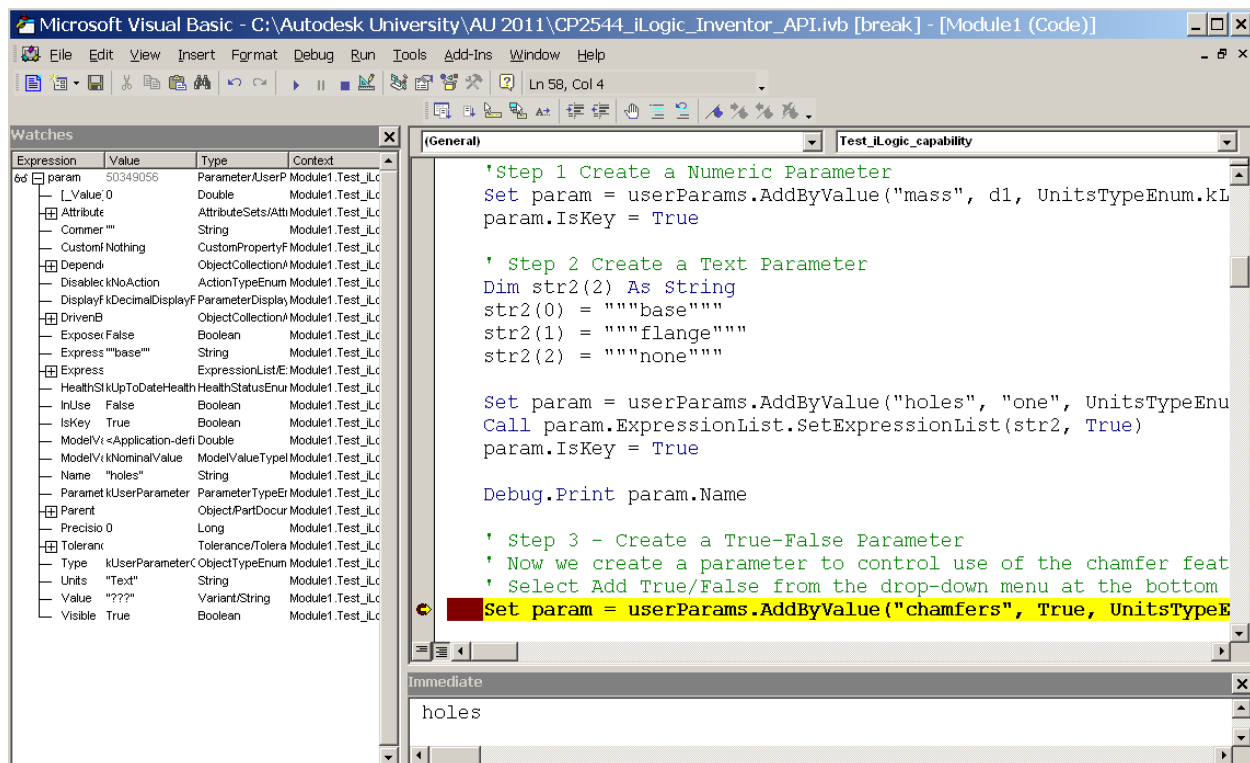
'Get the active document and display the file name
Dim oDoc As Document = _invApp.ActiveDocument
System.Windows.Forms.MessageBox.Show _
    ("Active document = " & oDoc.FullFileName)

Catch ex As Exception
    System.Windows.Forms.MessageBox.Show(ex.ToString())
End Try

