

IM469414

# Drawing Automation with API and New iLogic Snippets in Inventor 2021

Sergio Duran  
Manufacturing Technical Consultant  
[sduran@advconsulting.co](mailto:sduran@advconsulting.co)

## Learning Objectives

- Learn how to prepare a 3D model before automating a 2D drawing.
- Discover the new iLogic snippets to automate 2D drawings in Inventor 2021.
- Discover the differences between iLogic and Inventor API when automating 2D drawings.
- Learn how to determine the best approach to automate your drawings.

## Description

Autodesk added more iLogic snippets in the latest version of Inventor software to make drawing automation easier. This class will teach you how to prepare 3D models to easily automate the creation of drawings. Learn how to use the new iLogic snippets to automate different annotations such as dimensions, leaders, balloons, and more. In addition, you will discover when you should go beyond drawing automation capabilities with iLogic and start using the Inventor API in this process. Finally, you will identify the right approach to automate your Inventor drawings.

## Speaker(s)

Sergio Duran is a mechanical engineer and an Autodesk Certified Instructor with more than 12 years of experience working with Autodesk Manufacturing Solutions. Speaker at Autodesk University events (Las Vegas, Mexico City and the online version). Previously, he worked for two Autodesk authorized resellers and training centers as an applications specialist. Extensive experience and knowledge in CAD, design automation, simulation (CAE), visualization as well as product data management (PDM). Proven success in training students and clients on CAD, CAE and PDM applications. Sergio currently works as an independent consultant providing professional consulting, implementation, training, and support services. He assesses business issues and assists clients in design solutions, optimization and efficient workflows. Additionally, he teaches Autodesk Inventor and AutoCAD courses at Sheridan College.

## TABLE OF CONTENTS

Introduction.....	3
3D Model Preparation .....	5
Prepare the model to automate drawing views.....	5
Prepare the model to automate annotations.....	6
Working with Attributes for iLogic rules .....	7
Working with Attributes for Inventor API.....	8
Working with Workfeatures.....	11
Considerations to automate annotations .....	13
View Orientation and wireframe model .....	13
Understanding Geometry Intents.....	17
New iLogic Snippets to automate 2D drawings in Inventor 2021.....	21
Differences between iLogic and Inventor API when automating 2D drawing .....	26
Drawing Automation process using the Inventor API .....	31
Important concepts and API objects for drawing automation .....	41
Best approach to automate your drawings .....	44
Resources .....	46

## Introduction

In a project, one of the most important deliverables is a 2D drawing. It can be send and shared using different output options such as hard-copy and digital formats (DWG, PDF, DWF, etc.). To achieve this result, you need to go through multiple steps to create a 2D drawing and it can be done manually by a user, programmatically using codes or with a combination of manual inputs and automation. The last option is very common where codes complete some repetitive tasks and the user finishes what is missing in the the drawing.

Next, you can see the drawing workflow. These steps can be accomplished either using the user or programming interface.

### Drawing Workflow

1. Drawing Standards and Styles
2. Drawing Resources (Definitions)
  - Sheet Formats
  - Borders
  - Title Blocks
  - AutoCAD Blocks (only for Inventor drawing files with .dwg extension)
3. Drawing Setup
  - Create sheets
    - Use sheet formats with specified caharacteristics and standard elements:
      - Sheet Size
      - Border and title block
      - Set of standard drawing views
      - Fit Views to Sheet (new checkbox in Inventor 2021). This setting resizes and positions the drawing views to fit them inside the sheet. The drawing views use a similar size and position of the source views when the sheet format was created.
      - Other settings such as edge display, flat patterns for sheet metal parts and Parts List for assemblies are saved in the sheet format since Inventor 2021.
    - Create a blank sheet
  - Add or change a border and title block from the drawing resources
4. Referenced models
5. Drawing Views
6. Annotations
7. Output
  - Print
  - Export or save as PDF, DWG, DXF, DWF and image files
  - Reports

All these steps can be automate it, but you need to evaluate whether it makes sense or not. In some cases, automating some of them may be a waste of time and there are scenarios where is unrealistic to complete entire process programmatically. This decision depends on different factors.

The first two stages of the process are very easy to automate since they usually follow common company and international standards. In fact, most of the companies will not even think of automating them since the drawing templates, standard and styles are setup in advance and shared to all Inventor users in the company.

Drawing standards and styles are usually standardized, created in advance by a CAD manager or Inventor user, and placed in a shared location. If this is done properly, the drawing automation process can skip the creation of the drawing standards and styles.

Likewise, a standard drawing template with all the company drawing resources (sheet formats, borders, title blocks, sketch symbols and AutoCAD blocks) created and placed in a shared location will save time in the drawing automation process since the definitions of the different drawing resources is not necessary. Adding the drawing resources in the sheet is part of the third stage of the drawing workflow.

The next two steps in the drawing process are simple and can be automated easily. Working with sheets, adding drawing resources and managing the referenced models (set, get or replace) is a common procedure for most of the projects. You can reuse lines of codes to save time.

Drawing views and annotations are the most difficult phases of the automation process and require careful planning. They depend on the models to be used and the type of drawing to be created. These two stages will determine if the drawing automation can be completed with a single button/click. Complex scenarios are unlikely to be automated entirely, but you can accomplish part of this stage programmatically and the rest with some user inputs.

The last step in the workflows is the output. It is the way to present the 2D drawing printed on paper or using a digital format. Regardless of the output option you want to use, this step is usually common and standard for all company projects, then it can be automated. It is recommended to use either external iLogic rules or even better add-ins.

Clearly, the drawing automation is not a simple process. It requires Inventor knowledge, some programming experience, and understanding of the drawing workflow, referenced models and the outcome. For example, catalog-based models and simple drawings like approval drawings (drawing for quotes) are doable and can be completed programmatically without any user interaction. However, a more complex scenario like a manufacturing drawing of a customer-based model may be unachievable to create with a single click.

## 3D Model Preparation

In the drawing automation process, drawing views and annotations require careful planning to successfully automate these two steps.

A drawing view references a model and annotations like dimensions, leader notes, balloons and more are attached to drawing entities such as drawing curves, centerlines and centermarks. Drawing curves are the representation of 3D model faces and edges.

### Prepare the model to automate drawing views

If the automation is using an existing drawing where the drawing views are already placed and the code only repositions and resizes with the scale factor, they do not need any change of the 3D model views. iLogic snippets allow to reposition and resize drawing views. However, if the drawing automation requires adding views, it is important to be familiar with the model and set 3D model views properly by using the ViewCube.

In your 3D model, right click on the ViewCube > Set Current View as > Front

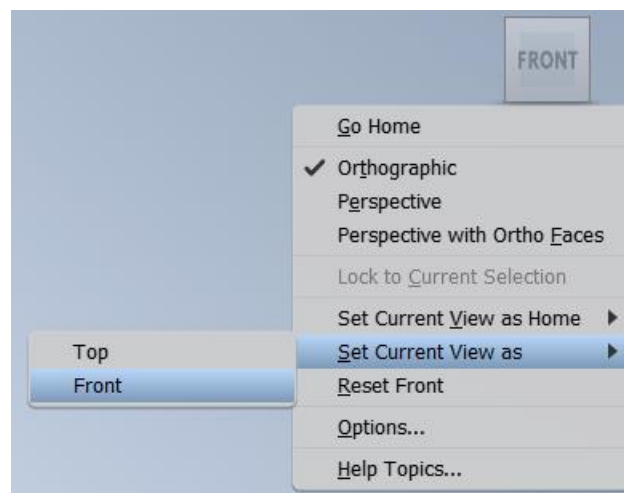


Image 1 – Set the Front view in the 3D model

The iLogic library does not have any snippet to add drawing views, however Inventor API has the collection object **DrawingViews** which allows the user to create the drawing views programmatically. Next, the image shows the drawing views that can be added using Inventor API and it highlights the most frequently used views: base, projected, section, detail and auxiliary.

# DrawingViews Object

## Methods

Name
AddAssociativeDraftView
AddAuxiliaryView
AddBaseView
AddDetailView
AddDraftView
AddOverlayView
AddProjectedView
AddSectionView

Image 2 – Methods to add drawing views using Inventor API

The orientation of the 3D model is used in the DrawingView method to create drawing views:

**DrawingViews.AddBaseView**( Model As Document, Position As Point2d, Scale As Double, **ViewOrientation As ViewOrientationTypeEnum**, ViewStyle As DrawingViewStyleEnum, [ModelViewName] As String, [ArbitraryCamera] As Variant, [AdditionalOptions] As Variant ) As DrawingView

The orientation of the model within the drawing view can use any value from the list **ViewOrientationTypeEnum**. For instance, the right view from your 3D model (Right view in the ViewCube) is placed using the value kRightViewOrientation or 10755.

## Prepare the model to automate annotations

There are two ways to place annotations:

1. Specifying a position on the sheet by using 2D coordinates (x, y). Notes and Parts Lists are drawing annotations that only need coordinates.
2. Using drawing entities to attach the annotation and a point on the sheet to place the annotation. Dimensions and Leader Notes require these two conditions. The drawing entities where annotations are attach to may have relationships with the 3D model.

When an annotation is attached to a drawing curve, the drawing curve is the representation of a model face or edge. Additionally, these type of annotations can also be attached to centermarks and centerlines. These two objects represent either cylindrical geometry (holes, revolved and extruded rounded shapes) or workfeatures.

The stage of annotations in the drawing automation process requires some preparation in the 3D model to easily identify where the annotations will be attached.

### Working with Attributes for iLogic rules

Define attributes for faces and edges to identify the drawing curves that represent them. If iLogic snippets will be used in the drawing automation, you should use the feature Assign Name that let you assign name to faces and edges.

In the Part environment, Right click on a face or edge > Assign Name > enter a meaningful name.

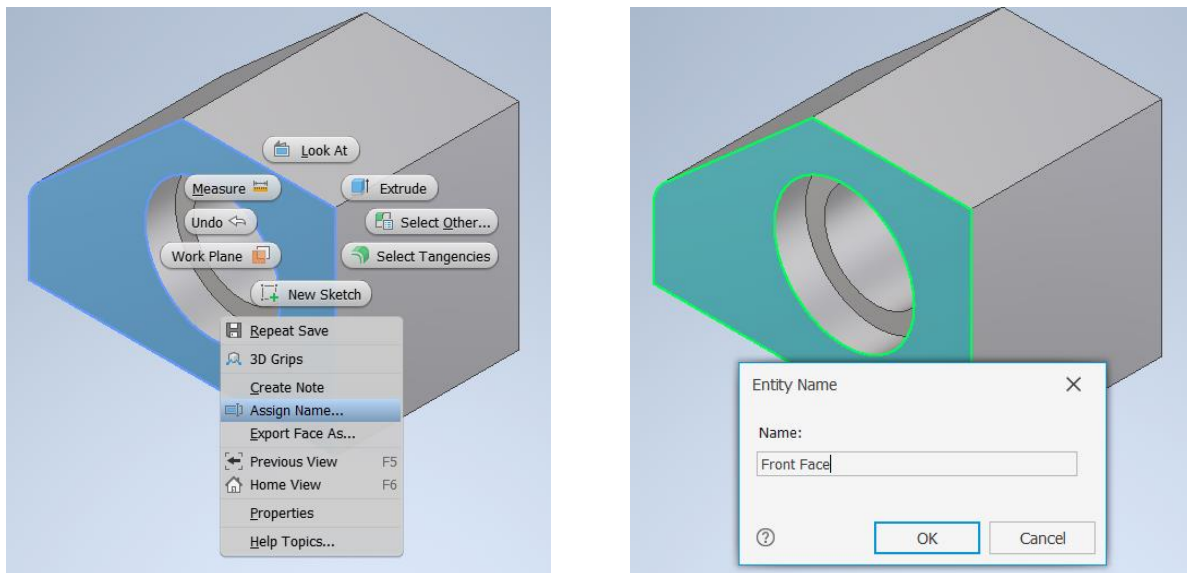


Image 3 – Assign Name to a part face

When the first attribute is defined, the iLogic browser adds a new tab named *Geometry* next to the Rules tab. All entities with an assigned name (faces, edges and vertices) are displayed under a folder named Geometry (see image below).

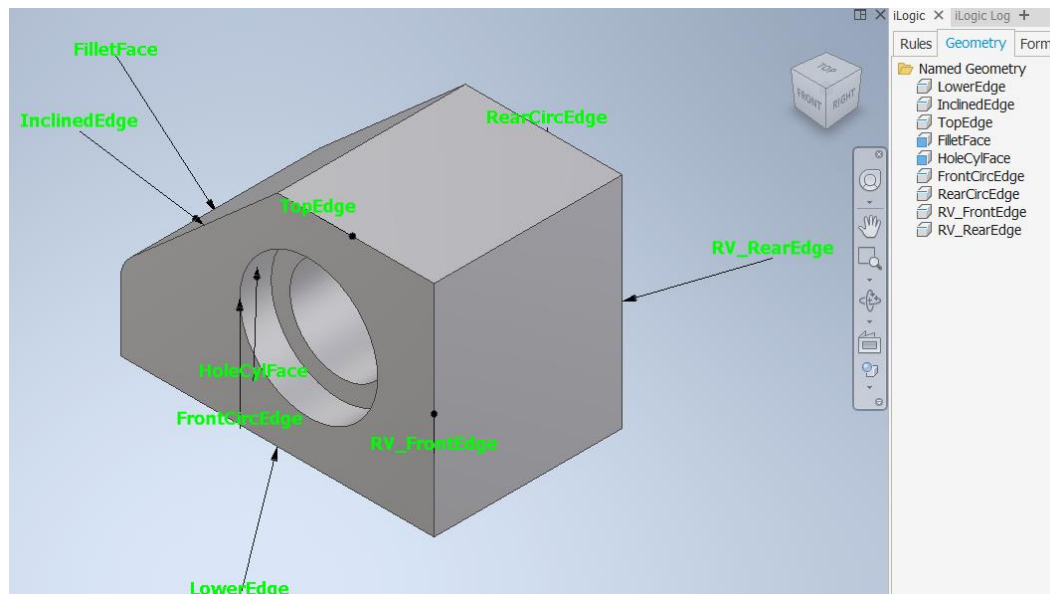


Image 4 – Geometry tab in the iLogic browser

Right click menus over the labels, browser, faces and edges allow to control label visibility and manage the names (edit, rename and delete).

