



IM319632

Hidden Secrets of Automating Drawings Using Inventor

Brian Ekins
Ekins Solutions LLC

Learning Objectives

- Learn about the internals of Inventor drawings.
- Learn about different strategies for automating drawings.
- Learn how to add annotations to a model.

Description

Automating the creation of drawings is a difficult process. This class will look at the internals of Inventor software to help you better understand what's really going on when a drawing is made. With this understanding as a foundation, we'll look at some different approaches to automating creating a drawing along with several sample programs using Inventor software's API (Application Programming Interface) that demonstrate the various techniques.

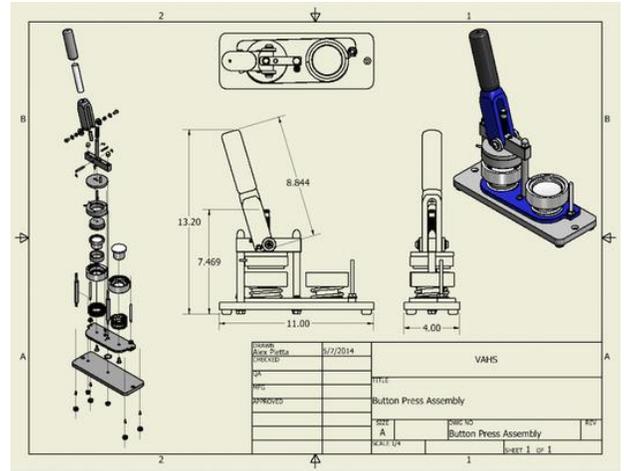
Speaker

My work in the CAD industry has been varied and began in 1977 when I saw my first CAD system. Since then I've developed and taught courses, worked as an Application Engineer where I supported sales by working with potential customers to model parts, build assemblies, create drawings, performed analyses, machined parts, and customized the software. Because of my end-user perspective and experience writing customizations, I moved into the role of the designer for the Solid Edge API. Later, I moved to Autodesk where I designed the API for Inventor and then Fusion 360. As part of the API designer role, I've written the documentation, many of the sample programs, taught classes and supported those using the API's. Now I have my own business where I help Inventor and Fusion 360 users be more productive by providing tools and customizing the software into a tool specific to their needs.

Introduction

Automating drawings is not an easy task and over the years there have been many presentations at Autodesk University discussing different approaches. For this presentation, I'm hoping to take a slightly different approach and take a "behind the scenes" look at how Inventor creates a drawing. I believe having this understanding can help you better understand how to accomplish the things you need to do.

To get started let's dissect the pieces of drawing and look at each piece in more detail. If we look at the drawing to the right, it consists of a sheet of a defined size, a border, title block, various views of an assembly and some dimensions. Those are the result but what are the underlying objects that allow those to be created? That's what this next section discusses.



Anatomy of an Inventor Drawing

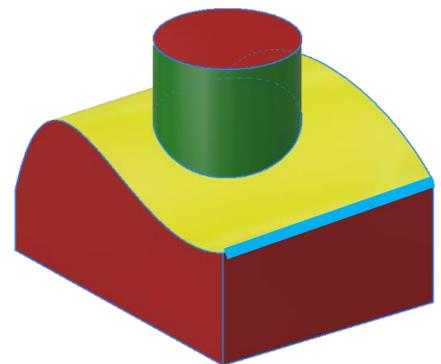
Let's dissect the drawing above and look at all the pieces that are needed to make it. Some of this may seem like it doesn't have anything to do with drawings, but you'll see when we get to the example programs how having this knowledge allows you to access the information you need to automate the drawing process.



Parts

To a drawing, a part is a Boundary-Representation (B-Rep) of a solid model. The entire discussion about B-Rep data is beyond the scope of this class but there are a couple of things that are important to understand.

People typically think of a part as a collection of features. It's true that the features define how to construct and recompute a part but the result of computing those features is a B-Rep solid model. The most important B-Rep concept to understand is that a solid model isn't really solid but is made up of a collection of surfaces that tightly connect to create a watertight volume. In the picture to the right, I've colored the faces red, yellow, and green. Even though the surfaces have different shapes they're all represented in the B-Rep model as a *face*. In this example, the red faces have planes that define their geometry, the green face has a cylinder, and the yellow face has a spline surface.



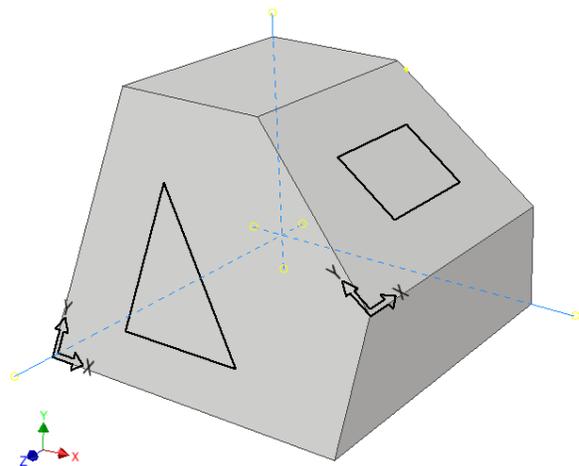
In addition to faces, a B-Rep model also has *edges*. Edges are the curves in the model. They define the outer shape of each face and are also the intersection of adjacent faces. Faces share edges so the edge highlighted in cyan is a single edge and is shared between the red and yellow faces it connects. Edges also have geometry associated with them and the cyan edge has a line that defines its geometry but other edges in this model use a spline or circle.

Part Space

Something that's probably obvious but needs to be clarified is the coordinate system that's used when modeling a part. You can see the coordinate system of the part by looking at the base work features of the part. The origin point is at the (0,0,0) point of part space and the X-Axis points in the X direction, the Y-axis in the Y direction, and the Z-axis in the Z direction. All the B-Rep geometry in the part is defined with respect to this coordinate system.

Sketch Space

Another coordinate system to be aware of within a part is the sketch coordinate system. In the picture to the right, there is a solid model and two sketches. The dashed blue lines are the X, Y, and Z work axes which intersect at the origin of part space. There are sketches drawn on two faces of the model. Sketches are truly 2D, have their own coordinate system and all the geometry they contain is relative to the sketches coordinate system. The sketch defines how its 2D coordinate system is positioned in 3D space. In the picture to the right, I've illustrated the coordinate system of each of the sketches. If the model should change in shape that will cause the sketches to recompute and reorient themselves which will change the position and orientation of the sketch coordinate system and the geometry in the sketch will move to maintain their same position relative to the sketch coordinate system.



Assemblies

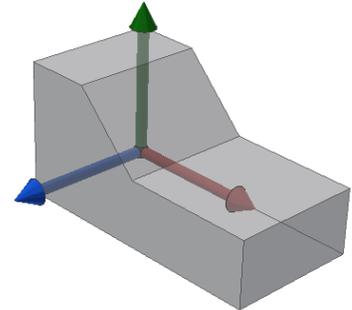
An assembly is a grouping of parts or other assemblies. The assembly environment provides tools to add parts and assemblies, position them with respect to each other, and get reports about what the assembly contains. It's important to remember that the assembly doesn't contain the geometry, but references part documents that contain the geometry.