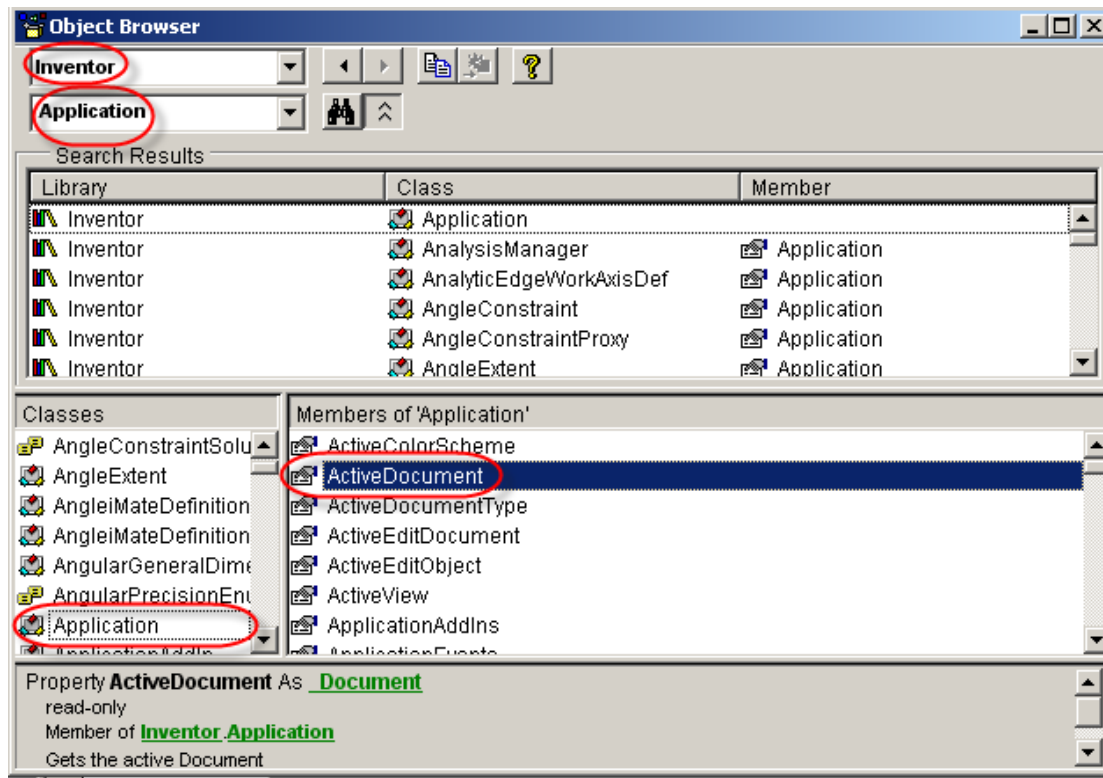
End Sub
End Class

```

We recommend using VB.NET or other programming environment for production applications. (VBA could be removed in future versions of Inventor). However VBA can be useful for quick prototyping and debugging. In this screenshot you see the Watches window that shows the current values of a parameter at runtime. (When the break point was hit). You can use Debug.Print to print values in the immediate window. In this example the name property of the param is “holes”.



Also the Object Browser provides a way to explore the objects in the Inventor API. You can display the Object Browser from the VBA View Menu (View>Object Browser or F2). You can select a library to search and a string to search on. In the screen shot below shows the Inventor Library is selected and the search is for Application. The selected class is Application. The selected member is the ActiveDocument property. (The Object Browser is also available in VB.NET)



Skill Set needed for Inventor API

1. Be able to access an API based on COM (component Object Model) using a programming language such as VB.NET or C#.
2. Know how to work with the Inventor API Objects (there are a lot of them)

Advantages of using the Inventor API compared to iLogic rules

1. Ability to create AddIns that have a User Interface (Ribbon elements, Toolbars, menus) Using iLogic rules to implement a user interface for a large application could be user unfriendly.
2. Intellisense and debugging are available in tools such as VB.NET. (When you close the iLogic Edit Rule dialog, if there is a problem it will only tell you which line is causing the error)

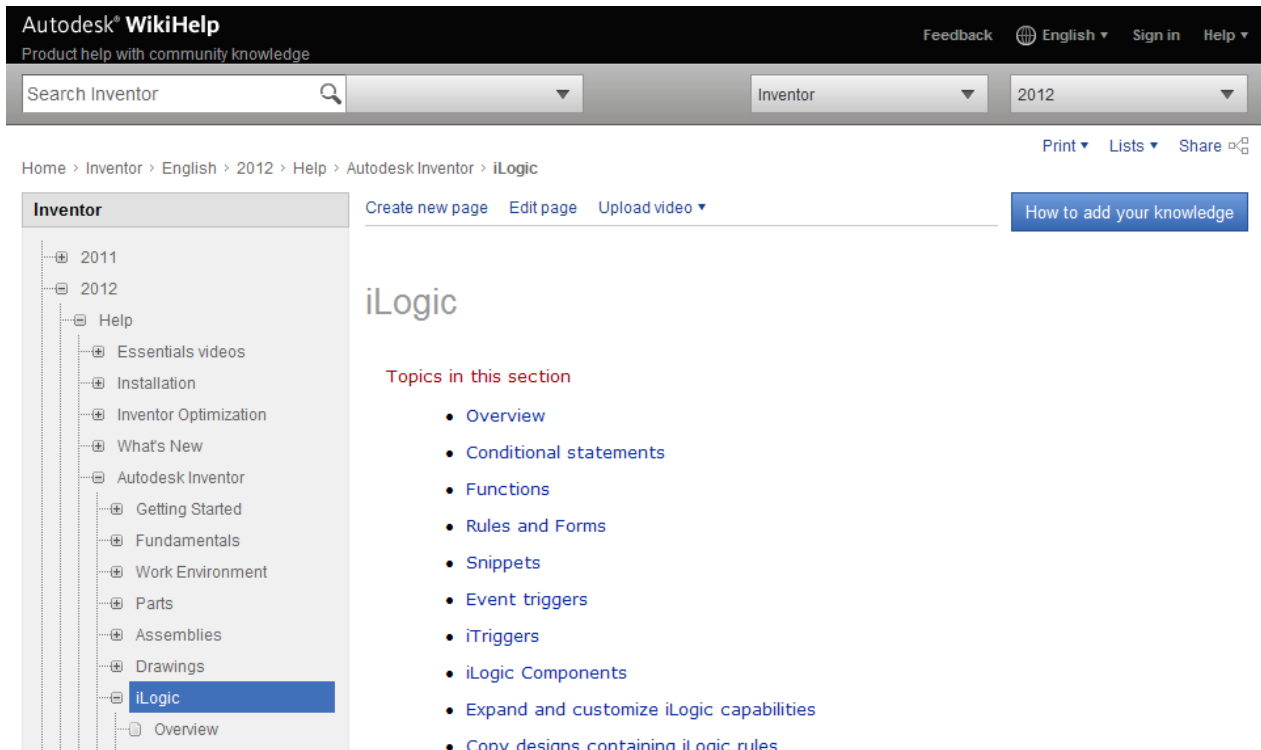
iLogic Introduction

The main purpose of iLogic is to enable a way for companies to create different configurations of assemblies easily (by just changing a parameter). With iLogic the parameter is more than just a formula, but actually drives a program (rules) to create different configurations of an assembly, part or drawing. This functionality of Inventor got many people into automating Inventor even though they would not consider themselves programmers.

Using the Inventor API directly requires more programming than using iLogic. iLogic achieves this by providing a user interface (Edit Rule dialog) that allows you to add code to a rule text window by clicking on items in the dialog and then changing values of the added code to the correct values, so they match the names of elements in the document. (Parameter, Feature, component names etc.)

The Inventor help has a great section on iLogic. Here is the link:

<http://wikihelp.autodesk.com/Inventor/enu/2012/Help/0073-Autodesk73/0673-iLogic673>