Watch this demo to learn about assigning names to faces, edges and vertices.

<https://autode.sk/31Axr2N>

## Working with Attributes for Inventor API

Let's take a look at the Attribute definition and its structure in the Inventor Object Model before describing three methods to create attributes to be used with Inventor API.

An attribute allows to associate information or add metadata to Inventor entities that support attributes. The attribute functionality was only available through the API before Inventor 2020. Now, attributes can be defined using the user and programming interface, although they have some limitations when being defined with the new feature *Assign Name* of the user interface. An attribute does not define anything in the 3D model. It is useful to be used as a tool for different purposes. One of them is to find geometry (e.g. faces and edges) programmatically. Occurrences, features, sketches, parameters and more 3D model entities have names by default, but faces and edges do not have names. Attribute definition is an extra step to achieve the result of naming faces and edges.

When faces and edges have names, they can be found, referenced and used programmatically. You can name faces and edges to place assembly constraints, place drawing annotations and more.

In the Inventor API Object Model, these are the attribute objects and its hierarchy (see image below).

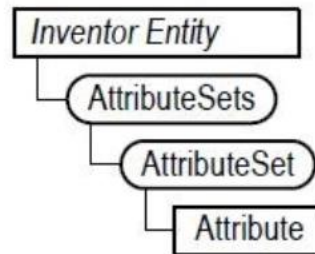


Image 5 – Object model for Attributes

The required structure to work with Attributes is very similar to the object model for iProperties. If you have worked with iProperties using the Inventor API object model, then you will easily understand how to use attributes.

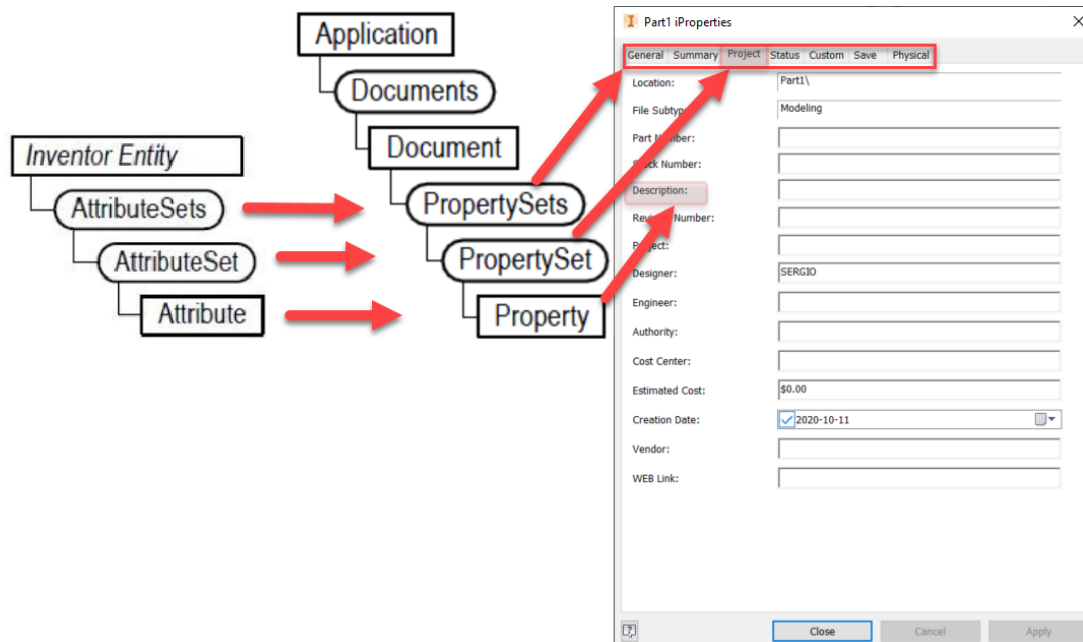


Image 6 – Comparison of object model for Attributes and iProperties

While Property Sets are under a Document, the attribute sets are under an Inventor entity that supports attributes. For instance, faces and edges are objects that support attributes. The Attribute Sets object contains all the attribute set objects. An attribute set contains attributes. An attribute has a name and a value.

If Inventor API is used for the automation of annotations, then you can use one of these three options to define the attributes:

## 1. Create attributes from scratch using a code

Next, there is a code that creates an AttributeSet named “General”, an Attribute named “Name” and assigns a value “Top Face”

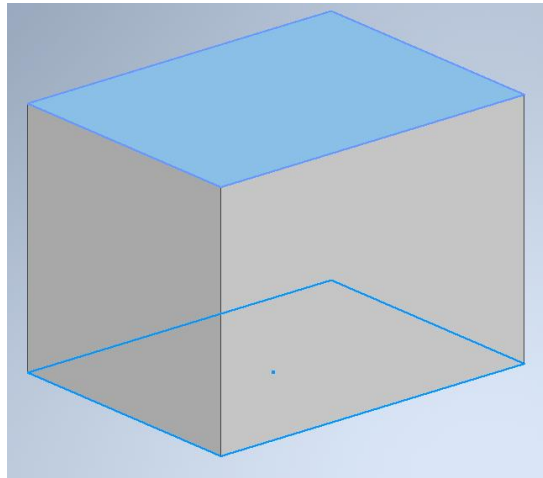


Image 7 – Face with attribute

Creating an attribute: create a box (draw a rectangle and extrude it upwards). The end face is the top face. Copy and paste this code inside an iLogic rule. Run the rule.

Note: This is a VB.NET code that also works when creating an add-in. You should only replace “ThisApplication” with “g\_inventorApplication”

```
'Get the part document; use "g_inventorApplication" instead of
"ThisApplication" to be use in an add-in
Dim oDoc As PartDocument = ThisApplication.ActiveDocument
'Get the first extrusion
Dim oExt As ExtrudeFeature = oDoc.ComponentDefinition.Features(1)
'Get the end faces (There is only one)
Dim oEndFaces As Faces = oExt.EndFaces
Dim oEndFace As Face = oEndFaces(1)

'Define an attribute set
Dim oAttSet As AttributeSet = oEndFace.AttributeSets.Add("General")
'Define an attribute and assigns a value
Dim oAtt As Attribute = oAttSet.Add("Name", kStringType, "Top Face")
```

If you want to know more about attributes, read Introduction to Attributes from the Autodesk blog Mod the Machine.

[https://modthemachine.typepad.com/my\\_weblog/2009/07/introduction-to-attributes.html](https://modthemachine.typepad.com/my_weblog/2009/07/introduction-to-attributes.html)

## 2. Use the Assign Name command in the context menu

This method was explained previously. Right click on faces and edges to assign names.

Inventor always uses the same AttributeSet name and Attribute name when this method is used. For all assigned names of faces and edges, the AttributeSet name is ***iLogicEntityNameSet*** and the Attribute name is ***iLogicEntityName***. These are the two default names to be used when searching for attributes.

## 3. Use a free Add-In provided by Autodesk and Brian Ekins to manage attributes.

You can install an add-in to manage the attributes to be used in the Inventor API. You can download it from the Autodesk blog Mod the Machine

[https://modthemachine.typepad.com/my\\_weblog/2015/08/attribute-helper-update.html](https://modthemachine.typepad.com/my_weblog/2015/08/attribute-helper-update.html)

This add-in named the Attribute Helper was created by Brian Ekins the original designer of the Autodesk Inventor® programming interface (API). In the previous link, the latest version found is “2.4”. Brian released a newer version on his website. The enhanced add-in is named Nifty Attributes version 3.0. You can read the instructions and download it from this link:

[https://ekinssolutions.com/nifty\\_attributes/](https://ekinssolutions.com/nifty_attributes/)

Watch this demo to learn how create attributes using these three methods:

<https://autode.sk/37xyd4p>

## Working with Workfeatures

Workfeatures is another way to attach annotations with information coming from the 3D model. Create workfeatures (work planes, work axes and work points) in locations of your model that you can get from your drawing views. You can also use the origin workfeatures in your drawing if they are located at points that you can use to place annotations. Get strategic points from your model to attach annotations in the drawing by using origin and user workfeatures.

Sometimes the plan to automate the annotations is a process going back and forth between the 3D and 2D environments. This means that you may have the set of drawing views already placed in the 2D drawing, think of the annotations you want to place, then go back to your model and check if you can use existing workfeatures or need to create new ones.

The three workfeatures Work Planes, Work Axes and Work Points can be used to place annotations, however, Work Points are more convenient since they are easier to use and are very handy.

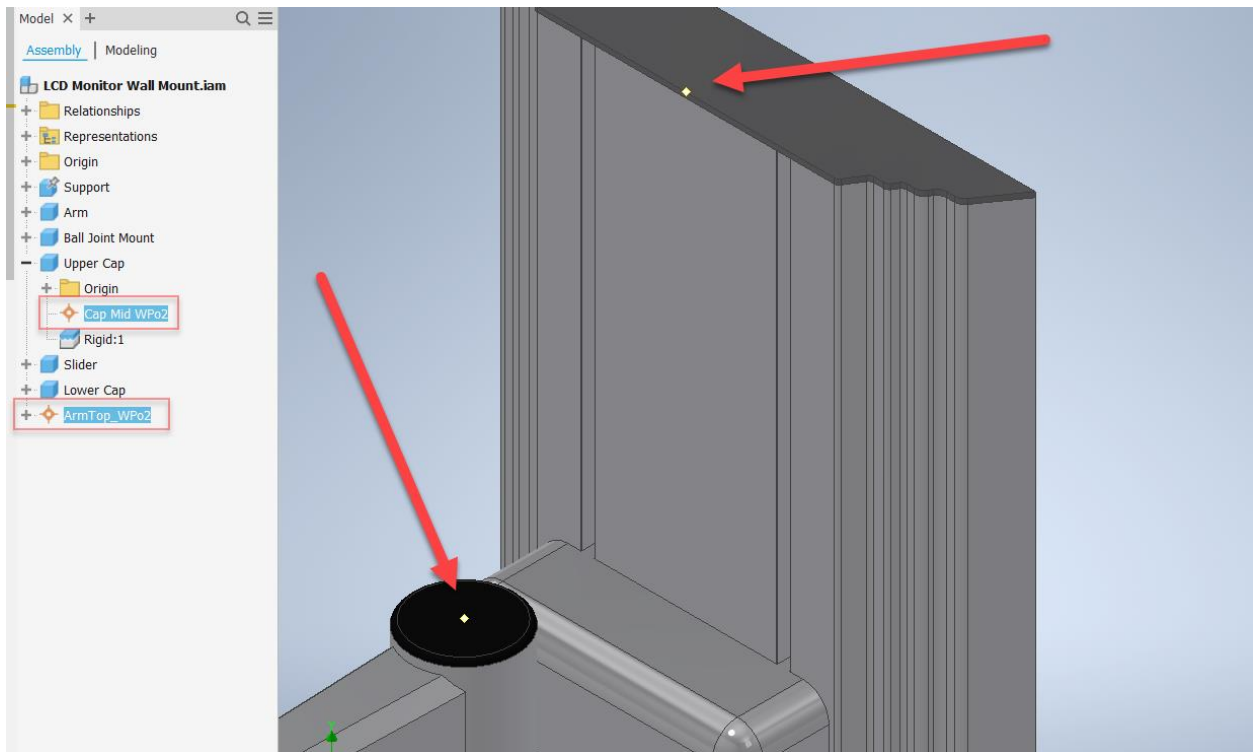


Image 8 – Creation of Work Points

In a coming section of this handout, the use of workfeatures for annotations will be described.

While iLogic snippets can directly work with workfeatures when defining the GeometryIntent, Inventor API requires the conversion of workfeatures into centermarks or centerlines and then declare the GeometryIntent to be used for annotations. GeometryIntent is also an object and concept that will be explained later.

In the drawing environment, Work Points will always be converted into centermarks. A Work Axis can be either a centermark when the axis is normal to the model orientation in the drawing view or a centerline when the axis is parallel to the model orientation in the drawing view. You can only use a Work Plane when it is displayed as a line in the 2D view and is converted into a centerline. Working with centerlines to attach dimensions is doable but it may require to specify either the start point, end point or any other point that works as the *intent* to attach the dimension. This is the reason why working with work points is easier, faster and handier.

Renaming work features is one of the best practices when using them to automate annotations. The new iLogic snippets in Inventor 2021 use the work feature name in the model when declaring the geometry intent that is used to create the annotation.