When you insert a part or assembly into an assembly, you're creating an *occurrence*. Conceptually, an occurrence is quite simple in that it is a reference to a part or assembly file and a position and orientation of where that part or subassembly exists within the assembly.

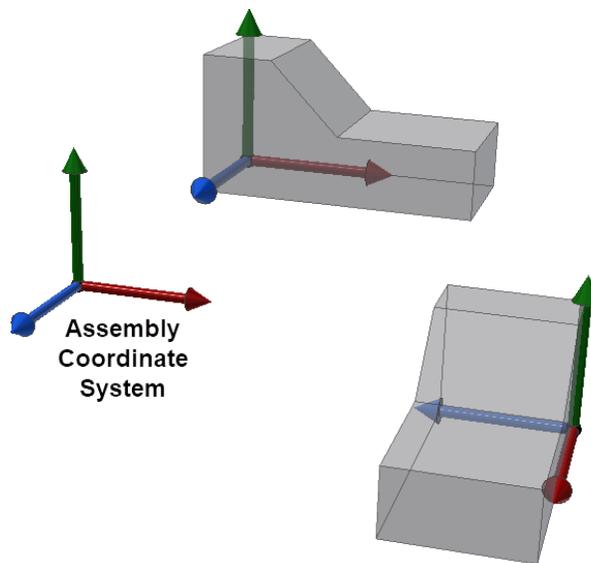
Assembly Space

Placing a part into an assembly doesn't have any effect on the part. In fact, the part doesn't even know it's being used in the assembly. The part coordinate system remains unaffected and all the geometry in the part remains defined with respect to the part coordinate system. However, each assembly has its own coordinate system and the occurrences are positioned relative to the assembly coordinate system.

Here's a simple example where we start with a part and its coordinate system. Remember that all the part geometry is relative to this coordinate system.

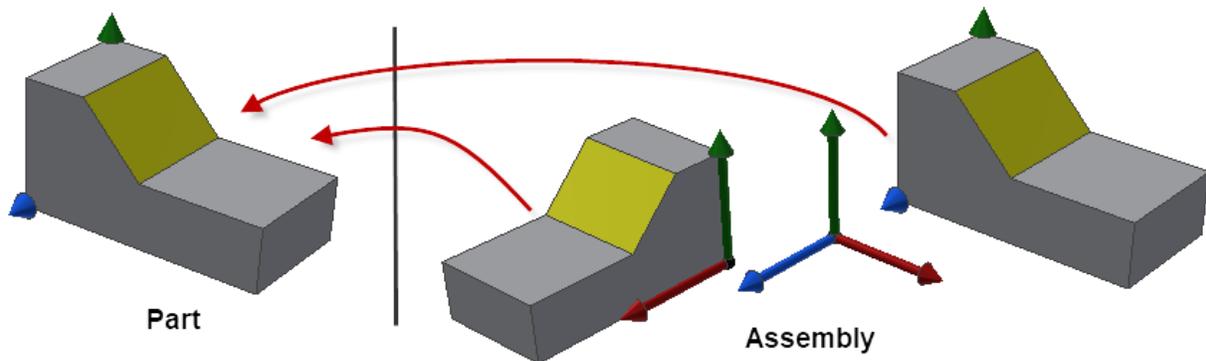


Now we place two instances of that part into an assembly. The assembly has its coordinate system and the occurrences exist in the assembly and reference the part. In this case, the occurrences are referencing the same part but they define a unique position for each occurrence within the assembly coordinate system.



Proxies

Proxies are very important, but it is a topic that people struggle with. Here's an overview description that will hopefully help. Looking at the yellow face shown below, the real face exists inside the part. In the assembly, the part has been placed twice. To the end-user, it now appears that there are two unique yellow faces in the assembly. I can get coordinates from the face and they're different for each face. I can use the faces to create constraints and they behave independently. For all practical purposes, there are two yellow faces in the assembly.



The real face only exists in the part and what you're seeing, measuring, and selecting in the assembly is a face proxy. The proxy represents the face as if it exists in the assembly. If you perform a measurement to get the position of one of the corners of the face, Inventor does some tricks internally to give you the answer you expect. It gets the position of the real corner in the part relative to the part coordinate system and then because the occurrence defines the position of the part in the assembly, it uses information in the occurrence to change the measurement result so it's correct for the assembly. Because the occurrence is different for each of the instances in the assembly, the results will be different when I get the coordinates from one of the faces versus the other.

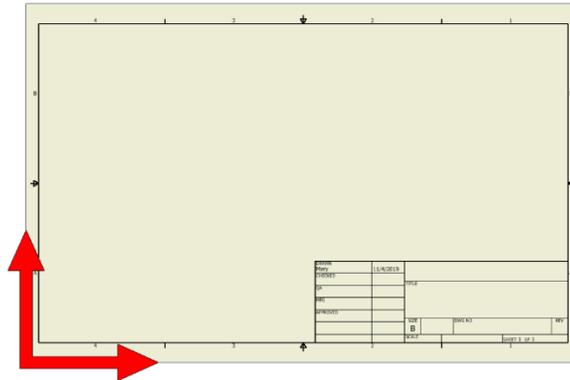
The important thing to remember is that if you're working with geometry in an assembly, you're working with proxy objects that represent the real geometry in the part. In most cases, this is all transparent when using the API, but there are some cases where you need to understand this in order to get things to work.

Drawings

Conceptually a drawing is a different type of display of a part or assembly that also supports various types of annotations to let you dimension and add other kinds of notes and labels to provide an accurate, fully described, documentation of the part or assembly. The interesting thing for us is how the internals of Inventor work to go from a 3D model of any size to a sheet of paper.

Sheets

The core of a drawing is a sheet and it owns all the drawing specific data. Like parts and assemblies, a sheet also has its own coordinate system where the origin is in the lower-left corner of the sheet, and X is to the right and Y is up, as shown below.

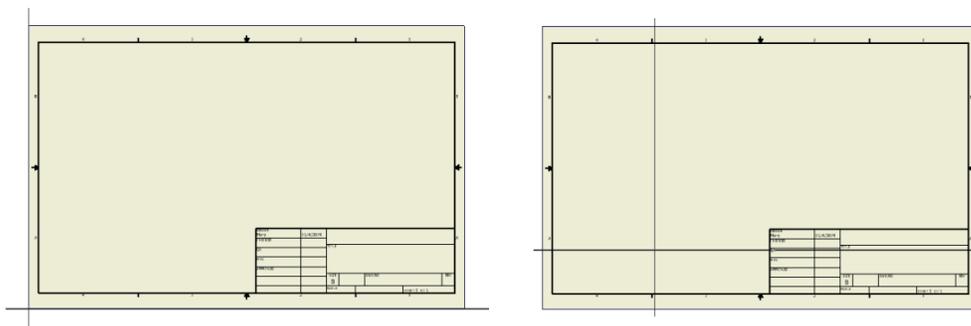


All the things that the sheet owns (drawing views, dimensions, notes, etc.) are positioned on the sheet relative to this coordinate system. This coordinate system is fixed and can never change.