The screenshot shows the Autodesk WikiHelp interface. At the top, there's a navigation bar with "Autodesk® WikiHelp", "Product help with community knowledge", "Feedback", "English", "Sign in", and "Help". Below this is a search bar labeled "Search Inventor" and two dropdown menus for "Inventor" and "2012". The main content area has a breadcrumb trail: "Home > Inventor > English > 2012 > Help > Autodesk Inventor > iLogic". On the left is a sidebar with a tree view of the help content, where "iLogic" is selected. The main content area is titled "iLogic" and lists "Topics in this section": Overview, Conditional statements, Functions, Rules and Forms, Snippets, Event triggers, iTriggers, iLogic Components, Expand and customize iLogic capabilities, and Copy designs containing iLogic rules. There are also links for "Create new page", "Edit page", "Upload video", and "How to add your knowledge".

Also these iLogic videos on You Tube will be very helpful when getting started with iLogic.

[Autodesk Inventor 2012 Tutorials - iLogic Part 1 - Overview](#)

[Autodesk Inventor 2012 Tutorials - iLogic Part 2 - Parameters](#)

[Autodesk Inventor 2012 Tutorials - iLogic Part 3 - Parameter Filters](#)

[Autodesk Inventor 2012 Tutorials - iLogic Part 4 - Feature Suppression](#)

[Autodesk Inventor 2012 Tutorials - iLogic Part 5 - Dimensions](#)

There are three iLogic tutorials in the help file: iLogic Basics, iLogic – Part Modeling and iLogic Assemblies. Here is the link:

<http://wikihelp.autodesk.com/Inventor/enu/2012/Help/2144-Tutorial2144/2145-Inventor2145>

The main objective of the tutorials is to help you understand how to create iLogic rules. When you create an iLogic rule, it is similar to creating a procedure or function in the API. A rule will have two things:

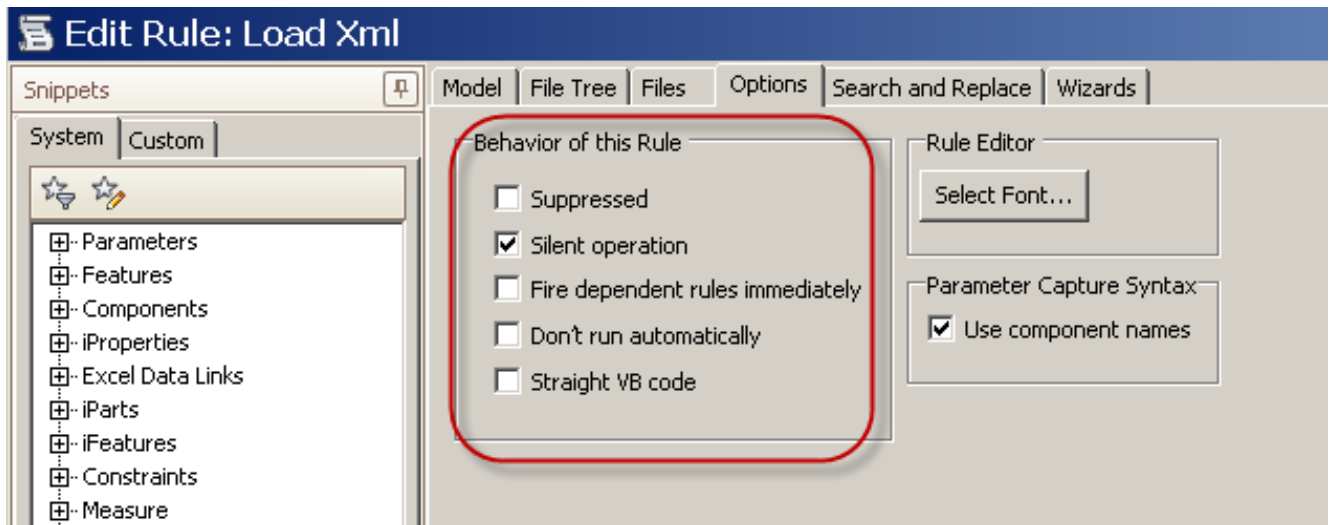
1. VB.NET code
2. iLogic functions

The VB.NET code is similar to code used in a Visual Studio VB.NET project. The iLogic functions will only work in an iLogic rule however. You can also run VB.NET code directly. (This VB.NET code can use the Inventor API). Five things that are specific to iLogic and are not part of the Inventor API are the following:

1. User interface elements such as the Edit Rule dialog
2. iLogic functions
3. Forms created by the iLogic AddIn (using the Form Editor)
4. The automatic behavior of running iLogic rules
5. Rules saved in the document

Control the behavior of Rules:

You can control the behavior of the rule by changing the settings on the Options tab in the Edit Rule dialog.



Here is the description of these settings:

Suppressed - The rule does not run automatically

Silent Operation – Stops Inventor dialogs from displaying

Fire dependent rule immediately – If another rule refers to a parameter that this rule changes, that other rule will run immediately

Don't run automatically – The rule does not run when a parameter changes

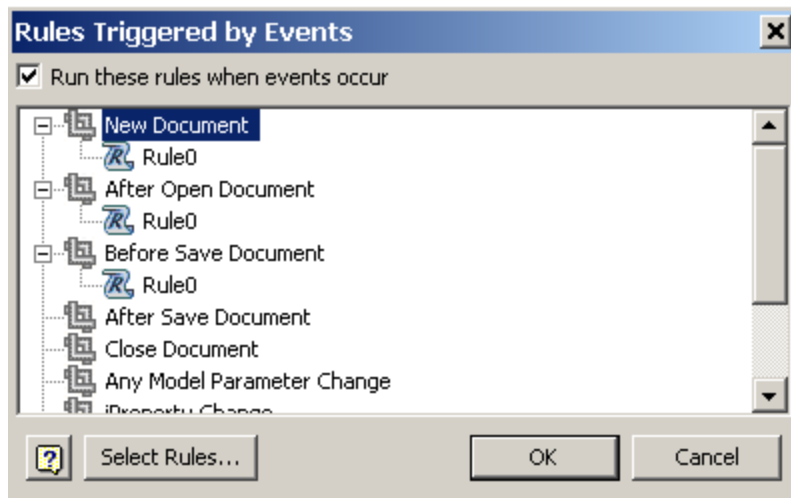
Straight VB code – Contains VB code that can be run from a different rule, (parameters that are referenced will not automatically cause this rule to be run)

Sub Main

The Sub Main in a rule is needed if you have other functions (or sub) in the rule. If you add a function (most likely to avoid duplicating the same code several places in the rule), then you need Sub Main to delimit the main code of the rule. The code you put in Sub Main is the code that is guaranteed to run when the rule runs. To run code in other function's or sub's in the rule you would call them from the main function.

Event Triggers