Geometry Intent object

```
Dim namedGeometry1 = VIEW.GetIntent("entityName")
entityName: name of the model geometry (face, edge, vertex or workfeature)
```

Inventor API can use either the workfeature name or a number that indicates the index of the workfeature in the 3D model browser. Rename the workfeatures or sort them in the model browser before you use them in the codes to automate the annotations. Next, you can see the API property to return a Work Point from the collection object WorkPoints. You can enter either the number that specifies the work point to return from the work points in the 3D model browser or its name.

*WorkPoints.Item( Index As Variant ) As WorkPoint*

Index Variant Input Variant value that specifies the object to return. This can be a **numeric value** indicating the index of the item in the collection or it can be a string indicating the **work point name**.

## Considerations to automate annotations

### View Orientation and wireframe model

When you place a drawing view, some model edges are converted into drawing curves while others disappeared. This behavior depends on the orientation of the model used in the 2D drawing view and the display style. The Inventor drawing environment uses a 3D wireframe model and project it into a 2D sheet space. Let's take a look at the following example. This 3D model is a simple shape with a fillet and a hole. According to the view orientation of the 3D model, the front view is the one where the counterbore hole was placed. In the front view of the 2D drawing, the right vertical edge of the front view will be displayed, but the rear vertical edge will disappear since these two edges are collinear and the rear edge is behind the front edge.

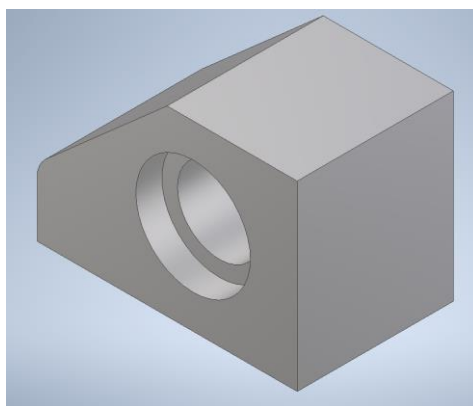


Image 9 – 3D model

By default, Inventor uses a 2D view orientation in the drawing environment. Therefore, the user does not see what happens behind the drawing. With this view orientation, the user cannot see the wireframe model behind the sheet, then may not know how Inventor displays the 3D model in a 2D drawing.

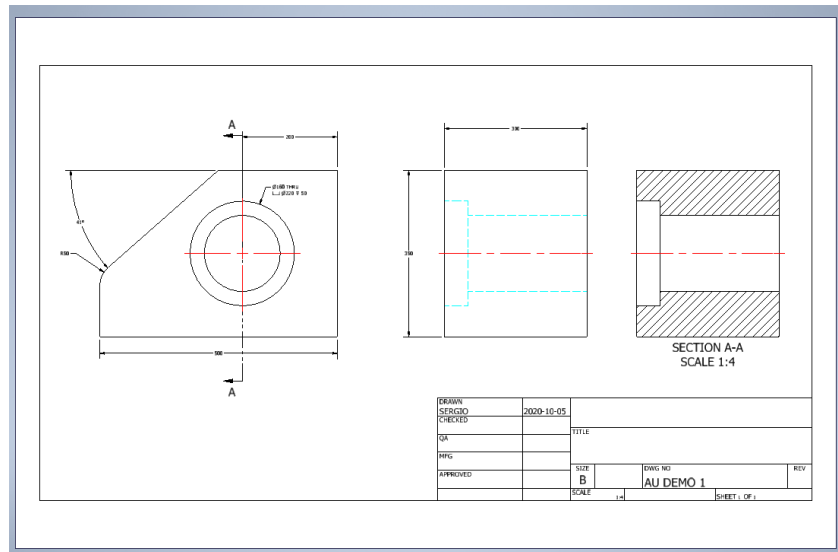


Image 10 – 2D Drawing

Inventor uses a 3D wireframe model that only has the required model edges to display the drawing curves of a drawing view. In the next image, all the rear edges of the 3D model (edges of the back face) disappeared in the 2D front view (view on the left in the sheet that uses the projection of the top wireframe). The rear edges are not needed since they are collinear in an orientation normal to the front view of the model. Inventor only needs the edges of the front view and two circular edges from the counterbore hole to properly display the drawing curves of the drawing view.

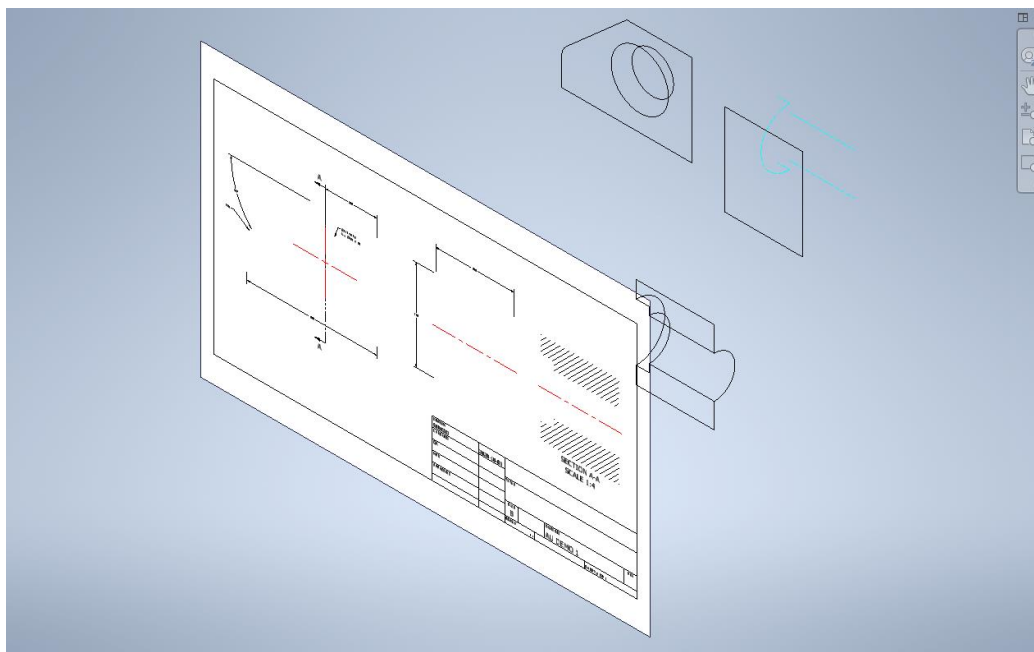


Image 11 – 2D Drawing with an isometric camera orientation

In the previous image, the drawing view style displays hidden lines. The side view (view in the middle of the sheet created with the projection of the top right wireframe) displays the hole with hidden lines (cyan lines). If the drawing view style is changed to Hidden Line Removed, the wireframe removes these hidden lines and they are not displayed.

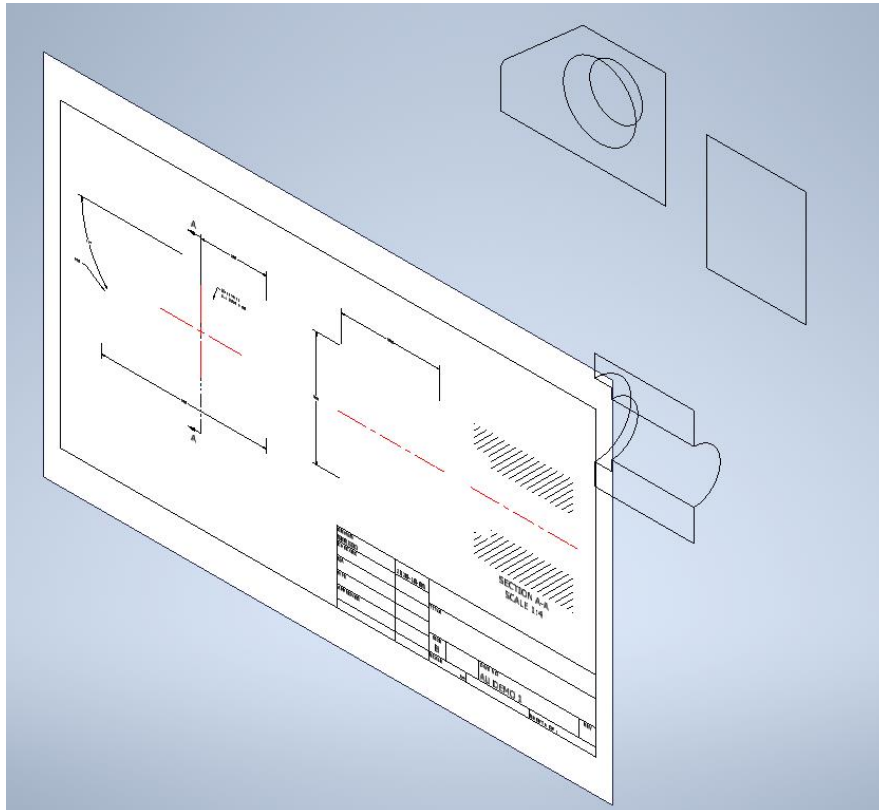


Image 12 – Isometric view orientation displaying drawing views with hidden lines removed

An Inventor 2D drawings view is just the projection of a 3D wireframe model on a sheet.

It is important to understand how Inventor creates the drawing, because some edges may be displayed in some drawing views but disappear or are hidden in others. In this example, if you need to attach an annotation to the external circular edge of the counterbore hole, you can only do it in the section view because it is not displayed in the side view regardless of the drawing view style (Hidden Line and Hidden Line Removed).

You should consider how some edges are used in some drawing views and are not displayed in others when creating attributes for edges and faces.

Use the following code to change the drawing view orientation. This code only sets two camera orientations: Front and Isometric. It uses VB.NET language, then it can be used in an iLogic rule or an add-in.

```

'***This code changes document view by using a camera orientation

'[ Gets or creates a multi-value parameter with two text values (Front and
Isometric)
Dim oUserParams As UserParameters =
ThisApplication.ActiveDocument.Parameters.UserParameters
Dim oCameraParam As Inventor.Parameter
Dim oCamList(1) As String
oCamList(0) = ""Front""
oCamList(1) = ""Isometric""

Try
    oCameraParam = oUserParams.Item("CameraOrientation")
Catch
    oCameraParam = oUserParams.AddByValue("CameraOrientation", "",
kTextUnits)
Finally
    oCameraParam.ExpressionList.SetExpressionList(oCamList)
End Try
']

'Gets the active view of the document within the current window
Dim oDocView As View = ThisApplication.ActiveView
Dim oCamera As Camera = oDocView.Camera

'Sets the type of orientation of the camera (Front or Iso)
If oCameraParam.Value = "Front"
    oCamera.ViewOrientationType =
ViewOrientationTypeEnum.kFrontViewOrientation
Else 'Isometric
    oCamera.ViewOrientationType =
ViewOrientationTypeEnum.kIsoTopRightViewOrientation
End If

'Apply the current camera definition to the view
oCamera.ApplyWithoutTransition

```

The first part of the code only gets or creates a multi-value parameter with two options: Front and Isometric. The second part of the code changes the view orientation and is pretty straightforward. Get the camera of the current view, define the desired view orientation and then apply the method to change the orientation of the view.

Watch this demo to understand how Inventor creates a 2D drawing using the wireframe model  
<https://autode.sk/37yRWAA>

## Understanding Geometry Intents

A Geometry Intent is an Inventor API object with two components, the geometry and the intent. The geometry is the drawing geometry being used to attach the annotation and the intent is the precise point to attach it. The geometry is mandatory while the intent depends on the case and sometimes there is no need to define an intent. A good example is when using centermarks. A centermark does not need an intent, since the definition of the geometry intent with the centermark is enough. However, a centerline has a start point, end point or any point using a relative value (from 0 to 1), then the geometry is the centerline and the intent may be a specific point to attach the annotation precisely.

When using Inventor API, the geometry can be: **drawing curve**, sketch entities from a sheet sketch, drawing dimension, **centerline and centermark**. The three highlighted drawing entities are more common options.

In the following image, the 3D model on the left has two work points to be used in the annotations automation. The drawing on the right displays a linear dimension 85 mm. This dimension is attached to two centermarks that were created from the 3D model work points. The centermarks are highlighted on the right (they were enlarged for this example).