Sheet Sketches

You can also create a sketch on a sheet and draw 2D geometry. When you first create a new sketch, it's coordinate system is the same as the sheet. However, once it's created it's possible to drag the sketch around on the sheet so the origin of the sketch coordinate system can be different than the sheet coordinate system. As far as I'm aware, it's not possible to rotate the sketch so it's X and Y axes are always in the same direction as the sheet.

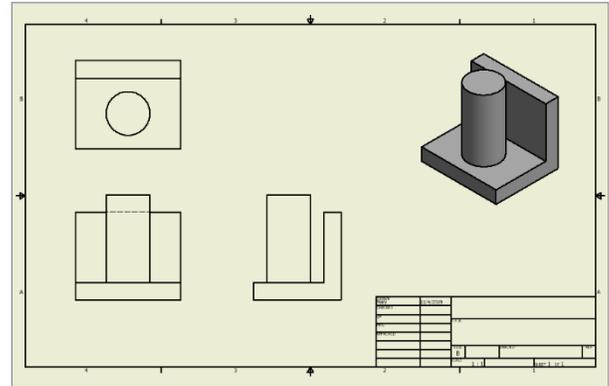
It's easy to know where the origin of the sketch is because when the sketch is active, Inventor draws lines along the X and Y axes, as shown below. The picture on the left shows a sketch in the default location and the one on the right shows a sketch that has been moved.



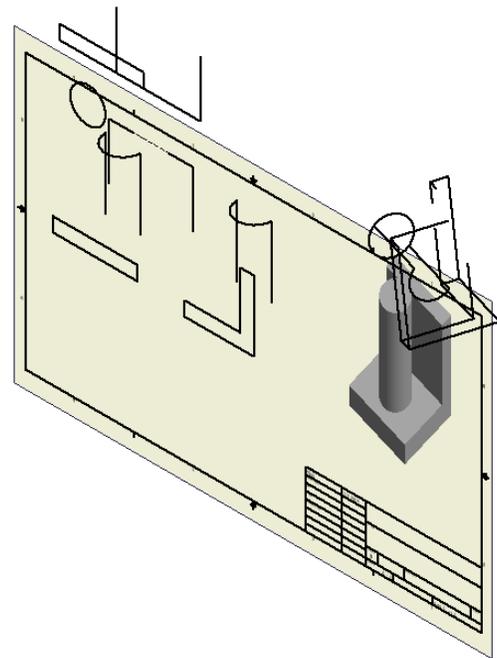
The scale of a sheet sketch is always 1:1. That is if you draw a line that is 5 centimeters long it will be 5 centimeters long on the sheet.

Drawing Views

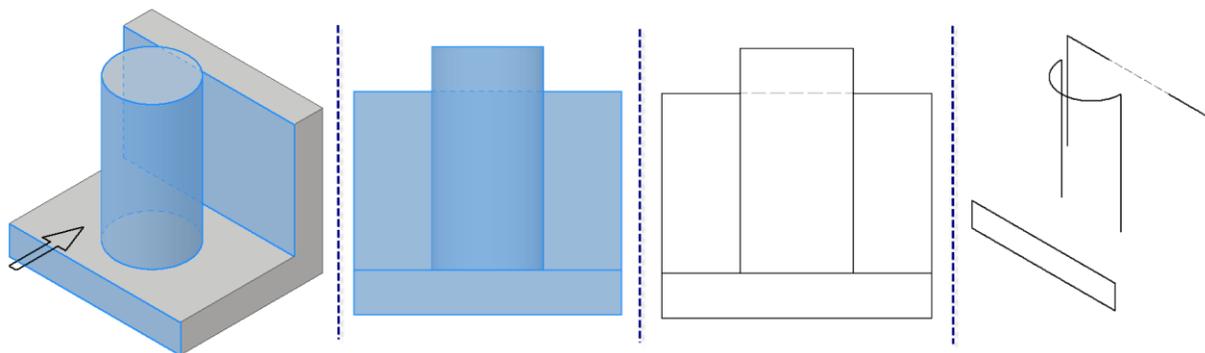
Drawing views allow you to reference a part or assembly and create a 2D representation suitable for a drawing. When you create a drawing view you specify the part or assembly file and some other options that control how the view is drawn (scale, style, which representation to display, the label, etc.). The drawing view is then added to the sheet. Its position is defined relative to the sheet coordinate system. To the right is an example of views placed for a very simple part. The graphics Inventor creates provide an accurate 2D representation just as if you drew it using pencil and paper or AutoCAD, however, everything is not as it seems.



When you're in a drawing, the view commands only allow you to zoom and pan but not rotate the view. However, the API doesn't have the same limitation. To the right is the same drawing after changing the view to an isometric view.



You can see in the rotated view that it's not a simple 2D drawing but is a modified version of the 3D model. The picture below illustrates better what's happening. The figure on the far left is the solid model with an arrow showing the direction of the camera for the desired drawing view. Also, the three faces that are visible from that direction are highlighted in blue. The second figure shows the same solid model but from the orientation of the desired view. The third figure shows the view in the drawing and is exactly what you would expect. The surprise is the fourth figure which shows the rotated view. Compare the first and fourth figures and you can see that the curves represent a modified version of the original solid.

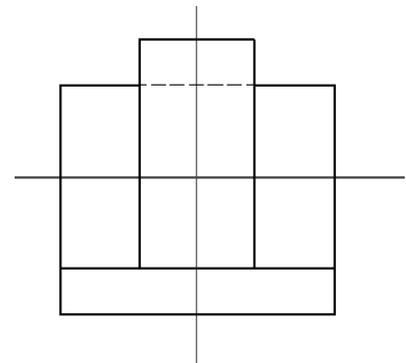


There are three things I want to point out. First, is that the line along the back of the part changes between visible and hidden. Second, is the lines in the drawing that represent the edges of the cylinder don't exist in the model. Silhouette edges are created as part of the drawing view calculation process. Third, is that there aren't any overlapping curves.

Each drawing view also has its own coordinate system. For a drawing view, the origin of the coordinate system is in the center of the view. The scale of the drawing view coordinates are the same as the drawing view. Where the drawing view scale primarily comes into play is when you create a sketch on a drawing view.

Drawing View Sketches

When you run the **Start Sketch** command you can click on an existing drawing view or the sheet. If you click a drawing view it will create a sketch associated with the drawing view. In the picture to the right, a drawing view was clicked, and a sketch created. You can see by the axis lines that the origin of the sketch is at the center of the view. The location of the axes lines is a good visual indicator of whether you created a sketch on a drawing view or the sheet.

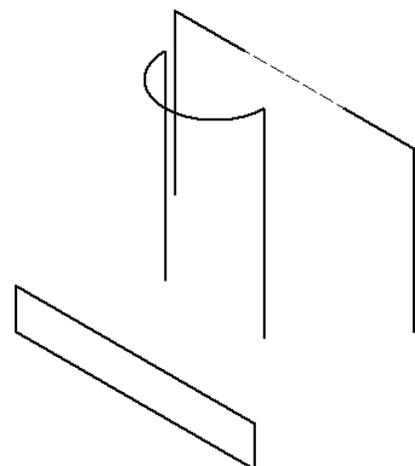


The scale of a drawing view sketch is the same as the drawing view. For example, if you've created a drawing view that is half scale and create a sketch on that view and then draw a line that is 5 inches long, it will display on the sheet as 2 ½ inches. The sketch line is 5 inches long but its display is half that. However, if you dimension the line, the dimension will show it being 5 inches long.