Using Event Triggers allow you to hook up rules to run when an event occurs. Here is the dialog that allows you to control which rule runs when the event occurs. The New Document rule would run when a new document is created based (template) on the document with the rule.



iTrigger

iTrigger provides a way to run a rule manually from the iLogic Panel in the ribbon. When you click on “iTrigger” a user parameter named iTrigger0 is added. After this, every time “iTrigger” is clicked, it updates the value of the parameter. If you have a rule that refers to this Parameter the rule will run. (It does not matter what the name of the variable is, in this case it is myTrigger).

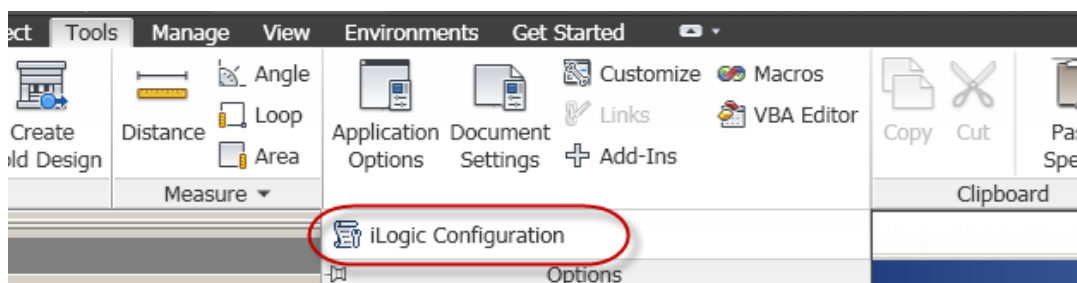
```
myTrigger = iTrigger0
MessageBox.Show("Message", "Title")
```

External Rules

Rules can be in text files and can be accessed from rules that are saved in the document. In the iLogic Rule Browser you can add the external rules by selecting them by right clicking on the External Rules tab and selecting “Add External Rule” (You can multi-select, to add all files in a directory). The external rules are global: they aren’t tied to a particular document. They can be used in a way similar to VBA macros.

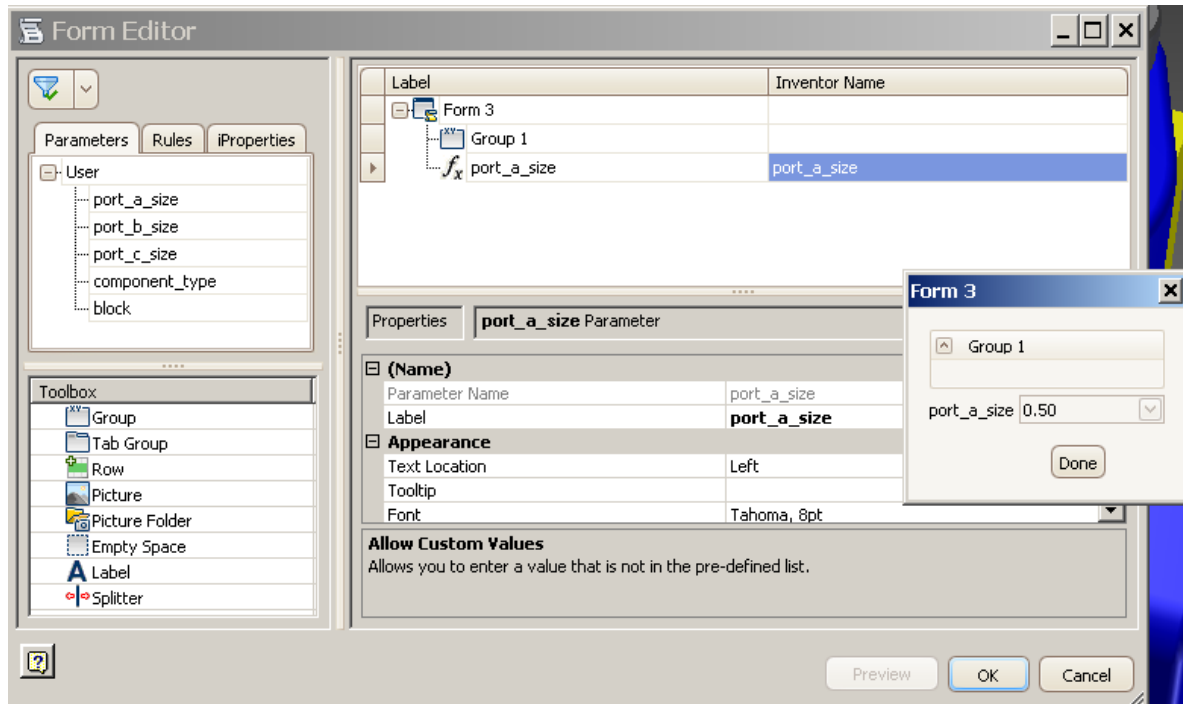
iLogic Configuration

This iLogic feature is launched from the Options Panel on the Tools tab. iLogic Configuration will allow you to set the search directories for External Rules and DLLs that are used from iLogic rules. You can also set the default extension for External Rules.



iLogic Forms

You can create a UI for iLogic using the Forms and Global Forms tab in the iLogic Browser. To add a new form right click on the tab and select “Add Form”. This will launch the Form Editor where you can drag and drop to create the user interface on the form. There is a preview that shows you what the form looks like. (You could also use an external dll that defines the user Interface to work with rules).



Skill Set needed for iLogic

1. Understand Parameters and how they drive rules
2. Understand some VB.NET code such as If statements
3. Understand the elements in the rules are, (know what the colors indicate)
4. Understand Functions like - GoExcel - automatically opens a spread sheet

Advantages of using iLogic

1. Quickly create different configurations of the design based on parameter values without having to create events.
2. Easier for non programmers - don't have to learn about programming, can use iLogic functions using drag and drop.

Using the Inventor API in iLogic rules

There are two reasons you may want to use the Inventor API directly in an iLogic rule.

1. There is not an iLogic function that does what you need.
2. You know the Inventor API better than you know the iLogic functions

One of the most common uses of the Inventor API in an iLogic rule is when you need to run operations on multiple items. Most of the iLogic functions require the name of a specific item, and only work on that item. So to look at all items of a given type, you need to get the collection from the API and use a loop to iterate over it. Here is a rule that uses the Inventor API to loop through the layers collection and sets the visibility of the layers to False.