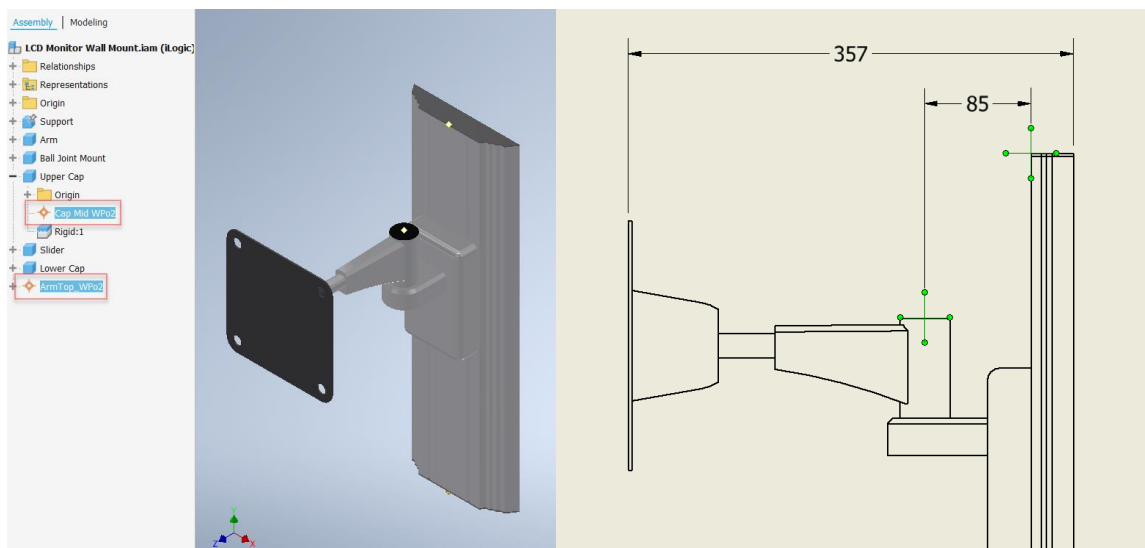


Image 13 – Model with work points and a dimension attached to centermarks

The next Inventor API code to place this linear dimension demonstrates how the centermarks were created from work points (*Centermarks.AddByWorkFeature*). Then, the geometry intents were created from the centermarks, but they only needed the geometry and there is no intent.

```
Dim oCMark1 As Centermark = oCenterMarks.AddByWorkFeature(oArmTopWPo, oSideView)
Dim oCMark2 As Centermark = oCenterMarks.AddByWorkFeature(oCapMidWPoProxy, oSideView)

oCMark1.Style = oCMStyleForDims
oCMark2.Style = oCMStyleForDims

oCMark1.Visible = False
oCMark2.Visible = False

Dim oGI1 As GeometryIntent = oSheet.CreateGeometryIntent(oCMark1)
Dim oGI2 As GeometryIntent = oSheet.CreateGeometryIntent(oCMark2)

Dim oDimPos As Point2d = oTG.CreatePoint2d((oCMark1.Position.X+oCMark2.Position.X)/2 , oCMark2.Position.Y + 1)

Call oGenDims.AddLinear(oDimPos, oGI1, oGI2, kHorizontalDimensionType)
```

Image 14 – Code to create centermarks and geometry intents to place a linear dimension

In the following method, the intent is an optional argument

*Sheet.CreateGeometryIntent( Geometry As Object, [Intent] As Variant ) As GeometryIntent*

Intent: Intent point on the input geometry. These are the different intent options where the highlighted intents are the most frequently used options.

- **PointIntentEnum value:** kMidPointIntent, kEndPointIntent, and more.
- A geometry if the intent is the intersection of two geometries.
- **Point2d** object that specifies a sheet point on the geometry.
- **Double value** (0 to 1) indicating the parameter on the input curve geometry.

In the next example, the intent point is required to place the annotation precisely. The bottom face of the 3D model is displayed as a drawing curve. The drawing curve is obtained by finding the bottom face that represents it. The geometry intent is created using this drawing curve as the geometry and an intent point. The intent used in this case is a *double value*. In this scenario more intent types can be used, but this example only demonstrates the double value which is an intent type named parameter intent (kParameterIntent). The image shows three positions of the same leader note using 0.25, 0.5 and 0.75 in the intent argument. The left corner is the double value zero and the right corner is one. Any value from zero to one goes from left to right on this drawing curve.

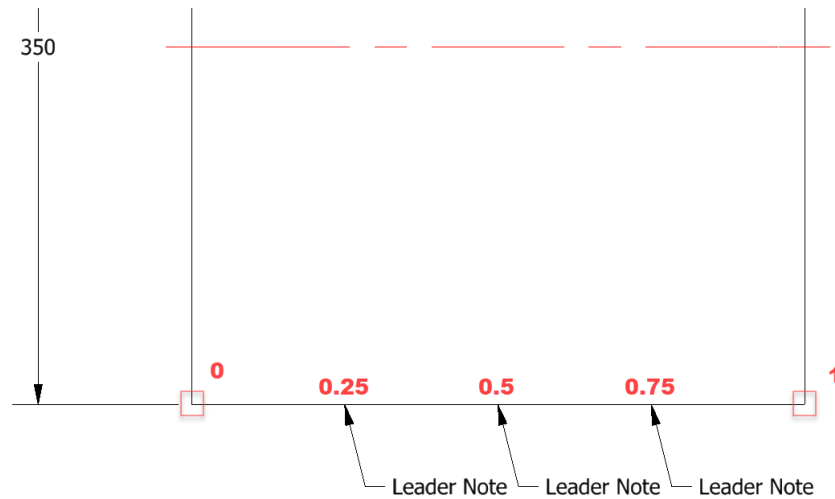


Image 15 – Leader notes placed on the lower drawing curve of a drawing view

Below, the image displays the last lines of the API code used to place the leader note. Notice how the intent *0.75* which is a parameter intent type is used to place the leader precisely near the right corner of the edge

```
oDrawingCurves = oSideView.DrawingCurves(oLowerFace)
Dim oLowerCurve As DrawingCurve = oDrawingCurves(1)

Dim oLowerCurveGI As GeometryIntent = oSheet.CreateGeometryIntent(oLowerCurve, 0.75)
Dim oLeaders As LeaderNotes = oSheet.DrawingNotes.LeaderNotes

Dim oLeaderPointsColl As ObjectCollection = ThisApplication.TransientObjects.CreateObjectCollection
Dim oLeaderPos As Point2d = oTG.CreatePoint2d(oLowerCurveGI.PointOnSheet.X + 0.25, oLowerCurveGI.PointOnSheet.Y - 1)

oLeaderPointsColl.Add(oLeaderPos)
oLeaderPointsColl.Add(oLowerCurveGI)

Call oLeaders.Add(oLeaderPointsColl, "Leader Note")
```

Image 16 – Creation of a geometry intent using a drawing curve and double value to place a leader note

Now, the Geometry Intent can be used in iLogic with some limitations.

The geometry can only use a face, edge, vertex or workfeature from the 3D model.

```
Dim namedGeometry1 = VIEW.GetIntent("entityName")
entityName: name of the model geometry (face, edge, vertex or workfeature)
```

The intent only works with the PointIntentEnum type.

```
Dim namedGeometry1 = VIEW1.GetIntent("NamedGeometry1", )
```

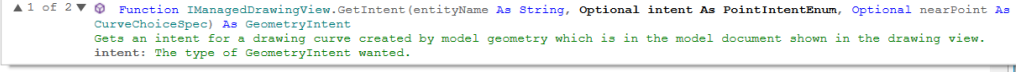


Image 17 – Optional intent in the iLogic DrawingView.GetIntent method

```
VIEW1.GetIntent("NamedGeometry1", PointIntentEnum.)
```

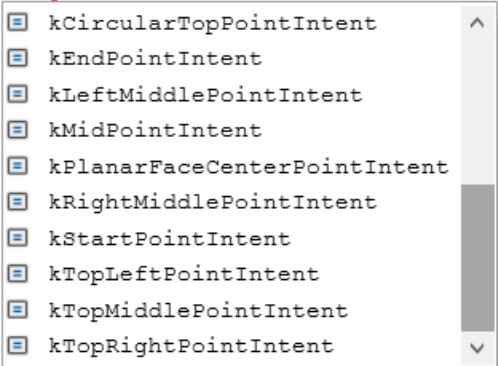


Image 18 – Intellisense to select one of the PointIntent values

If you need to use another type of intent, then you should use the GeometryIntent object of the Inventor API.

The definition of the GeometryIntent objects are different in the Inventor API and iLogic libraries. When you create a GeometryIntent using Inventor API, this is a method of the sheet object. The geometry object used in this method (e.g. a drawing curve or centermark) was defined by using the drawing view object as shown in the two previous API codes. The iLogic is a drawing view method as shown in the image above.

## New iLogic Snippets to automate 2D drawings in Inventor 2021

Inventor 2021 added new snippets to add annotations automatically. iLogic snippets allow to add the following dimensions: linear, angles, radius and diameters. They also let you automate the creation of centermarks, centerlines, centered pattern, leader notes, hole notes and balloons.

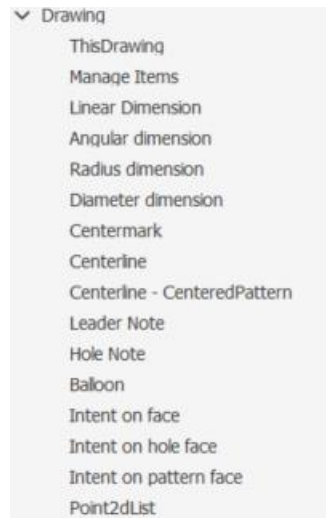


Image 19 – New Inventor 2021 iLogic Snippets for annotations automation

When you click on a new iLogic snippet to automatically place an annotation, you only need to replace values and arguments. If you click on the Linear Dimension snippet, this is what you get.

```
Dim Sheet_1 = ThisDrawing.Sheets.ItemByName("Sheet:1")
Dim VIEW1 = Sheet_1.DrawingViews.ItemByName("VIEW1")
Dim namedGeometry1 = VIEW1.GetIntent("NamedGeometry1")
Dim genDims = Sheet_1.DrawingDimensions.GeneralDimensions
Dim linDim1 = genDims.AddLinear("Dimension 1", VIEW1.SheetPoint(0.5, -0.1),
namedGeometry1)
```

First, replace sheet and drawing view names in the first two lines if you have renamed them. Second, replace the “NamedGeometry1” with your named face or edge. Dimensions can be added with one or two GeometryIntents. For example, if you need to place a dimension between two geometry entities (e.g. from edge to edge), then copy the line **Dim namedGeometry1 = VIEW1.GetIntent("NamedGeometry1")** and enter a second named edge for the object in the argument, **Dim namedGeometry2 = VIEW1.GetIntent("NamedGeometry2")**. The fourth line gets the object general drawing dimensions, there is no need to make any change in this line. Finally, replace the values in the last line to place the linear dimension. The important values are the point to place the linear dimension and the GeometryIntent. The point values are any value from 0 to 1, where 0 is the bottom left corner and 1 is the top right corner of the drawing view. The GeometryIntent is the one you declared in the third line.

The other drawing annotation snippets follow a similar structure: get the sheet, get the drawing view, declare the geometry intent using named faces or edges, get the object general dimensions,

and finally create the annotation. Therefore, you delete some duplicated lines if you are creating more than one annotation. A sheet, a drawing view and the object general dimensions only need to be once in the code.

The Geometry Intents can be created multiple times since they are related to the drawing views. Let's take a look at the next example:

A face with a name "RightFace" requires different geometry intents if it will be used in multiple drawing views. A geometry intent object for the top view and another object for the front view.

```
Dim RightFaceGI_TV = TopView.GetIntent("RightFace")
Dim RightFaceGI_FV = FrontView.GetIntent("RightFace")
```

It also applies for workfeatures. If a Work Point needs to be used in two drawing views, it requires two geometry intents.

```
Dim WPo3_FV = FrontView.GetIntent("TopRight_FrontWPo3")
Dim WPo3_SV = SideView.GetIntent("TopRight_FrontWPo3")
```

When automating one or multiple annotations, you should use the second snippet from the new Drawing category in Inventor 2021 named *Manage Items*. These two lines in your code automatically remove annotations when necessary.



Image 20 – Snippet Manage Items in the Drawing category

When you click on the **Manage Items**, it adds the following lines:

```
ThisDrawing.BeginManage()
' Statements to add dimensions, annotations, etc. go here.
ThisDrawing.EndManage()
```

Delete the comment and add your annotation automation lines between the two magenta/purple lines **ThisDrawing.BeginManage** and **ThisDrawing.EndManage**. Annotations inside this block will be clean or deleted when not required. You only use methods to add annotations and there is not need to call methods to delete them since they will be deleted automatically when they are not needed.

Next, there is an example that demonstrates the annotations automation in the 2D drawing of the block with the counterbore previously displayed. The 3D model has named faces and edges.

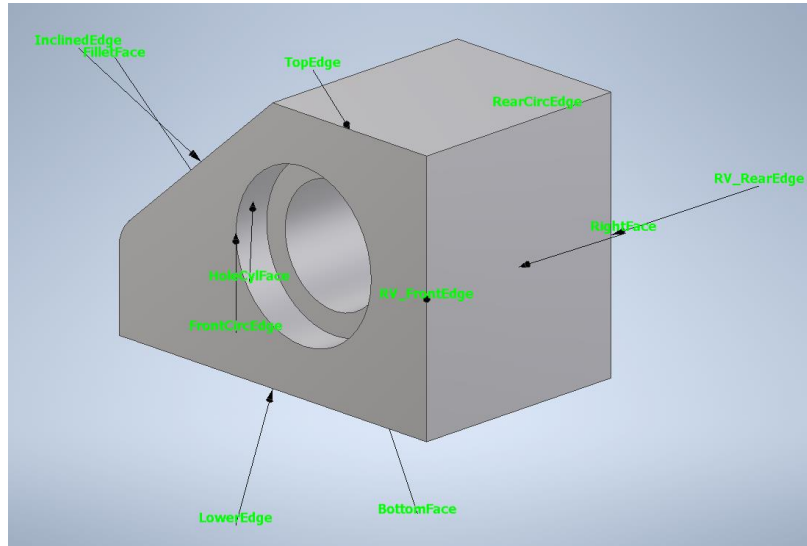


Image 21 – 3D part with named edges and faces

Below, the 2D drawing displays general dimensions, centerlines, a centermark and a hole note that were placed automatically with the new iLogic snippets.