Like a sheet sketch, a drawing view sketch can be moved after it's created so its origin doesn't have to be at the center of the drawing view.

Drawing Curves

We've seen that there is some unexpected geometry in a drawing and that the geometry is more like the 3D model it came from instead of simple 2D geometry like you would expect. Although this geometry is like geometry in the 3D model it also has some differences. If we go back to the previous model, there are some good examples there. The top of the cylinder is a full circle in the model but in the drawing, it's a 180-degree arc because that's all that can be seen in the drawing. The upper horizontal line is covered in the center by the cylinder, so the center of the line is shown with a dashed line. And finally, the silhouette of the cylinder is represented by two lines.



All the geometry you see in a drawing view is represented by DrawingCurve objects. A DrawingCurve object has a relationship with the model or sketch geometry that is was

created from. All the curves in the picture above are associated with an edge that exists in the original part model, except for the two silhouette lines which are associated with the cylindrical face. Through the API you can get the model entity the drawing curve is associated with.

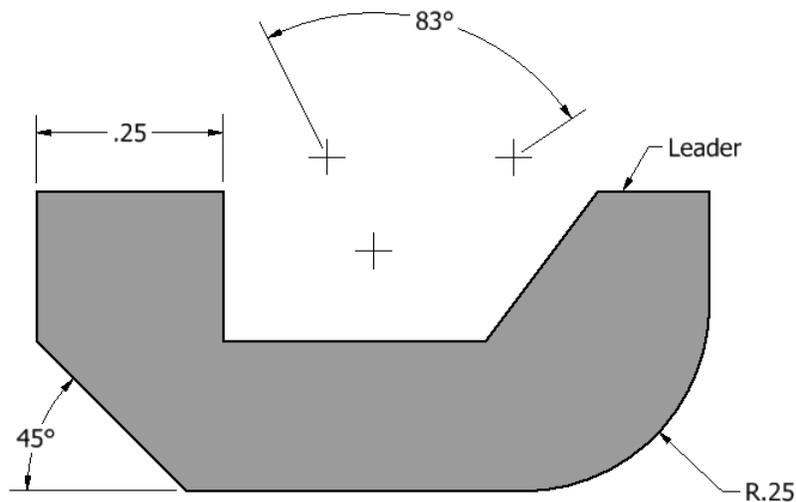
It's these drawing curves that you'll use when adding any annotations that are attached to geometry; dimensions, leaders, balloons, etc.

Drawing Curve Segments

There's also an object called `DrawingCurveSegment`. When the user selects any of the geometry in the drawing view, what's returned is a `DrawingCurveSegment` object. Typically, there is a one-to-one relationship between a `DrawingCurveSegment` and a `DrawingCurve` but there are cases where there can be multiple `DrawingCurveSegments` associated with a `DrawingCurve`. In the picture above, the line that is partially hidden is a good example. That entire line is one `DrawingCurve` but it has three `DrawingCurveSegment` objects. If you work with selections or highlighting you'll use `DrawingCurveSegment` objects but when working with annotations you'll use `DrawingCurve` objects. It's easy to go from one to the other. The `Parent` property of the `DrawingCurveSegment` object returns its parent `DrawingCurve` object and the `DrawingCurve` object has a `Segments` property that returns a collection of the `DrawingCurveSegment` objects associated with it.

Geometry Intents

When adding annotations, you specify what the annotation will attach to by passing in `GeometryIntent` objects. The `GeometryIntent` object is one of the more confusing objects in the drawing API. The purpose of the `GeometryIntent` object is to provide a convenient way to pass the information that defines how an annotation attaches to geometry. It's a combination of the "Geometry" and the "Intent" of how you want to attach to the geometry. By combining the geometry and intent into a single object it makes the API functions that need to use this information must simpler. Let's look at some examples using the annotations below.



In many cases, just the geometry is enough and the intent portion isn't needed. For example, to create the arc dimension in the picture above, the VBA code below can be used. Notice that when creating the GeometryIntent object, only the Drawing curve is passed in. Remember that selections return a DrawingCurveSegment but annotations require a DrawingCurve. The second argument to the CreateGeometryIntent method is for the intent and is optional and in this case, isn't needed. For an arc and some other dimension types, a combination of the geometry and the point specifying the position of the dimension text is enough for Inventor to know how to attach the annotation.

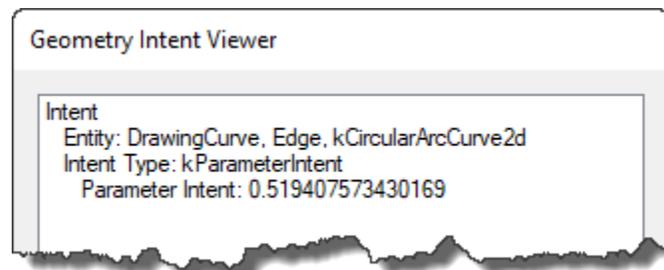
```
' Have a drawing curve segment selected.
Dim curveSeg As DrawingCurveSegment
Set curveSeg = ThisApplication.CommandManager.Pick(kDrawingCurveSegmentFilter, "Arc 1")

' Get the parent drawing curve from the segment.
Dim curve As DrawingCurve
Set curve = curveSeg.Parent

' Create a GeometryIntent that only contains the curve.
Dim intent1 As GeometryIntent
Set intent1 = sht.CreateGeometryIntent(curve)

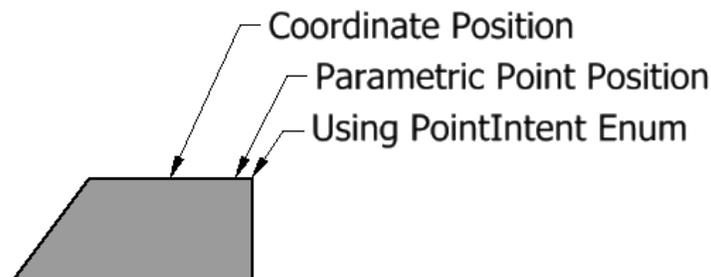
' Add the dimension.
Call sht.DrawingDimensions.GeneralDimensions.AddRadius(tg.CreatePoint2d(15, 10), intent1)
```

The same is true for most of the annotations in the previous picture. To create the linear dimension, the two lines are enough because they're parallel and Inventor can figure out what to do. For the angle dimension, the two lines are also enough. For the angle between the center marks, the three center marks without any intent information are enough. I've written a little utility that can be useful to understand what geometry intents Inventor is expecting. Look for the IntentViewer project in the associated class material. To use it you manually create the annotation you want and then run the utility and select the entity. Below is the result when querying the radial dimension placed above. In this case, Inventor is returning a GeometryIntent that has more information than what was provided when the dimension was created. It's using a parameter value to specify the location of where the dimension attaches to the arc. What a parameter value is, is discussed below, but this shows that you don't always need all the information that Inventor is returning for an existing entity. In this case, where the dimension attaches to the arc is dependent on the position of the dimension text. When writing code to create annotations, you can start with a GeometryIntent that's just the geometry and then add intent information only if it's required which you can determine with a little bit of trial and error.