```
Dim drawDoc as DrawingDocument = ThisDrawing.Document
Dim layers as LayersEnumerator = drawDoc.StylesManager.Layers
For Each layerX As Layer In layers
    layerX.Visible = False
Next
```

Inventor API constants in an iLogic rule, you have to add the Enum type

When you use Inventor API constants in an iLogic rule, you have to add the Enum type. (This is the case when as using VB.NET also). In this example **kSolidOutputType** is a constant. To use it in a rule, you need to include the enum type (**BaseFeatureOutputTypeEnum**) as is demonstrated in this code snippet:

```
oFeatureDef2.OutputType = BaseFeatureOutputTypeEnum.kSolidOutputType
```

You can use the help file to find the Enum type. Searching on **kSolidOutputType** will turn up a page with this information:

```
Public Enum BaseFeatureOutputTypeEnum
    kSurfaceOutputType = 90113
    kSolidOutputType = 90114
    kCompositeOutputType = 90115
End Enum
```

An alternative to adding the enum type is to add an Imports statement in the rule:

```
Imports Inventor.BaseFeatureOutputTypeEnum

oFeatureDef2.OutputType = kSolidOutputType
```

Test the Inventor API in a rule

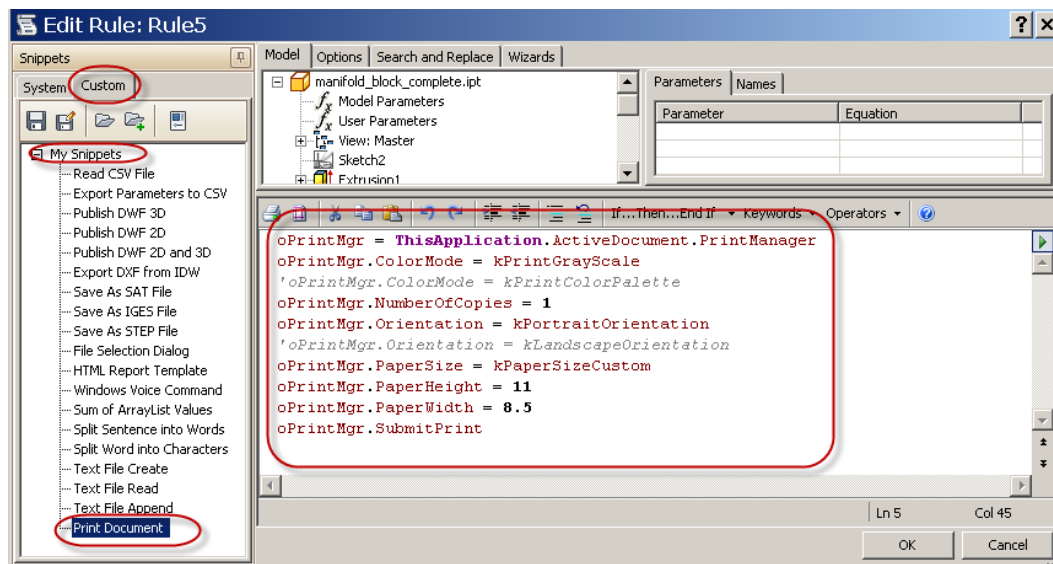
One way to explore using the Inventor API directly from iLogic is to perform the following steps:

1. Open the Inventor API help file.
2. Go to the examples topic.
3. Select one of the VBA examples.
4. Open a VB.NET project in Visual Studio (Express version is a free download)
5. Paste the VBA example into a VB.NET code window (this removes the set statements) If the VB.NET project has referenced the Inventor interop.dll and the Inventor Namespace has been imported in the code window you can easily fix the enumerations. VB.NET wants to have the enumerations qualified. (Fix the enumerations)
6. Copy the code.
7. Paste the code into the rule editor
8. Change the code to use Thisdoc.Document instead of ThisApplication.ActiveDocument

One reason why this approach is useful if you are going to use the Inventor API from iLogic is because the iLogic rule text window does not support intellisense or debugging. If you get the code working from VBA or VB.NET then it should be able to work in a rule with just minor changes.

Example of using an iLogic code snippet that uses the Inventor API

Many of the Publish and Save rules Under the Custom snippets, are using the API directly. Here is a screenshot of the Edit Rule dialog. Notice the code in the Print Document rule:



You can copy the code from this rule into a VBA procedure and add the Set statement to the first line (Set oPrintMgr = ThisApplication.ActiveDocument.PrintManager) and it will run without any other changes.

The code will also work in VB.NET with a few minor changes. Below is the working code from a VB.NET project. (A form with one button). These are the changes needed:

1. ThisApplication is instantiated using the Inventor.Application that is instantiated using GetActiveObject in the New procedure of the form. (ThisApplication works in VBA but not VB.NET)
2. oPrintMgr is declared as an Inventor.PrintManager
3. VB.NET requires the full name of the enumerations, PrintColorModeEnum, PrintOrientationEnum and PaperSizeEnum.

```
Imports Inventor

Public Class Form1
    Dim invApp As Inventor.Application

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' MessageBox.Show("invApp " & invApp.ActiveDocument.FullFileName)
        Dim ThisApplication As Inventor.Application
        ThisApplication = invApp

        Dim oPrintMgr As Inventor.PrintManager

        oPrintMgr = ThisApplication.ActiveDocument.PrintManager
        oPrintMgr.ColorMode = PrintColorModeEnum.kPrintGrayScale
        'oPrintMgr.ColorMode = kPrintColorPalette
        oPrintMgr.NumberOfCopies = 1
        oPrintMgr.Orientation = PrintOrientationEnum.kPortraitOrientation
        'oPrintMgr.Orientation = kLandscapeOrientation
        oPrintMgr.PaperSize = PaperSizeEnum.kPaperSizeCustom
        oPrintMgr.PaperHeight = 11
        oPrintMgr.PaperWidth = 8.5
        oPrintMgr.SubmitPrint()
    End Sub

    Public Sub New()

        ' This call is required by the designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call.
        invApp =
System.Runtime.InteropServices.Marshal.GetActiveObject("Inventor.Application"
)
```

```
End Sub
End Class
```

Here are a few other examples of using the Inventor API with iLogic functions in an iLogic rule.

Set the same border for all sheets in a drawing

The iLogic drawing functions for sheets and views require you to specify the sheet or view by name. If you want to apply a function to all sheets or views, or read data from all, without looking at the total number of sheets, you can iterate through all the sheets and set the border like the following. Notice that a string is used to set the border. This is an iLogic function. In the API you would need to call AddBorder and use a BorderDefintion object not a string. The Inventor API in this example is the code using the Sheets collection.