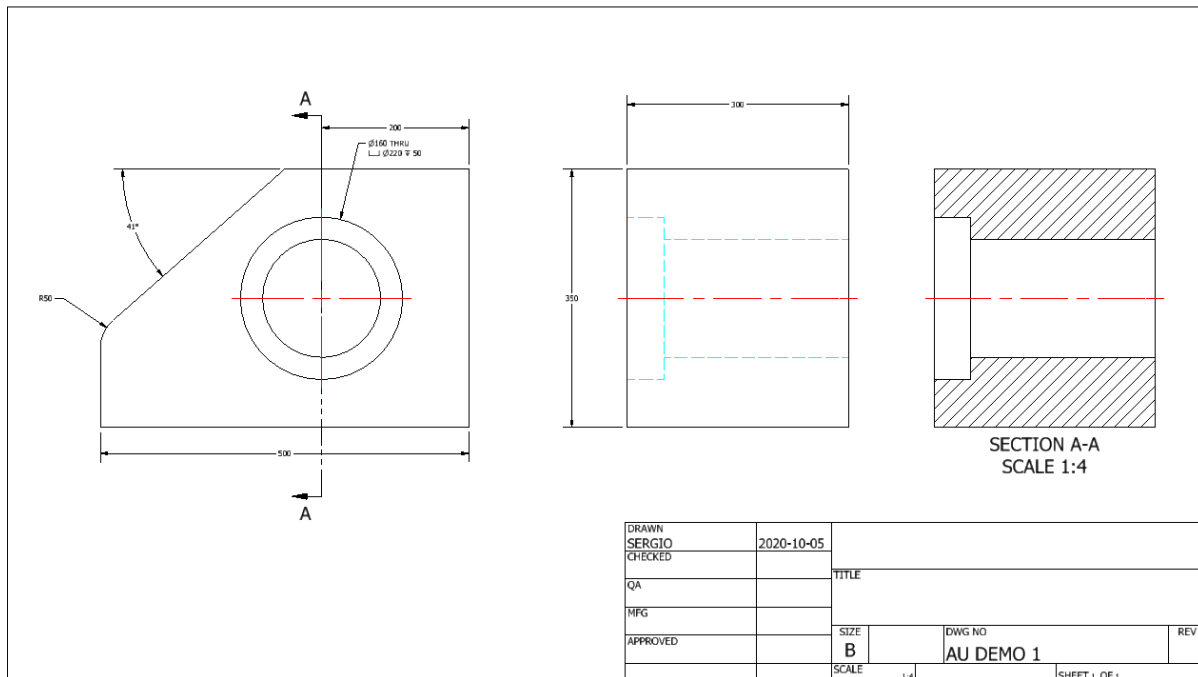


Image 22 – Automated annotations in a 2D drawing

Watch these demos to learn about automating annotations with iLogic

<https://autode.sk/3jjHT4J>

<https://autode.sk/3krVY1q>

<https://autode.sk/2FXMrjO>

This is the code that placed the annotations in the 2D Drawing.

```

'''This code places either overall annotations or all annotations

'Declare main objects: sheet, drawing views, annotations and geometry intents

'Get sheet and drawing views
Dim Sheet_1 = ThisDrawing.Sheets.ItemByName("Sheet:1")
Dim FrontView = Sheet_1.DrawingViews.ItemByName("Front")
Dim SideView = Sheet_1.DrawingViews.ItemByName("Side")
Dim SectionView = Sheet_1.DrawingViews.ItemByName("A")
'Get general dimensions and hole thread notes
Dim genDims = Sheet_1.DrawingDimensions.GeneralDimensions
Dim holeThreadNotes = Sheet_1.DrawingNotes.HoleThreadNotes

'Geometry Intents to be used on the Front View
'Geometry Edge, No Point Intent
Dim LowerEdgeGI = FrontView.GetIntent("LowerEdge")
'Geometry Cylindrical Face, No Point Intent
Dim Hole_FView As GeometryIntent
If Parameter("1-iLogic Attr_Annotations.ipt.HoleInclude") = True
    Hole_FView = FrontView.GetIntent("HoleCylFace")
End If

'GeomInt for WorkPoint3 and 4
Dim WPo2_FV = FrontView.GetIntent("Hole_TopQuad_FrontWPo2")
Dim WPo3_FV = FrontView.GetIntent("TopRight_FrontWPo3")
[']

'Geometry Intents to be used on the Side View
'GeomInt for Face
Dim RightFaceInt = SideView.GetIntent("RightFace")

Dim WPo3_SV = SideView.GetIntent("TopRight_FrontWPo3")
Dim WPo4_SV = SideView.GetIntent("TopRight_RearWPo4")
'Geometry Edge, Intent MidPoint of the Edge
Dim FrontRightEdgeInt = SideView.GetIntent("RV_FrontEdge", PointIntentEnum.kMidPointIntent)
Dim RearRightEdgeInt = SideView.GetIntent("RV_RearEdge", PointIntentEnum.kMidPointIntent)
[']

'Geometry Intents to be used on the Section View
Dim oSV_CLineInt1 As GeometryIntent
Dim oSV_CLineInt2 As GeometryIntent
If Parameter("1-iLogic Attr_Annotations.ipt.HoleInclude") = True
    oSV_CLineInt1 = SectionView.GetIntent("FrontCircEdge", PointIntentEnum.kCircularLeftPointIntent)
    oSV_CLineInt2 = SectionView.GetIntent("RearCircEdge", PointIntentEnum.kCircularLeftPointIntent)
End If
[']

'Place Annotations
ThisDrawing.BeginManage()

'Create overall dimensions
Dim LinearDim1 = genDims.AddLinear("WidthDim", FrontView.SheetPoint(0.5, -0.1), LowerEdgeGI)
Dim LinearDim2 = genDims.AddLinear("HeightDim", SideView.SheetPoint(-0.25, 0.5), RightFaceInt, , _
    DimensionTypeEnum.kVerticalDimensionType)
Dim LinearDim3 = genDims.AddLinear("DepthDim", SideView.SheetPoint(0.5, 1.25), WPo3_SV, WPo4_SV)

'Create centerlines
If Parameter("1-iLogic Attr_Annotations.ipt.HoleInclude") = True
    Dim centermark = Sheet_1.Centermarks.Add("Hole Centermark", Hole_FView)
    Dim centerline = Sheet_1.Centerlines.Add("Hole Centerline", {FrontRightEdgeInt, RearRightEdgeInt})
    Dim Section_centerline = Sheet_1.Centerlines.Add("SV_Centerline", {oSV_CLineInt1, oSV_CLineInt2})
End If

'Place detailed dimensions
If Dimensions = "All Dimensions"

'Angle
Dim angDim1 = genDims.AddAngular("AngularDim",
    ThisDrawing.Geometry.Point2d(2.5, 6.5),
    FrontView.GetIntent("InclinedEdge"), FrontView.GetIntent("TopEdge"))

'Add fillet radius when unsuppressed
If Parameter("1-iLogic Attr_Annotations.ipt.FilletInclude") = True
    Dim filletDim = genDims.AddRadius("Fillet Radius", FrontView.SheetPoint(-0.1, 0.5), FrontView.GetIntent("FilletFace"))
End If

'Add hole note and dimensions when unsuppressed
If Parameter("1-iLogic Attr_Annotations.ipt.HoleInclude") = True
    Dim holeNotePt1 = FrontView.SheetPoint(.7,1.1)
    Dim holeNote1 = holeThreadNotes.Add("Hole Note 1", holeNotePt1, Hole_FView)
    'Horizontal location of the hole
    Dim HoleHorizLoc = genDims.AddLinear("Hole Horiz Loc", FrontView.SheetPoint(0.75, 1.16), WPo2_FV, WPo3_FV, _
        DimensionTypeEnum.kHorizontalDimensionType)
End If

End If

ThisDrawing.EndManage()

```

Inventor 2021 is the first version with iLogic snippets to automate the drawing annotations. There are still some annotations and dimensioning methods such as chain, baseline and ordinate that are not possible with iLogic snippets. Next image shows a drawing of a sheet metal part where ordinate dimensions were automatically placed.

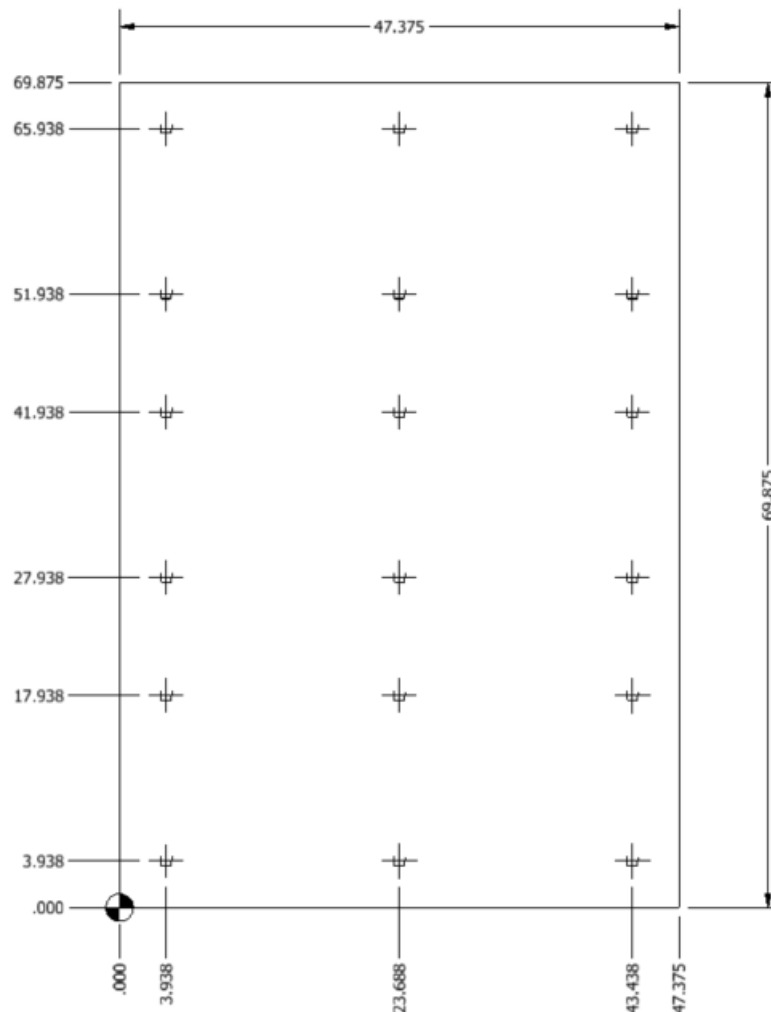


Image 23 – 2D drawing of a sheet metal part with ordinate dimensions and origin indicator

Although you cannot automate the annotations of this drawing with iLogic snippets, you can do it with Inventor API and this is a very good example to understand why you need to go beyond iLogic capabilities and start using the Inventor API.

## Differences between iLogic and Inventor API when automating 2D drawing

In this section, the iLogic drawing automation capabilities are listed. The list let you understand what iLogic can and cannot do in the drawing automation process. However, this does not mean that Inventor cannot complete a task programmatically when you discover an iLogic limitation. Go beyond iLogic and use Inventor API since it can perform almost everything you do through the user interface. Additionally, there are some cases where you want to use Inventor API anyhow since iLogic can place an annotation but it does not have access to all the options using the user interface and the API.

Let's take a look at these two examples. First, you want to add more information in the text of a dimension. When a part uses a common radius value, it is very common to add the suffix **TYP** that means *typical*. The dimension on the left can be placed using the new iLogic snippets, but if you need to override or change the dimensions text, then you need to use Inventor API to get this result (annotation on the right).

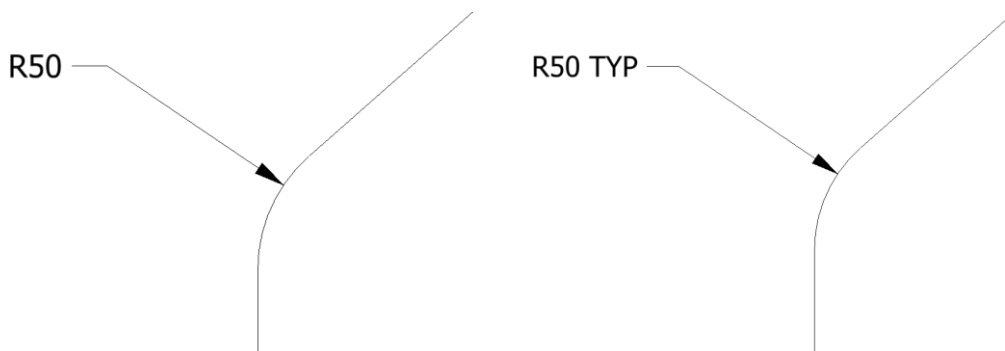


Image 24 – Default and modified radius values

Another example is about centerlines and centermarks. Now, iLogic let you automate the creation of centermarks, centerlines and centered pattern. However, currently it does not have the easiest and fastest method to place centermarks, centerlines and centered pattern at once which is the Automated Centerlines command. This command in the user interface is exposed as a method of the DrawingView object in the Inventor API.