One more thing about the Geometry part of a GeometryIntent object is that it's not limited to DrawingCurve objects. You can attach some annotations to other types of objects. For example, in the picture above, the angle dimension is referencing the three center marks. It's also possible to attach some annotations to a dimension and leaders to almost anything. In those cases, you provide the object you want to attach to as the Geometry argument of the CreateGeometryIntent method.

Now, let's look in more detail at the "intent" part of a geometry intent. This is where the concept of the GeometryIntent gets much more confusing because there are five different ways to specify the intent. The picture below shows the result of using items 2, 3, and 4 from the list below.



1. No intent – We already discussed this above, where you don't provide an intent argument when the geometry by itself provides Inventor with enough information to place the annotation.
2. 2D point – In this case you provide a Point2d object as input to specify a coordinate location where the annotation should connect. Typically, this is a point on the curve or other annotation you're attaching to but if it's not directly on the entity the input coordinate will be projected onto the entity. The example code below uses this by getting the position of the midpoint of the drawing curve and then using that coordinate as input to create the leader.

```

' Get the coordinate of the midpoint from the drawing curve.
Dim pnt As Point2d
Set pnt = curve.midPoint

' Create the Geometry Intent using the point.
Dim geomIntent As GeometryIntent
Set geomIntent = sht.CreateGeometryIntent(curve, pnt)

' Define the input and create the leader.
Set leaderPoints = ThisApplication.TransientObjects.CreateObjectCollection
Call leaderPoints.Add(tg.CreatePoint2d(20, 14))
Call leaderPoints.Add(geomIntent)
Call sht.DrawingNotes.LeaderNotes.Add(leaderPoints, "Leader Text")

```

3. Parameter – This is different than the parameters you use to control the size of your model. All curves have their own 1-Dimensional coordinate system which is referred to as "parametric space". You can think of it as a number line that follows along a curve.

From a drawing curve, you can get a Curve2dEvaluator object that will give you information about the curve in its parametric space. Some useful functionality on the Curve2dEvaluator object is the GetParamExtents method which returns the min and max param values of the curve. There are also some methods to get the length of the curve at a parameter value, the parameter value at a certain length, and convert between model and parametric space. The example code below uses this to attach a leader at 1/10th of the parametric length of the curve from its start point.

```

' Get a parameter value that is ten percent along the parameter
' space of the curve.
Dim minParam As Double
Dim maxParam As Double
Call curve.Evaluator2D.GetParamExtents(minParam, maxParam)
Dim param As Double
param = ((maxParam - minParam) * 0.1) + minParam

' Create the Geometry Intent using the parameter value.
Dim geomIntent As GeometryIntent
Set geomIntent = sht.CreateGeometryIntent(curve, param)

' Create the leader using the code in the sample above.

```

4. PointIntentEnum – You can use a value from the PointIntentEnum to specify a specific “key point”. This enum is large and what’s a valid value will depend on the geometry type and what kind of annotation you’re placing. The most common values used are kStartPointIntent, kMidPointIntent, and kEndPointIntent to specify the start, middle, or end of the curve but there are others when the geometry is a circle to specify points around the circle, and others to specify a location around a dimension or text box. This is used in the example below to attach the leader to the start of the curve.

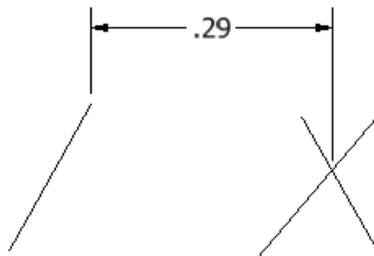
```

' Create the Geometry Intent using the enum value.
Dim geomIntent As GeometryIntent
Set geomIntent = sht.CreateGeometryIntent(curve, PointIntentEnum.kStartPointIntent)

' Create the leader using the code in the sample above.

```

5. Another Curve – The intent object can be a second DrawingCurve when you want to place the annotation at the intersection of two curves, as illustrated below.



Drawing Automation

There are several approaches to automating the creation of a drawing. Let me begin by saying that I don't believe it's currently possible to write a program that can take ANY model and will create a good, complete drawing. Having said that, there are still things we can do to help with the creation of drawings and for a known part or assembly it is possible to fully automate the creation of a drawing.

Use Inventor Associativity

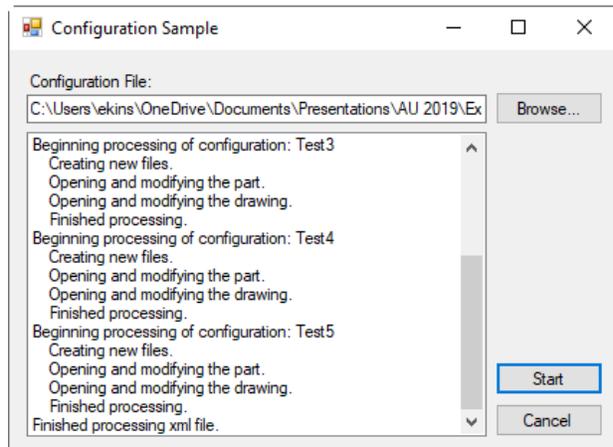
This first approach to drawing automation takes advantage of Inventor associativity and in a lot of cases doesn't require any use of the drawing portion of the Inventor API. With this method, you manually create the drawing. Once you have the model and a good drawing you can automate creating a copy of both files, editing the model to get the desired size and then allowing the drawing to automatically update.

In the best case, the creation of the "new" drawing is fully automatic and is done just by opening the drawing and allowing it to update. In some cases, there might be a little bit of clean-up but still much less than if you started from scratch.

An example of this approach is the Configurator sample that's part of this presentation. This is an interesting sample because it demonstrates the use of a lot of different functionality. The main idea is that it starts with a parametric model that will result in any of the variations you want by editing parameters and possibly suppressing and un-suppressing features.

Some tools this sample uses are Apprentice and some basic use of the Inventor API. Apprentice is a programming tool that provides a small subset of the Inventor API functionality and provides some additional capabilities that the Inventor API doesn't have. One of these is the ability to create copies of files and update the references within the files to reference the new copies. The Design Assistant utility that's delivered with Inventor uses Apprentice.

The sample uses an input XML file where you specify the "template" files, which is the original part and drawing, and the parameter values for each new instance you want to create. It then creates a new folder for each instance and using Apprentice copies the template files over using the new names specified in the XML file. The references between files are updated as part of this step. Next, it uses the Inventor API and Inventor to open the part and change the parameters. Finally, it opens the drawing and allows it to update and then saves the updated files.