```
borderName = "CustomBorder"
For Each sheetX As Sheet In ThisDrawing.Document.Sheets
    ThisDrawing.Sheet(sheetX.Name).Border = borderName
Next
```

Set the same scale for all views in a drawing

This rule iterates through each sheet, each view and changes the scale.

```
myScale = 1.5
For Each sheetX As Sheet In ThisDrawing.Document.Sheets
    For Each viewX As DrawingView In sheetX.DrawingViews
        If (Not viewX.ScaleFromBase) Then
            viewX.Scale = myScale
        End If
    Next
Next
```

Switch design views in an assembly

This rule will switch between two design views based on a True/False parameter named Component_Visibility

```
Dim assemDoc as AssemblyDocument = ThisDoc.Document
Dim viewReps as DesignViewRepresentations =
assemDoc.ComponentDefinition.RepresentationsManager.DesignViewRepresentations

If (Component_Visibility) Then
    viewReps("Default").Activate
Else
```

```
viewReps ("View1").Activate
End If
```

iLogic API

iLogic™ does have an API that you can use to automate iLogic to add or modify rules. You can also use the API to run code in dlls from rules. To use the iLogic API you need to reference a dll named “Autodesk.iLogic.Interfaces.dll”. This dll is located in the bin directory in the Inventor Install, which is usually:

C:\Program Files\Autodesk\Inventor 2012\Bin

There are several ways the interfaces in this dll are accessed. One way is to use the **liLogicAutomation interface**. The other way to use the iLogic API is called the “**iLogic Rule API**”. (The iLogic Rule API is discussed farther below).

liLogicAutomation Interface

This interface can be used from COM or .NET, from in-process or out-of-process. It has more functions than the ILowLevelSupport interface that is used for the iLogic Rule API.

iLogic AddIn Automation property

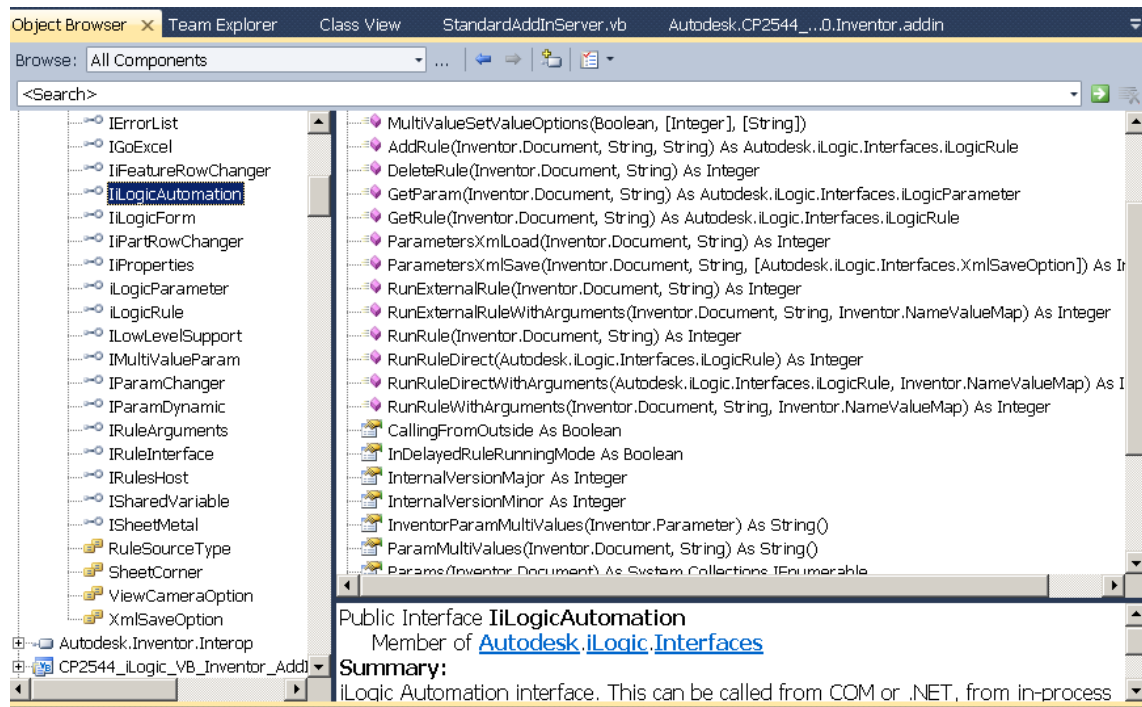
After you have referenced “Autodesk.iLogic.Interfaces.dll”, you use the ItemById method of the ApplicationAddIns collection of the Inventor Application to get the iLogic AddIn. The iLogic AddIn Automation property will allow you to instantiate the liLogicAutomation Interface. This code snippet from VB.NET AddIn running in-process shows how the GUID of the iLogic AddIn is passed into the ItemById method. The Automation property of the iLogic AddIn is used to instantiate liLogicAutomation.