*DrawingView.SetAutomatedCenterlineSettings( [AutomatedCenterlineSettings] As Variant )*

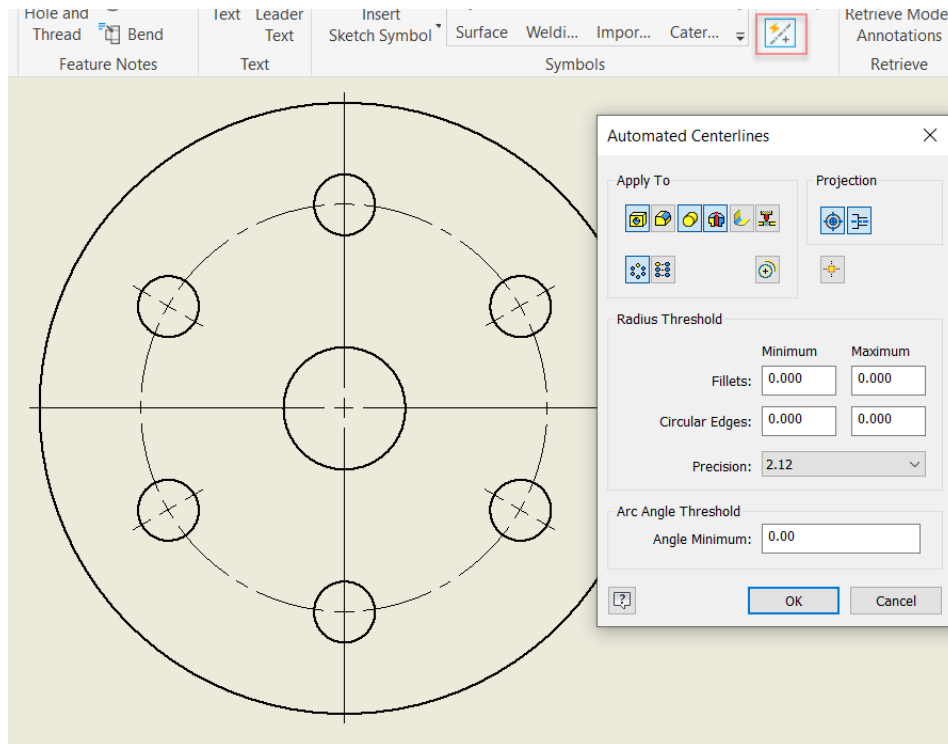


Image 25 – Command Automated Centerlines

In the drawing automation process, there are more steps where you may find limitations using iLogic like placing drawing views or creating a drawing from scratch. These are only few examples but there are more situations where you cannot achieve the result with iLogic. If you need to go beyond these limitations use Inventor API for the drawing automation process. iLogic is an Inventor add-in that only has some of the API functionality. It is built on top of the Inventor API, then whatever you do with iLogic can be done with the Inventor API and even more.

Watch this demo about using Inventor API <https://autode.sk/3mgUckq>

This section only compares iLogic and Inventor API for the drawing workflow. If you want to understand the main differences between iLogic and API regardless of the environment or you do not even know what the API is, then watch the following Autodesk University class.

### iLogic and the Inventor API

Speaker: Brian Ekins – Designer of the Inventor Application Programming Interface (API)

<https://www.autodesk.com/autodesk-university/class/iLogic-and-Inventor-API-2016>

Next, there is list of the iLogic capabilities and limitations in the drawing automation process. As mentioned before, Inventor API can perform most of the steps you follow through the user interface, then the programming interface will be able to carry out any step that iLogic cannot execute.

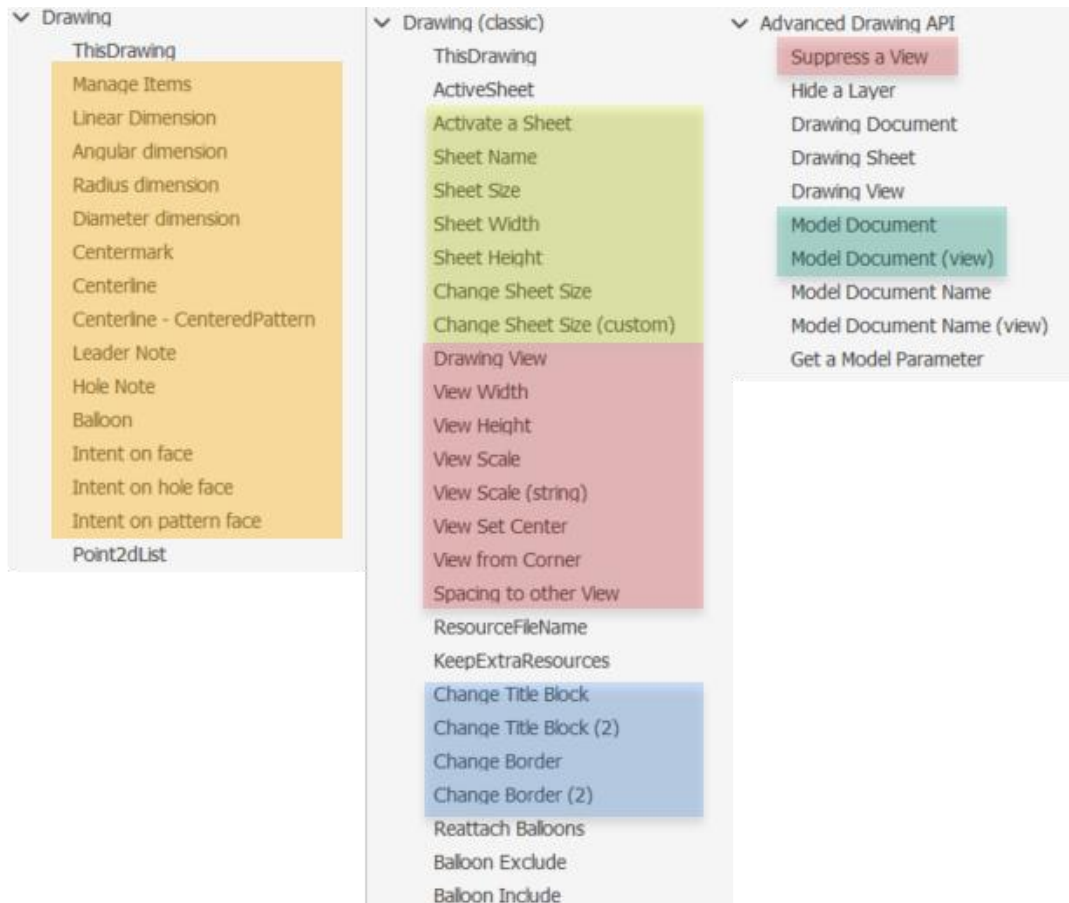


Image 26 – System snippets for drawing automation

iLogic snippets in the drawing automation process

CAN:

- Drawing Setup (yellow and blue boxes)
  - Activate a sheet
  - Rename sheets
  - Get and change sheet size
  - Get sheet dimensions (width and height)
  - Change title blocks and borders
- Referenced Models (green box)
  - Get referenced documents
    - Get first model document shown in the drawing
    - Get the model document of a drawing view

- Drawing Views (red boxes)
  - Position drawing views
  - Change scale factor
  - Suppress/unsuppress a drawing view
- Annotations (orange box)
  - Add general dimensions such as linear, angular, radius and diameter
  - Add centermarks, centerlines and centered pattern
  - Add leader notes
  - Add hole notes
  - Add balloons
  - Only use Attributes and Workfeatures from your 3D model to add annotations

### Geometry Intent

```
Dim namedGeometry1 = VIEW.GetIntent("entityName")
```

entityName: name of the model geometry (face, edge, vertex or workfeature)

### Output

Use codes from the Custom tab under Snippets. There are codes to publish DWF, export as DXF and print documents. You can modify them to suit your company needs. However, all these codes use Inventor API objects, methods and properties instead of iLogic libraries.

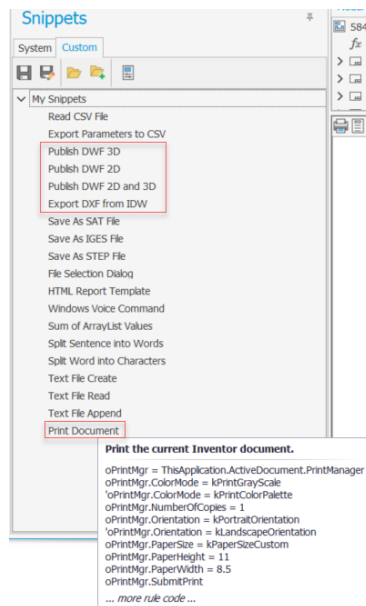


Image 27 – Custom Snippets

## CANNOT:

- Create, get, set and delete Drawing Standards and Styles
- Creates definitions of drawing resources (Sheet Formats, Borders, Title Blocks, Sketch Symbols and AutoCAD Blocks)
- Drawing Setup
  - Create Sheets (simple method “New Sheet” or using Sheet Format)
  - Delete Sheets
- Referenced Models
  - Define the referenced model when placing base views
  - Replace a referenced model
- Drawing Views
  - Add drawing views
  - Delete drawing views
- Annotations
  - Use other dimensioning methods such as baseline, ordinate and chain to place individual dimensions or multiple dimensions grouped in a set using baseline set, ordinate set and chain set. The advantage of using any of these dimensioning methods is that all dimensions are placed at once.
  - Add automated centerlines
  - Use centermarks and centerlines to define geometry intent to place annotations
  - Find drawing curve segments, entities on a sheet sketch, centerlines and centermarks by using 3D model points from the model and points 2D in the sheet to place annotations.
  - Retrieve model annotations
  - Override a dimension text
- Output
 

Create any output file. All automated output options: save as, export, print and reports require API objects, methods and properties. In addition, it is recommended to execute these automated output actions from add-ins.

## Drawing Automation process using the Inventor API

### 1. Drawing Standards and Styles

In the user interface, this is the dialog box where you create, edit and saved drawing standards and styles.

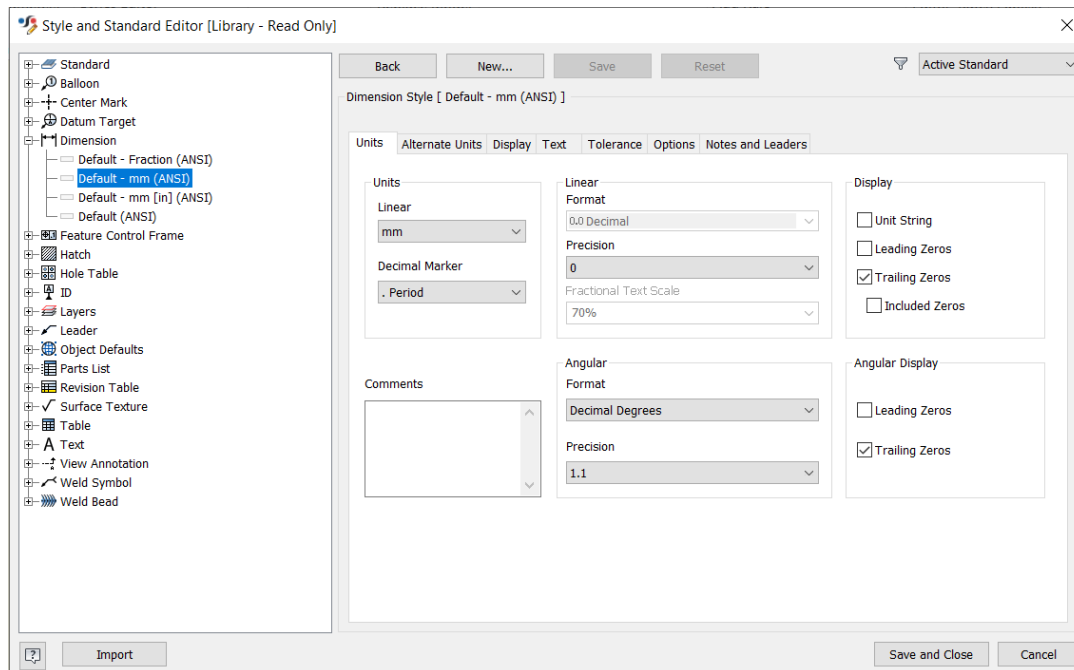


Image 28 – Style and Standard Editor dialog box

This is the section of the Inventor API object model that shows the objects to create, edit and delete drawing standard and styles using the application interface.

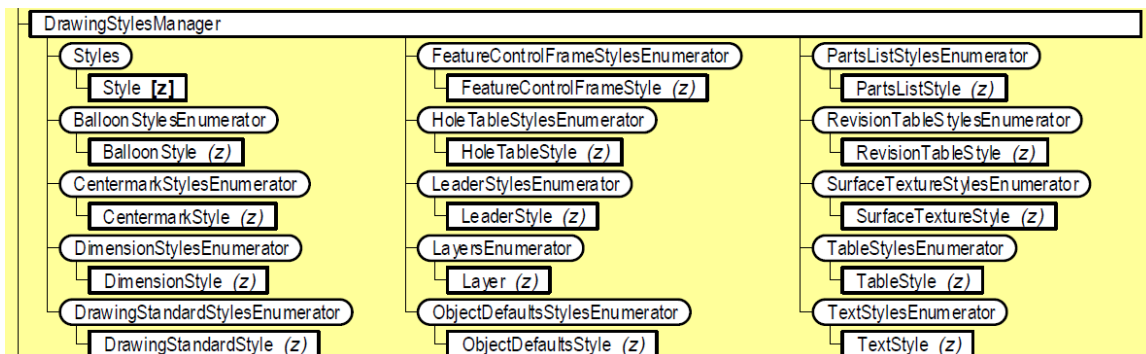


Image 29 – Inventor API objects for Drawing Styles and Standards

## 2. Drawing Resources (Definitions)

This is the drawing browser and the panel Define on the Ribbon where you can create the drawing resources: sheet formats, borders, title blocks, sketch symbols and AutoCAD Blocks (only in Inventor DWG drawings).