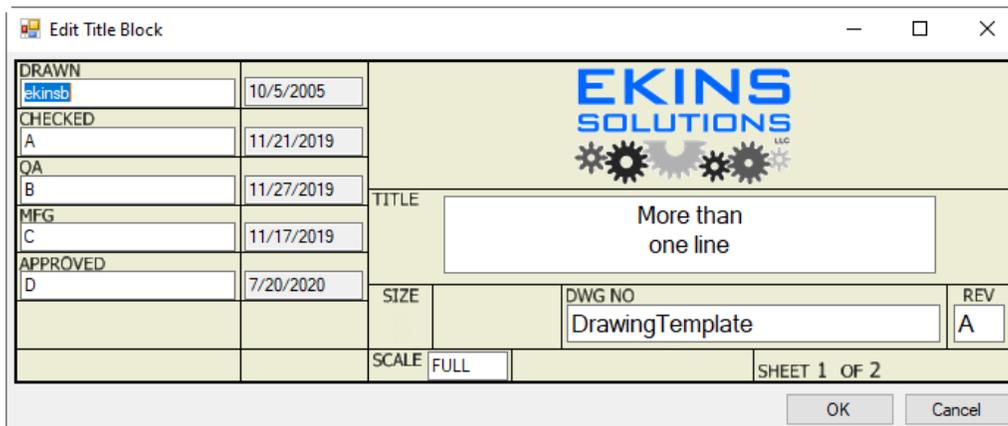


Drawing Tools

Another approach to improving the drawing creation workflow is to provide additional custom tools that the detailer can use to make them more productive. These kinds of tools can be very beneficial because they typically provide a generic capability that can be used on any drawing and doesn't involve coding for a specific part or assembly. The capability that these tools all rely on is the drawing portion of the Inventor API. Below is a list of sample programs provided as part of this presentation.

Title Block

This sample helps to edit the contents of the title block. All the program is doing is editing iProperty values, which the title block is dependent on, but it makes it much easier for the user because they don't have to understand which iProperty connects to which field in the title block.



DRAWN	ekinsb	10/5/2005		
CHECKED	A	11/21/2019		
QA	B	11/27/2019		
MFG	C	11/17/2019		
APPROVED	D	7/20/2020	TITLE	
			More than one line	
		SIZE	DWG NO	REV
			DrawingTemplate	A
		SCALE	SHEET 1 OF 2	
		FULL		

Custom Tables

The API supports creating tables which allows you to write a program that takes any external information and create a table with it. The CreateTable sample program creates a table using the contents of a specified Excel file.

Custom Notes

This sample lets you predefine a large list of common notes and then lets the user choose from the list and places a text box with the specified notes.

Center Dimensions

This sample helps to automatically clean up an existing drawing by centering the text between the extension lines on all linear dimensions. This can be particularly useful when you're parametrically editing a model that has an existing drawing but then need to clean up the drawing because the geometry has moved.

Balloon Renumbering

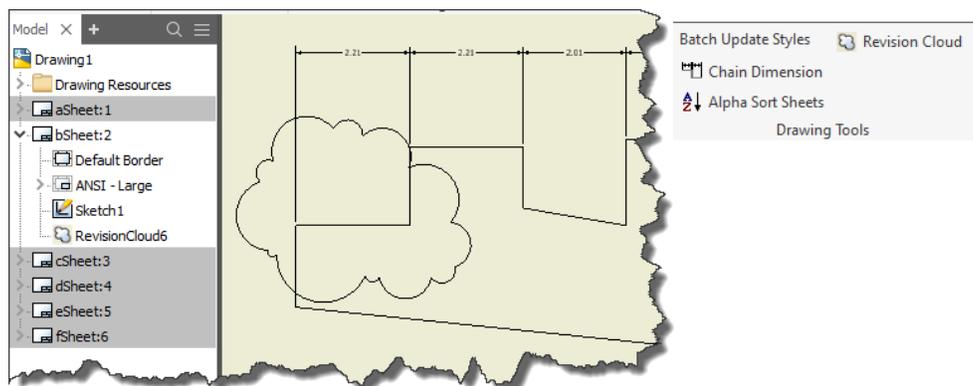
This program renumbers the balloons, so they are sequential beginning with the selected balloon and going in either a clockwise or counterclockwise direction around the drawing view.

User Tools

Inventor is delivered with a set of programs that are intended to be programming samples, but they also provide some useful functionality. Several of these provide functionality specific to drawings. To access these tools, go to the folder shown below for the version of Inventor you're using:

C:\Users\Public\Documents\Autodesk\Inventor 2020\SDK

In the SDK folder, there are two .msi files. These are installers that will install either the Developer Tools or the User Tools. Double-clicking will start the installer. Once installed you'll see the Drawing Tools panel in the Add-Ins tab of the ribbon. These samples illustrate some of what the Inventor API is capable of. The source code is provided as part of the installation but unfortunately, it's a C++ program so it will be difficult for many people to easily follow what it's doing. However, no matter what language you're using it's all using the same API and that add-in could have been written in Visual Basic or C#.