```
Private _iLogicAutomation As IiLogicAutomation = Nothing

Dim iLogicAddIn As Inventor.ApplicationAddIn =
_invApp.ApplicationAddIns.ItemById("{3BDD8D79-2179-4B11-8A5A-257B1C0263AC}")

_iLogicAutomation = iLogicAddIn.Automation
```

This screenshot of the Object Browser in VB.NET showing IiLogicAutomation



VB.NET AddIn example that adds a rule

This example iterates through the rules in a document and if a rule named ChangeView does not exist, it is added using the AddRule method of the IiLogicAutomation. Notice how a string for the rule is built up and is then set using the Text property of the new rule. The rule switches the design view based on the value of a parameter named View_Default_Or_View1. The call to a Sub AddParameterForView() will add the parameter to the document if it does not exist. (Code for this function is not shown here).

```
Private Sub mAsmButtonDef_OnExecute(ByVal Context As Inventor.NameValueMap)
    ' Get the Active Document
    Dim doc As Inventor.Document =
m_inventorApplication.ActiveDocument
    Dim bRuleExists As Boolean = False

    AddParameterForView()

    Try
        ' Get the rules
        Dim ruleCol As System.Collections.IEnumerable =
_iLogicAutomation.Rules(doc)
```

```

        ' make sure the rules collection exists
        If Not ruleCol Is Nothing Then
            ' string to hold the rule names
            Dim str_iLogRuleName As String = ""
            ' Go through each of the rules
            For Each iLogRule As iLogicRule In ruleCol
                ' Put the name of the rule in the string
                str_iLogRuleName = str_iLogRuleName & iLogRule.Name &
vbCrLf

                If iLogRule.Name = "ChangeView" Then
                    bRuleExists = True
                End If
            Next iLogRule
            ' Display the string with rule names
            System.Windows.Forms.MessageBox.Show(str_iLogRuleName)
        End If

        If Not bRuleExists Then

            ' Create a rule
            Dim newRule As iLogicRule =
                _iLogicAutomation.AddRule(doc, "ChangeView", "")

            ' string for the rule:
            Dim strRuleText As String
            strRuleText = " Dim assemDoc As AssemblyDocument =
ThisDoc.Document" & vbCrLf
            strRuleText = strRuleText & _
                "Dim viewReps As DesignViewRepresentations =
assemDoc.ComponentDefinition.RepresentationsManager.DesignViewRepresentations
" & vbCrLf
            strRuleText = strRuleText & " If (View_Default_Or_View1)
Then" & vbCrLf
            strRuleText = strRuleText &
"viewReps("""Default"").Activate()" & vbCrLf
            strRuleText = strRuleText & "Else" & vbCrLf
            strRuleText = strRuleText &
"viewReps("""View1"").Activate()" & vbCrLf
            strRuleText = strRuleText & "End If"

            ' Add the rule text
            newRule.Text = strRuleText

        End If

        ' Run the rule that was added
        _iLogicAutomation.RunRule(doc, "ChangeView")

    Catch ex As Exception
        System.Windows.Forms.MessageBox.Show(ex.ToString())
    End Try

End Sub

```


Late binding in an external exe

In an external exe, you cannot cast the iLogic AddIn's Automation object to the `liLogicAutomation` interface. You have to make it an Object and use late binding on it. Because of this there is no Intellisense or compile-time checking. However, you are able to access all the properties and functions. If you need to use the iLogic API from out-of-process then you can use early binding when you are writing your code and then change to late binding when you need to run it. (change your variables to Objects). Or you can use an Inventor® AddIn and run in-process.

liLogicAutomation interface in VBA

You have to use late binding in VBA with `liLogicAutomation`. `Autodesk.iLogic.Interfaces.dll` is a .NET-only assembly and it doesn't expose any COM interfaces. Because of this there is not a way to get intellisense or any error checking for the iLogic API in VBA. Late binding could work but it will be not be user friendly. If you really needed to use VBA you could create the code in VB.NET and then move it to VBA. It would take more work as the Set statements would need to be added to instantiate every object. In any case there is a VBA example "iLogicAutoTest.ivb" at a location similar to this:

Windows XP

Program Files\Autodesk\Inventor [version]\Samples\iLogic Samples\API\iLogicAutoTest.ivb

Windows Vista or (probably) Windows 7

Users\Public\Documents\Autodesk\Inventor [version]\Samples\ iLogic
Samples\API\iLogicAutoTest.ivb

Here is a code from this example. Notice how Object is used for all of the iLogic variables

```
Sub Test_iLogic()  
  
    Dim iLogicAuto As Object  
    Set iLogicAuto = GetiLogicAddin(ThisApplication)  
    If (iLogicAuto Is Nothing) Then Exit Sub  
  
    Dim curDoc As Document  
    Set curDoc = ThisApplication.ActiveDocument  
  
    iLogicAuto.ParamValue(curDoc, "text0") = "From Automation"  
  
    Dim curVal As String  
    curVal = iLogicAuto.ParamValue(curDoc, "text0")
```

```

Dim rule As Object
Debug.Print (" ----- Rules: -----")
Dim rules As Object
Set rules = iLogicAuto.rules(curDoc)
If (Not rules Is Nothing) Then
  For Each rule In rules
    Debug.Print (" --- " & rule.Name & " ---")
    Debug.Print (rule.Text)
  Next
End If

End Sub

Function GetiLogicAddin(oApplication As Inventor.Application) As Object

  Set addIns = oApplication.ApplicationAddIns

  'Find the add-in you are looking for
  Dim addIn As ApplicationAddIn
  On Error GoTo NotFound
  Set addIn = oApplication.ApplicationAddIns.ItemById("{3bdd8d79-2179-4b11-8a5a-257b1c0263ac}")

  If (addIn Is Nothing) Then Exit Function

  addIn.Activate
  Set GetiLogicAddin = addIn.Automation
  Exit Function
NotFound:
End Function

```

iLogic Rule API

The other way you can automate iLogic is using the iLogic Rule API. With this API you still reference “Autodesk.iLogic.Interfaces.dll”. (Located in C:\Program Files\Autodesk\Inventor 2012\Bin). One of the main differences is that you do not use the Automation property of the iLogic AddIn to instantiate the iLogicAutomation Interface. Instead you use iLogicVB from a rule to instantiate ILowLevelSupport in the dll. ILowLevelSupport is a .NET-only, iLogic Rule API interface. Other interfaces that are commonly instantiated this way are IParamDynamic and IMultiValueParam.

Here is an excerpt from a sample that ships with Inventor that uses the iLogic Rule API.

(C:\Program Files\Autodesk\Inventor 2012\Samples\iLogic Samples\Railing
Advanced\RailingFormAdvanced)

Notice the declarations for IParamDynamic, IMultiValueParam and ILowLevelSupport