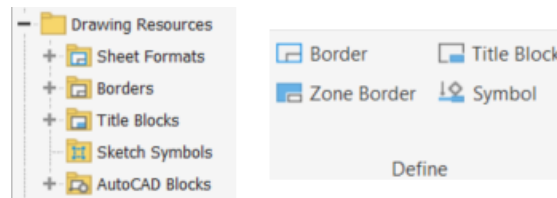


Image 30 – Browser folder and commands to create drawing resources

This is the section of the Inventor API object model that shows the objects to create, edit and delete the definition of drawing resources using the application interface.

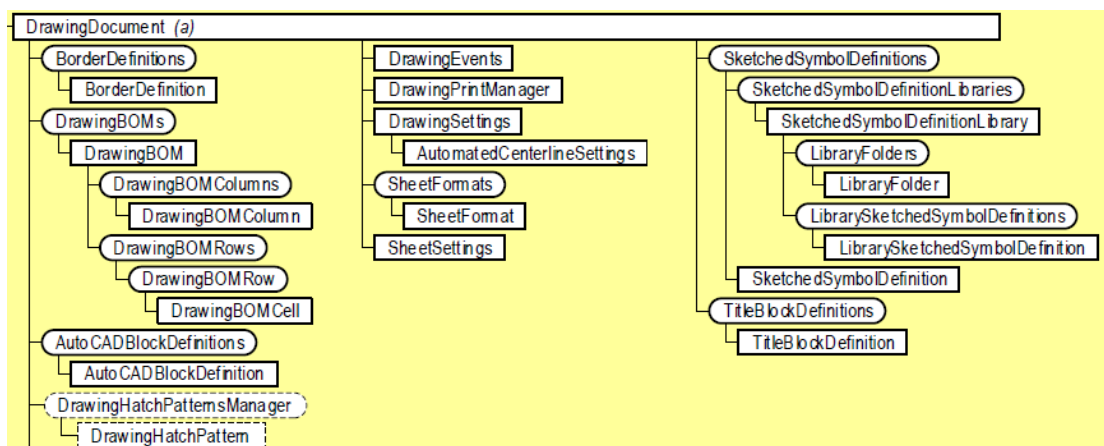


Image 31 – Inventor API objects for Drawing Resources definition

## 3. Drawing Setup

The sheets collection object has two methods to create sheets:

- **Create a blank sheet**

*Sheets.Add( [Size] As DrawingSheetSizeEnum, [Orientation] As PageOrientationTypeEnum, [SheetName] As String, [Width] As Variant, [Height] As Variant )*

You need to add the the border and title block definitions afterwards. Use these two methods to add them:

```
Sheet.AddBorder( BorderDefinition As Variant, [PromptStrings] As Variant )
```

```
Sheet.AddTitleBlock( TitleBlockDefinition As Variant, [TitleBlockLocation] As Variant,  
[PromptStrings] As Variant )
```

The only mandatory argument is the drawing resource definition. The other arguments are optional like prompt strings and the title block location.

- **Create a sheet using a Sheet Format**

This method is faster and more convenient since the border and title block can be stored in the sheet format for the drawing template. This method includes all: sheet size, border, title block, referenced model and more optional arguments

```
Sheets.AddUsingSheetFormat( SheetFormat As SheetFormat, [Model] As Variant,  
[SheetName] As String, [AdditionalOptions] As Variant, [TitleBlockPromptStrings] As  
Variant, [BorderPromptStrings] As Variant ) As Sheet
```

#### 4. Referenced Model

- **Get a referenced model**

The referenced model can be return from a drawing view. The drawing view object has a property to return the model document.

```
DrawingView.ReferencedDocumentDescriptor() As DocumentDescriptor
```

A *DocumentDescriptor* object describes the reference from a document to another document. A descriptor has all the information needed to find the referenced document and the state of the reference. You need to get this object first, and then the model document (e.g. Part or Assembly document). Below, the code demonstrates how to return the document object of the referenced model used by a drawing view.

```
Dim oModelDoc As Document
```

```
oModelDoc = oDrawingView.ReferencedDocumentDescriptor.ReferencedDocument
```

- **Set a referenced model**

When you place a drawing view using the Base View command using the User Interface, you look for a file in the project workspace or in the open documents list. Likewise, you need to set a model document when adding a base view programmatically. In fact, the first argument of the method `AddBaseView` of the `DrawingViews` collection object is the document of a 3D model.

*DrawingViews.AddBaseView( **Model As Document**, Position As Point2d, Scale As Double, ViewOrientation As ViewOrientationTypeEnum, ViewStyle As DrawingVisualStyleEnum, [ModelViewName] As String, [ArbitraryCamera] As Variant, [AdditionalOptions] As Variant ) As DrawingView*

## 5. Drawing Views

As described before in the section *3D Model Preparation*, the drawing views can be added with the methods of the `DrawingViews` collection object.

# DrawingViews Object

## Methods

Name
<a href="#">AddAssociativeDraftView</a>
<a href="#">AddAuxiliaryView</a>
<a href="#">AddBaseView</a>
<a href="#">AddDetailView</a>
<a href="#">AddDraftView</a>
<a href="#">AddOverlayView</a>
<a href="#">AddProjectedView</a>
<a href="#">AddSectionView</a>

Image 32 – Methods to add drawing views programmatically



## 6. Annotations

There are different methods to place annotations using Inventor API.

- **Attributes**

The concept of attributes was explained in the section *Working with Attributes for Inventor API*. By now, you should be familiar with attributes and how to create them. In this section, you will learn how to find them programmatically to place annotations.

The Inventor API has an object under the document object named *AttributeManager*.

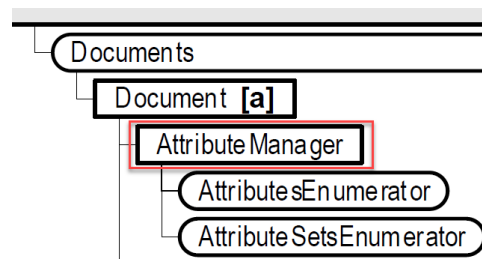


Image 35 – Attribute Manager object

It has a method to find objects that match the search criteria.

***AttributeManager.FindAttributes***( [AttributeSetName] As String, [AttributeName] As String, [AttributeValue] As Variant )

Below, there is a code that illustrates how to get a drawing curve on a view by finding a part edge with a specified attribute value. The part is in an assembly, then the edge proxy is required.

```

Dim oObjs As ObjectCollection
oObjs = oDoc.AttributeManager.FindObjects("General", "Name", "LEdge")
Dim oLEdge As Edge = oObjs.Item(1)
'Proxy of the Lower Edge to get the representation in the assembly
Dim oLEdgeProxy As EdgeProxy
oLowerCapOcc.CreateGeometryProxy(oLEdge, oLEdgeProxy)

Dim oDrawingCurves As DrawingCurvesEnumerator
oDrawingCurves = oView.DrawingCurves(oLEdgeProxy)
Dim oLowerCurve As DrawingCurve = oDrawingCurves(1)

Dim oLowerCurveGI As GeometryIntent
oLowerCurveGI = oSheet.CreateGeometryIntent(oLowerCurve)
  
```

- **Centermarks and Centerlines**

Another common method is the use of centermarks and centerlines. You can use existing centermarks and centerlines from holes, cylindrical and revolved geometries. You can also create them by using workfeatures from the 3D model. This alternative to place annotations was already described in the section *Working with Workfeatures* and *Understanding Geometry Intents*.

- **Using 3D and 2D Points**

Use a point from a 3D model to return it as a 2D point in the drawing sheet. This is a method of the drawing view object and its only argument is a point object (x, y, z).

*DrawingView.ModelToSheetSpace( ModelCoordinate As Point ) As Point2d*

Once you have taken a point from the model space and return a 2D point in the sheet space, use the method **FindUsingPoint** of a sheet object since it returns one of the following objects: drawing curve segments, entities on a sheet sketch, centermarks and centerlines.

*Sheet.FindUsingPoint( PointOnSheet As Point2d, [ProximityTolerance] As Variant )*

In case, your returned object is a drawing curve segment, then you can find its parent to use the drawing curve.

*DrawingCurveSegment.Parent() As DrawingCurve*

By now, you know that you can use any of these objects (drawing curves, centerlines and centermarks) to create the GeometryIntent object to attach an annotation.

Watch this demo about placing annotation with Inventor API <https://autode.sk/2IXSQfX>

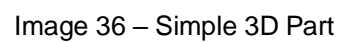
- **Retrieve Model Annotations**

Retrieve parameters from 3D model sketches and features. Retrieve is a method of the General Dimensions object. You need to specify the DrawingView or DrawingSketch to retrieve dimensions from. Optional argument is the object collection to indicate the dimensions to retrieve. If not specified, all the view or sketch dimensions are retrieved.

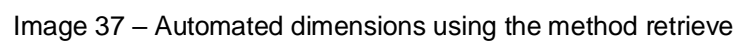
*GeneralDimensions.Retrieve( ViewOrSketch As Object, [DimensionsToRetrieve] As Variant )*

This is a handy method for simple geometries and shapes such as plates.

Watch this demo to learn about using retrieve with Inventor API <https://autode.sk/3jkeO9j>



Make sure you clean the sketches, placed the dimensions in the location the drawing will use as placement point. Place only the required dimensions since Retrieve without the optional argument ***[DimensionsToRetrieve]*** retrieves all.



For 3D models created with several sketches and features, collect sketch and feature parameters under one or more ObjectCollection objects to be used in one or multiple drawing views. Below, the images show a sketch for the main 3D model shape and another sketch to place holes using sketch points.

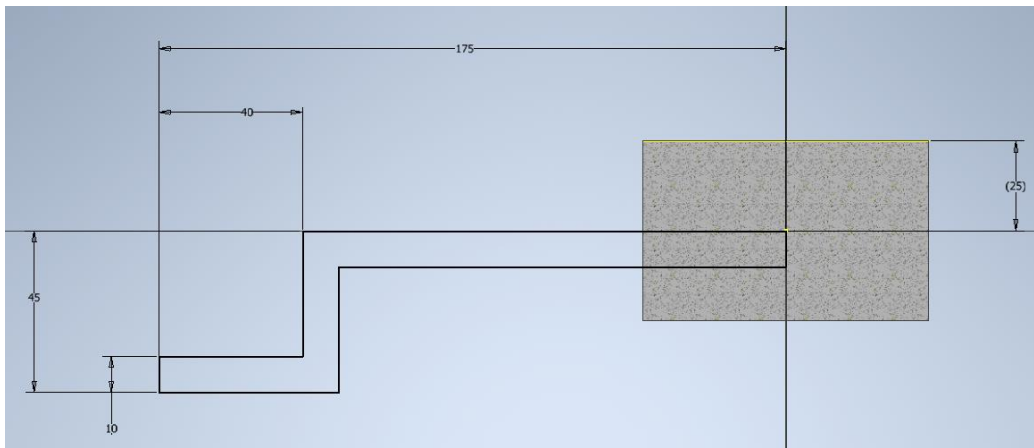


Image 38 – The main sketch of the 3D model

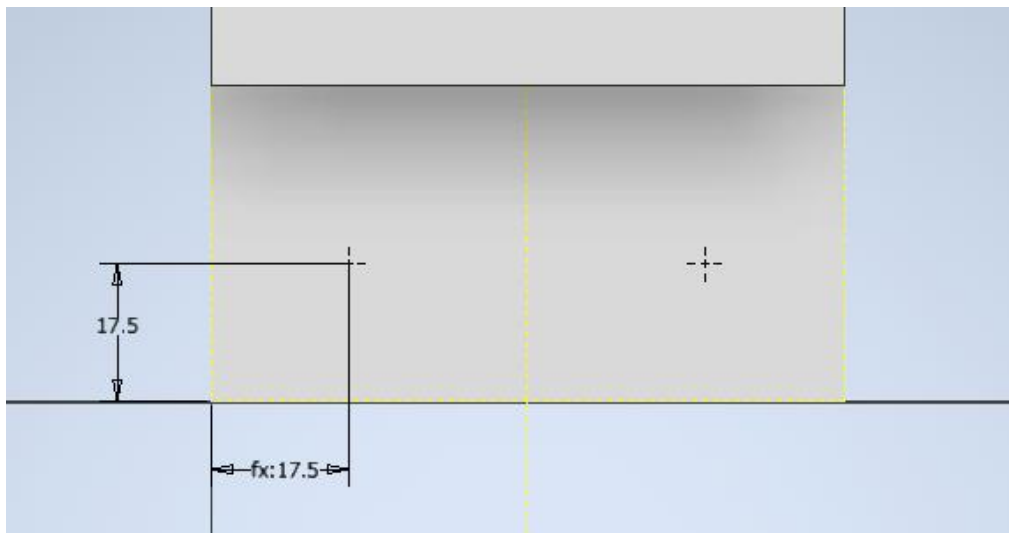


Image 39 – Sketch points for holes

The dimensions of the main sketch are collected in an object collection to be used in the base view and the two dimensions to position the sketch points are added to another object collection used in the projected top view. These two images only illustrate two

sketches, but the model contains more sketches and features. The code collects all sketch and feature dimensions in two object collections. One object collection is used for the base side view and the other one for the projected top view. The retrieve method is used more than once and the optional argument is used to specified the collected dimension constraints and feature dimensions.