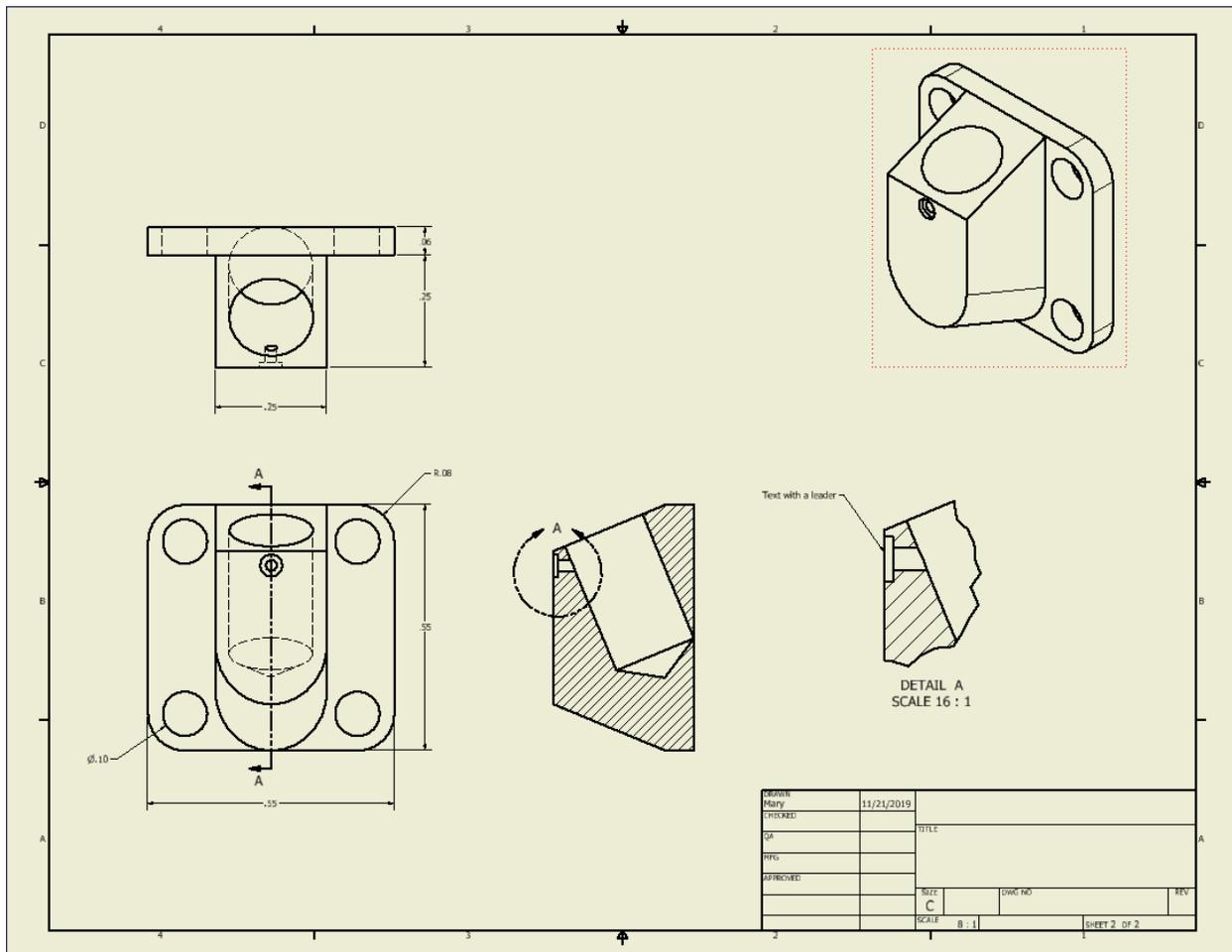
Drawing Automation

The next level of automation is to automate the creation of a drawing. Essentially, you want to automate the steps that a detailer would go through when creating a drawing; placing drawing views and adding the various annotations.

As I said before, I don't believe it's currently possible to create a complete and accurate drawing for ANY part or assembly. Many of the decisions you make when creating a drawing are very subjective and depend on the type of model, its shape, what needs to be shown, and company standards. For example, a drawing for a complex casting is going to be very different than a drawing for a structural channel. The casting drawing is likely to need section views, detail views, auxiliary views, and notes specific to that type of model and the structural channel drawing will be much simpler.

Of course, the simpler the drawing the easier it will be to automate. A program to automate the creation of a part drawing needs to have intimate knowledge about the part so it knows what views to create and what annotations to place. Even more, it needs to know details about the geometry so it can find the geometry to attach the annotations. If the part falls into the category of a “family of parts” it will likely be best to take advantage of Inventor’s parametrics and create the drawing manually and then just update the part and drawing as described previously.

Let’s look at the different steps of creating a drawing and how you can accomplish them with the API. The AutoDrawing sample demonstrates these steps by creating the drawing below.



Creating Sheets

The first step in creating any drawing from scratch is creating the sheet. The API provides full support for creating a sheet and editing an existing sheet so you can create a sheet of any size. You can also add a border and title block and specify any required text input for those.

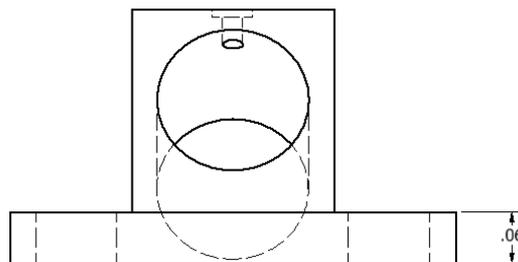
Creating Views

The API supports creating most view types. Using the API, you can get the overall size of a model and use that to help in determining the scale of the views. Depending on the types of views you're creating, the view creation process can be very straightforward or more complicated. For example, creating a base view just involves specifying the model to place, the scale, orientation, style, and its position on the sheet. Optionally you can supply additional information like the name to assign to the view, which representations to use, a custom orientation, if a sheet metal part should be folded or flat, or the member name of an iPart or iAssembly. Some of the other view types are more complicated to create because they require more input. For example, a section view requires a sketch to be drawn that represents the section line.

When specifying the document to place, it's passed to the `AddBaseView` method as a Document object, not a filename, which means you need to open the document first. Typically, you'll open it invisibly. This is what Inventor is doing internally when you create a drawing document.

Creating Dimensions

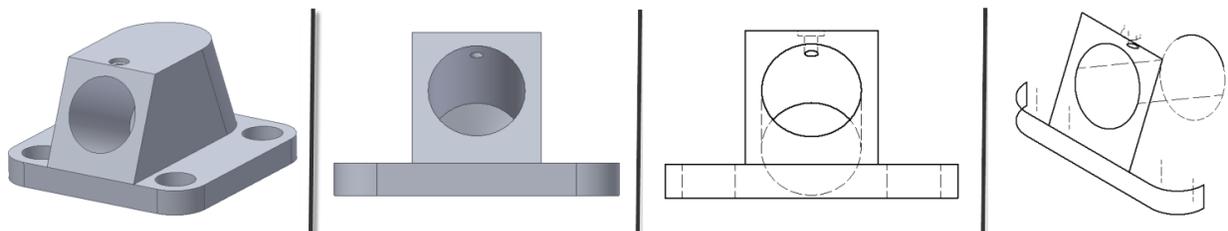
Creating dimensions is probably the most difficult part of the process. Let's look at the simple example below where we want to place the dimension to show the thickness of the base.



When I manually create this dimension, I select a curve at the bottom of the base and a curve at the top of the base and position the dimension. Very straightforward and easy. The method to place a dimension using the API is about as simple where you need to supply the two curves as `GeometryInput` objects and the position of the dimension. The problem is how to get the two curves. There are several approaches that can be used to place the dimension shown above.

Naming Curves – The first approach, and the one demonstrated in the AutoDrawing sample is to name the curves in the model and then use those names to find them later in the drawing. The sample uses *Attributes* to “name” the curves. Attributes are a way to add information to any entity in Inventor. Once you’ve added attributes you can quickly find those entities by using a search. iLogic recently added the ability to name entities. What they did was just provide a naming UI but internally it’s using the attribute functionality that has been in Inventor since Inventor 5. An alternative UI that provides access to the full capabilities of attributes is the **Attribute Manager** Utility, which you can get here: https://ekinssolutions.com/attribute_helper/. The “Working with Attributes” topic in the “Inventor API User’s Manual” under the “Custom Data” subtopic provides a good introduction. With Inventor 2020 the Attribute Manager program is being delivered as part of the **Developer Tools** installer as discussed above but you’ll always get the most recent version from my website.

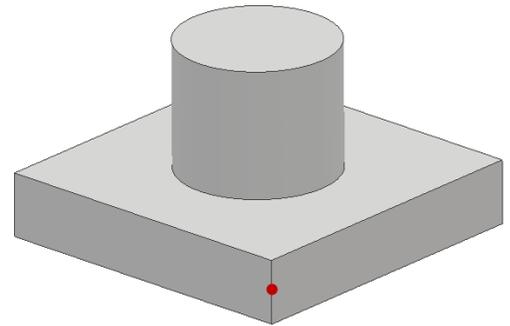
For the AutoDrawing sample, the Attribute Manager utility was used to add attributes to various edges in the model and those attributes are used to find the edges during the dimensioning process. This is straightforward except for determining which edges to add attributes to. The figures below from left to right, show the model, the model as viewed from the direction the view will be, the drawing view, and a rotated view of the drawing view. We can see in the rotated view that only the edges that are visible in that view are displayed. That means that only those edges are available when adding dimensions to that view. If you add an attribute to an edge on the back of the part and then try to get the drawing curve associated with that edge it will fail for that view because there isn’t a drawing curve that represents that edge.