```
Public Class RailingFormAdvanced
    ' These are instantiated by the rule
    Public Parameter As IParamDynamic
    Public MultiValue As IMultiValueParam
    Public iLogicVB As ILowLevelSupport
```

In the rule that uses this dll these interfaces are instantiated. Here is the code snippet from the rule:

```
AddReference "RailingFormAdvanced.dll"

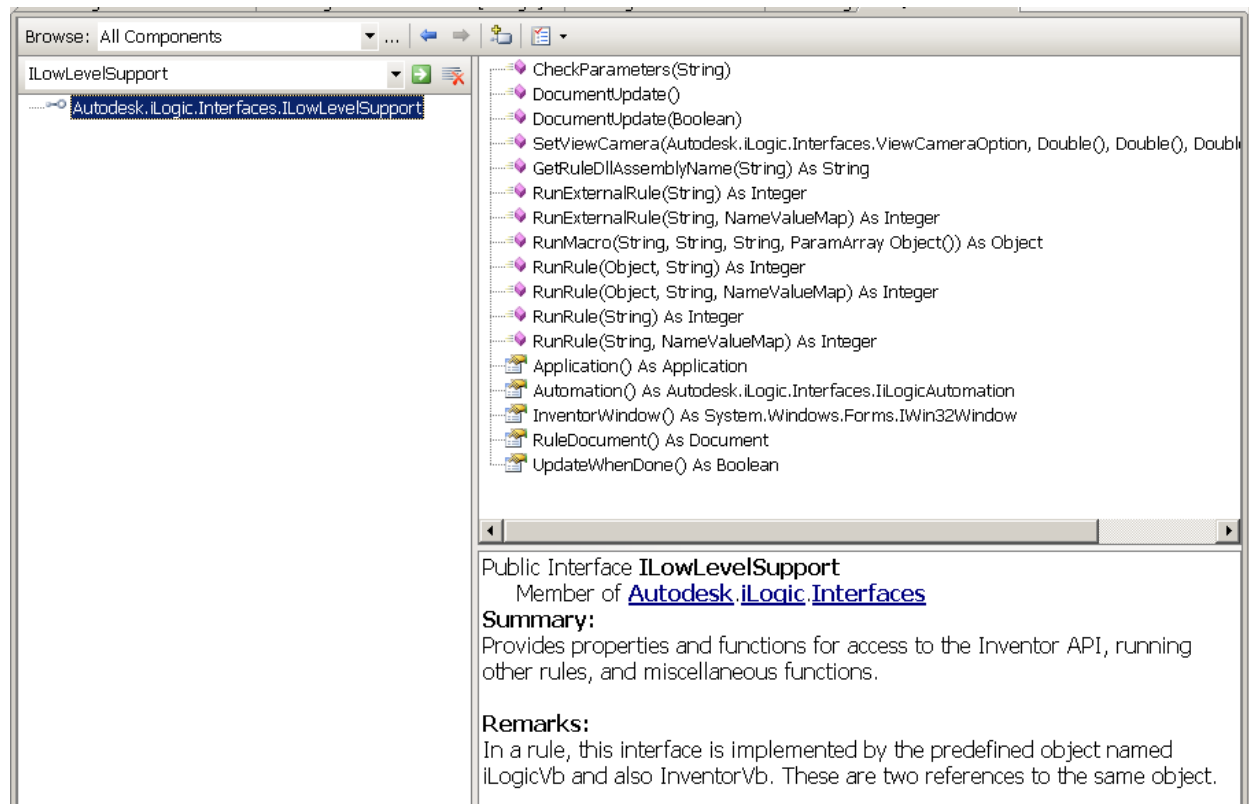
Sub Main()

Using dlg As New RailingFormAdvanced.RailingFormAdvanced

    dlg.Parameter = Parameter
    dlg.MultiValue = MultiValue
    dlg.iLogicVb = iLogicVb
```

Notice how this rule uses AddReference to add a reference to the dll. Then in Sub Main() the RailingFormAdvanced class is instantiated with the New keyword and the Parameter, MultiValue and iLogicVB are instantiated using the iLogic objects.

This screenshot of the ObjectBrowser shows the elements of ILowLevelSupport:



Summary

Thank you for attending this session on iLogic and the Inventor API. I hope you found the class enjoyable and valuable. iLogic and the Inventor API will allow you to create custom functionality that can save your company time by automating tasks that would otherwise be tedious and repetitive. In this handout you have seen what iLogic and the Inventor API are and how they can be used, either independently or together. Having multiple solutions to choose from is always a good thing and it is great to have iLogic and the Inventor API available.

Wayne Brill
Developer Technical Services Autodesk
wayne.brill@autodesk.com

Additional resources

A good starting point for all Inventor API is the resources listed on the Inventor Developer Center: www.autodesk.com/developinventor.

These include:

Training material, recorded presentations, and Add-ins Wizards.

My First Inventor Plug-in, a self-paced tutorial project for newbie's and power users who want to get up and running with Inventor programming:

<http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=17324828>

Information on joining the Autodesk Developer Network: www.autodesk.com/joinadn

Information on training classes and webcasts: www.autodesk.com/apitraining

Links to the Autodesk discussion groups: www.autodesk.com/discussion.

You will also find there many VB.Net and C# samples included in the Inventor API SDK installation.

Brian Ekins' blog, the Inventor API main designer: <http://modthemachine.typepad.com>.

If you're an ADN partner, there is a wealth of Autodesk API information on the members-only ADN website: <http://adn.autodesk.com>

ADN members can ask unlimited API questions through our DevHelp Online interface

Watch out for our regular ADN DevLab events. DevLab is a programmers' workshop (free to ADN and non-ADN members) where you can come and discuss your Autodesk programming problems with the ADN DevTech team.