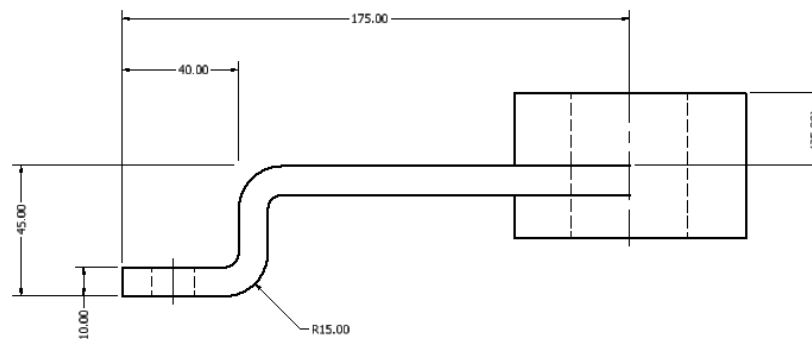
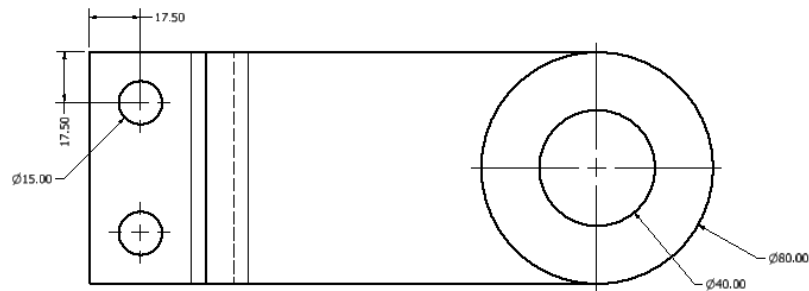


Image 40 – Automated annotations using the method Retrieve

## 7. Output

Inventor API has several options to present the drawings: hard copy, PDF, DXF, DWF, DWG and more.

The Autodesk Inventor 2021 Help provides sample programs under the Programming Interface section. Go to Translator > Export and you will find different examples to export drawings.

<https://help.autodesk.com/view/INVTOR/2021/ENU/?guid=GUID-DE98632B-3DC0-422B-A1C6-8A5A15C99E11>

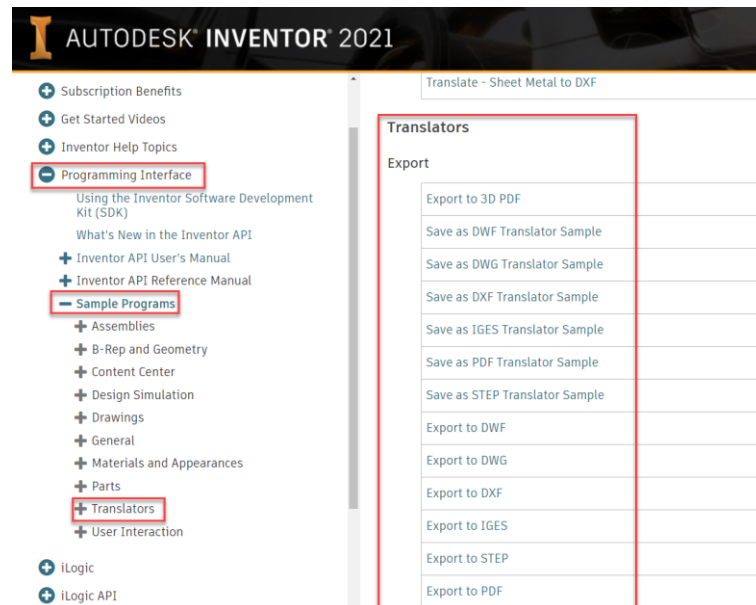


Image 41 – API Sample Programs

## Important concepts and API objects for drawing automation

### Collection Objects

Provide access to a group of related objects. In the Inventor API Object Model, they are represented with a box with rounded corners.

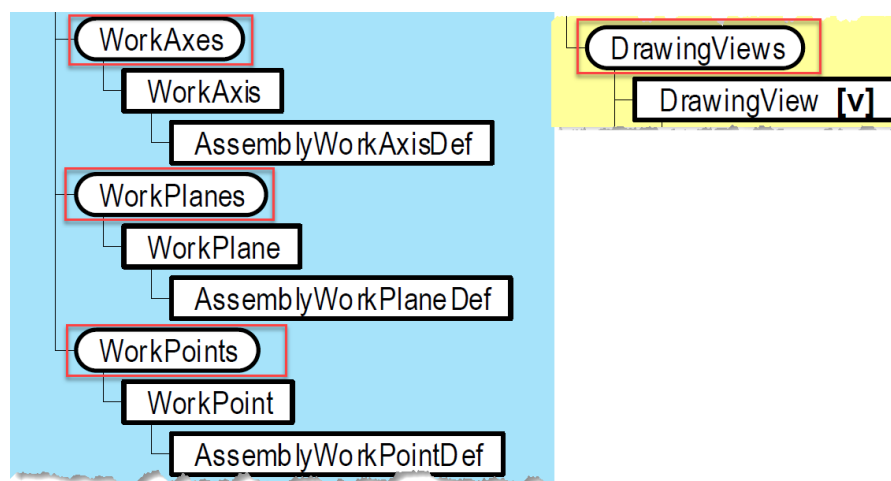


Image 42 – Inventor API collection objects

Some of them only allow you to use a number (e.g. 1, 2, 3) to return an object from the collection. For instance, the DrawingViews collection object only returns a drawing view with a number.

*Collection.Item(Index As Long) --> Only values, e.g. 1,2,3...  
DrawingViews.Item( Index As Long ) As DrawingView  
Index Long Input Long value that specifies the index of the collection to return.*

Other collections allow either a number or name. For example, the WorkPoints collection object lets you enter the work point name or the index number that returns the work point from the ordered list of work points in the model browser.

*Collection.Item(Index As Variant) --> values, e.g. 1,2... Or string "WorkPoint1"  
WorkPoints.Item( Index As Variant ) As WorkPoint  
Index Variant Input Variant value that specifies the object to return.  
This can be a **numeric value** indicating the **index of the item** in the collection or  
it can be a **string** indicating the **work point name**.*

## Transient Geometry

The term “transient” is used for objects defined temporarily. TransientGeometry is the way to define geometry temporarily to be used programmatically. For instance you can create points that the 3D model or 2D drawing do not have, but you need them in your code.

Method to create a point in a 3D space

*TransientGeometry.CreatePoint( [XCoord] As Double, [YCoord] As Double, [ZCoord] As Double )*

Method that creates a new Point2d object.

*TransientGeometry.CreatePoint2d( [XCoord] As Double, [YCoord] As Double )*

## Proxies

The concept of proxy in the Inventor API context is important when working with assemblies. The Inventor API Help defines the proxy as a *reference to an object through a particular occurrence*. For example, a part placed in an assembly is not embedded, it is actually linked. This means that the assembly does not have geometry. It only references geometry from components. This part has been placed twice (Part:1 and Part:2) in the assembly. In your code, you need to use an entity from the part (e.g. the top face) but only from the second occurrence (Part:2). Even though the top face in the first occurrence (Part:1) and the second occurrence (Part:2) is the same geometrically, they have different representations because they were placed in two different locations. Part:1 and Part:2 were constrained in two different 3D locations. If you need to query the top face of the second occurrence (Part:2), then you need to use a proxy.

In other words, a proxy is the representation of an object as if it were in the assembly.

In the user interface, you do not need to interact with proxies, but programmatically you must use proxies to work with the proper data in the programming interface.

This is Component Occurrence method that creates a proxy.

*ComponentOccurrence.CreateGeometryProxy( Geometry As Object, Result As Object )*

In this piece of code, a WorkPoint Proxy is created to be used in a drawing. First the WorkPoints collection of the part is obtained, then the second point of the collection is returned. This part may have one or multiple occurrences in the assembly, then the oPartOcc occurrence was previously defined (it is not displayed in these lines). Finally, a WorkPoint proxy is created using the second work point from this part occurrence.

```
Dim oPartWorkPoints As WorkPoints = oPartDoc.ComponentDefinition.WorkPoints
Dim oSecPartWPo As WorkPoint = oPartWorkPoints(2)

Dim oSecPartWPoProxy As WorkPointProxy
oPartOcc.CreateGeometryProxy(oSecPartWPo, oSecPartWPoProxy)
```

A part can be placed once or multiple times in an assembly. You take a specific occurrence named oPartOcc in this case. You need the second work point of the part (oSecPartWPo) but in this occurrence oPartOcc. The WorkPoint proxy (oSecPartWPoProxy) is the representation of the second work point (oSecPartWPo) in the assembly level.

If you want to know more about proxies, read this section in the Inventor API Help.

<http://help.autodesk.com/view/INVNTOR/2021/ENU/?guid=GUID-6A540540-CA8A-40AD-8EBF-C4BB1F3E7288>

## Best approach to automate your drawings

As mentioned in the introduction of the document, the drawing automation depends on the 3D models and type of drawing. Let's take a look at some different scenarios to understand if you should use iLogic, only Inventor API or a combination of both. In addition, these scenarios illustrate the possibility of completing the entire drawing workflow or only some stages programmatically. Every user would love a one button/click solution to complete the drawing but this may be unrealistic in some scenarios. You can create codes to complete some stages and segments of the others in the process. Although these complex scenarios will not let you get the drawing completely, you save time automating part of process.

### Scenarios

#### 1. Known Designs: Product Catalog and Standard Designs (Configurators)

- Approval Drawings (drawings for quotes): the entire drawing automation process can be completed.

##### Reasons:

- You are very familiar with your designs (geometry, parameters, etc.)
- Use of the same templates, drawing styles and standard.
- This type of drawing needs simple drawing views, overall dimensions and some other simple annotations.
- Standard output.

You can skip the first two stages (the creation of drawing standard, styles and all the drawing resources) since the templates and the design data files have all the required information. If you are using existing drawings and only need to reposition and scale drawing views, you can use iLogic snippets. However, if you need to place drawing views, your only option is Inventor API. Simple annotations can be done with iLogic snippets, but more detailed and complex annotations require Inventor API. The output always require Inventor API objects, methods and properties regardless of the the development environment you use (iLogic, VBA editor or Visual Studio). Use iLogic external rules or an add-in since they are generic programs to create the deliverable (PDF, DXF, DWG, print and more).

In summary, you can use only API or a combination of iLogic and API for this case.

- Manufacturing Drawings: the main difference between these two type of drawings, Approval and Manufacturing, is the detailed information for fabrication. A manufacturing drawing requires more drawings views (sections, details, auxiliary views, etc.) and very specific annotations for manufacturing (all dimensions, symbols, notes, tables, etc.). Then the main differences between this automation process and the first one are the drawing view and annotation stages. The complexity involved in these two steps require more complex programming and library objects that iLogic does not have. Inventor API is the only solution in this case.

2. Unknown Designs: Customer-Based Designs, design process to create a new product and other similar scenarios.

The complete drawing automation is unlikely to be accomplished for any kind of drawing. Unknown information such as geometry to be used to place annotations, amount of drawing views, required annotations and so on, makes the process unthinkable. However, some general tasks in some steps specially the first steps and the last one can be automated even though the model is unknown. These stages: drawing standards, styles, drawing resources, drawing setup, the referenced models and the output are very generic steps that can be automated. The only option in this case is Inventor API and best way to run the programs is with add-ins. Regarding the two most complex steps in the drawing workflow, drawing views and annotations, you can automate little pieces of the steps. For instance, start a drawing by automatically placing the most common set of drawing views: 3 orthographic views (front, right and top) and isometric view. Then, you will complete the rest of views such as sections, detail views and more. In the annotation segment, you can automate the placement and location of some tables, parts lists, some overall dimensions, general notes, centerlines and centermarks. You will need to complete the rest of the annotations manually.

In summary, this scenario will require user input but you save time in some repetitive and common tasks.

## Resources

- API Help

<http://help.autodesk.com/view/INVENTOR/2021/ENU/?guid=GUID-6FD7AA08-1E43-43FC-971B-5F20E56C8846>

- Autodesk Website – What's New in Inventor 2021 and 2020

<https://knowledge.autodesk.com/support/inventor/learn-explore/caas/CloudHelp/cloudhelp/2021/ENU/Inventor-WhatsNew/files/GUID-7BECFFC8-8237-4F58-B285-4975FE9C79F9-htm.html>

<https://knowledge.autodesk.com/support/inventor/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Inventor-WhatsNew/files/GUID-80AD0392-0B8C-4A27-A9B3-7466D53999BF-htm.html>

- Autodesk blog - Mod The Machine

[https://modthemachine.typepad.com/my\\_weblog/2010/02/accessing-iproperties.html](https://modthemachine.typepad.com/my_weblog/2010/02/accessing-iproperties.html)

[https://modthemachine.typepad.com/my\\_weblog/2009/07/introduction-to-attributes.html](https://modthemachine.typepad.com/my_weblog/2009/07/introduction-to-attributes.html)

[https://modthemachine.typepad.com/my\\_weblog/2009/09/attribute-utility.html](https://modthemachine.typepad.com/my_weblog/2009/09/attribute-utility.html)

- Brian Ekins website - Apps section: Nifty Attributes

[https://ekinssolutions.com/nifty\\_attributes/](https://ekinssolutions.com/nifty_attributes/)

- Autodesk University classes

iLogic and the Inventor API

<https://www.autodesk.com/autodesk-university/class/iLogic-and-Inventor-API-2016>

Hidden Secrets of Automating Drawings Using Inventor

<https://www.autodesk.com/autodesk-university/class/Hidden-Secrets-Automating-Drawings-Using-Inventor-2019>