To use edges from the part in the drawing you use the `DrawingView.DrawingCurves` method. This method returns all the drawing curves in a view. However, there is an optional argument where you can specify which curve to return by providing a model entity. In this case, I can provide the edge from the model and if there is a corresponding drawing curve, the method will return it. When using the named entity approach, you use the attribute to find the edge in the part or assembly and then use that edge as input to the `DrawingCurves` method. Once you have the needed drawing curves you can create the `GeometryIntent` objects and place the dimension.

The most common use of attributes is to add an attribute to an entity and then use that attribute to find it later, essentially naming the entity. But they’re much more powerful than that because you can add additional information besides just a name and can search for specific attributes by name and value.

Finding Curves – A second approach is to calculate the position where the curve is in the part or assembly space and then get the equivalent location in sheet space and use the API to find any curves at that location. Here's an example with a simple part. The coordinate in model space of the point indicated to the right is (5, 0, 0.5) in centimeters. Remember that the API ALWAYS uses centimeters because that's what Inventor uses internally. The code below finds a curve at that location in the front view.



```
' Get the drawing view to add the dimensions to.
Dim drawView As DrawingView = sht.DrawingViews.Item(1)

' The size of the part is known or you could use the API
' to query the model and look at parameters or part geometry
' to determine the size. In this example, it's already known.

' Get a point on the sheet that's equivalent to the model location.
Dim modelPoint As Point = tg.CreatePoint(5, 0, 0.5)
Dim sheetPoint As Point2d = drawView.ModelToSheetSpace(modelPoint)

' Find any curves at that location on the sheet.
Dim foundCurves As ObjectsEnumerator = sht.FindUsingPoint(sheetPoint)

If foundCurves.Count = 1 Then
    ' The found curve is a drawing curve segment so we need to get the drawing curve.
    Dim curve As DrawingCurve = foundCurves.Item(1).Parent

    ' Create the GeometryIntent objects using the curve.
    Dim geom1 As GeometryIntent
    Dim geom2 As GeometryIntent
    Set geom1 = sheet.CreateGeometryIntent(curve, PointIntentEnum.kEndPointIntent)
    Set geom2 = sheet.CreateGeometryIntent(curve, PointIntentEnum.kStartPointIntent)

    ' Calculate the position of the dimension text and create the dimension.
    Dim txtPoint As Point2d = sheetPoint.Copy
    txtPoint.X = txtPoint.X + 1
    sheet.DrawingDimensions.GeneralDimensions.AddLinear(txtPoint, geom1, geom2)
End If
```

Using WorkPoints – A third approach to adding dimensions is to not use the model geometry at all but instead to dimension to work points. Actually, it's not dimensioning to work points but is dimensioning to center points, which are created by importing a work point. Using work points for dimensioning is a very common approach and is the simplest and most straightforward of the three approaches. This technique involves adding work points to the model at any location that you want to dimension to. The names of the work points will be used to find them later.

A big advantage of this approach is that you don't run into the issue described above where only some curves are visible in certain drawing views. The work points will always be available regardless of the view orientation. The sample below illustrates this workflow by creating a dimension between the work points named "DimPoint1" and "DimPoint2".

```

' Get the model referenced by the drawing view.
Dim partDoc As PartDocument
Set partDoc = drawView.ReferencedDocumentDescriptor.ReferencedDocument

' Get the two work points in the model.
Dim wp1 As WorkPoint = partDoc.ComponentDefinition.WorkPoints.Item("DimPoint1")
Dim wp2 As WorkPoint = partDoc.ComponentDefinition.WorkPoints.Item("DimPoint2")

' Create center marks for the two work points.
Dim cm1 As Centermark = sheet.Centermarks.AddByWorkFeature(wp1, drawView)
Dim cm2 As Centermark = sheet.Centermarks.AddByWorkFeature(wp2, drawView)

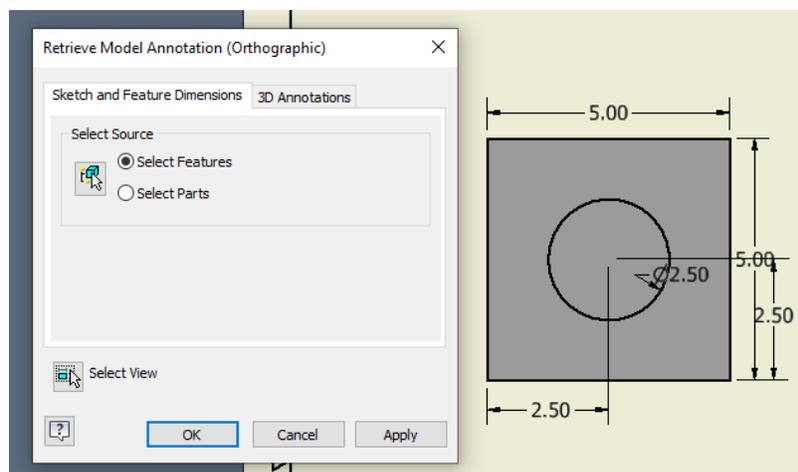
' Create GeometryIntent objects for the work points and add the dimension.
Dim geom1 As GeometryIntent = sht.CreateGeometryIntent(cm1)
Dim geom2 As GeometryIntent = sht.CreateGeometryIntent(cm2)
Dim txtPoint As Point2d = tg.CreatePoint2d(10, 15)
sht.DrawingDimensions.GeneralDimensions.AddLinear(txtPoint, geom1, geom2)

' Turn off the visibility of the center marks.
cm1.Visible = False
cm2.Visible = False

```

Of course, using work points only works when creating linear or angular dimensions. If you need to place a diameter or radial dimension you need the curve.

Retrieving – A fourth approach is to retrieve dimensions that already exist in the model. These are dimension constraints that were added to a sketch or model dimensions that were added automatically by Inventor when features were created. This is the equivalent of the **Retrieve Model Annotation command**, as shown below. In the API, you use a combination of the **GetRetrievableDimensions** and **Retrieve** methods on the **GeneralDimension** object.



In addition to retrieving dimensions from the model, you can also retrieve dimensions from a sketch that's associated with a drawing view. When creating dimensions, they must be attached to geometry. It's not possible to create a dimension between two arbitrary points in space. A workaround to this limitation is to create a sketch on the view where you want the dimension and create sketch points at those locations, draw a dimension constraint between the points and then retrieve that dimension into the drawing.

Associative Draft View

A capability of the API that is rarely used is something called an *Associative Draft View*. What this does is let you create a new drawing view that references a part or assembly but no geometry is drawn in the drawing view. It's an empty drawing view with the idea that you'll create a sketch in the drawing view and draw the geometry yourself. This is typically the most useful for complex assemblies but can also be used for parts. The problem this solves is when the actual view of the assembly as Inventor would create it isn't what you want. For example, you may have a complex assembly that if you create a standard drawing view becomes too busy to be very useful in a drawing. With this approach, you redraw the assembly using sketch geometry.

It can also be used to draw a completely different representation of the model. For example, you might have a piping model and then use this to draw a schematic that describes the model.

This is referred to as an "associative" draft view because there is an event that notifies you when changes are made to the original model so you can redraw the draft view so it's up to date with the model.

The SimpleLayout sample program demonstrates the associative draft view functionality.