

IronCAD Automation

1 : Introduction

Automation in IronCAD is becoming a very powerful part of the product and enables users to create complicated shapes with the click of a button or to help with repetitive tasks. IronCAD automation is most often done through Visual Basic which this document discusses, but can also be automated through Visual C++ and even Excel. Throughout the next few sections everything from setting up a Visual Basic environment to program IronCAD in to creating shapes will be touched on. Below is a list of fonts and what they'll mean in this document :

File Open...	Menu actions
<i>'Reference'</i>	Words associated with Visual Basic
<code>j = j + 1</code>	Visual Basic code

Now lets jump into the revolution!

2 : Setting Up The Visual Basic Environment

In order to automate IronCAD a new Visual Basic project needs to be created, along with *'referencing'*¹ IronCAD. To reference IronCAD go to **Project | References...** which will open the references for the current project. There will be many IronCAD entries in the list that is shown, but scroll down to the bottom of the entries and look for the entry "IronCAD 3.0 Object Library" then place a check mark next to it and hit OK. To view what is able to be automated in IronCAD take a look in the object browser by going to **View | Object Browser**. Once in the Object Browser there will be a combo box at the top that says "<All Libraries>" so click on it and change it to "IronCAD." This will display all the

¹ *'referencing'* is the way Visual Basic accesses libraries form another program or DLL. A reference is most often given to a Type Library (*.tlb) or a DLL.

classes in IronCAD that have been made available to users. Now Visual Basic is set up to automate IronCAD and code can start being added.

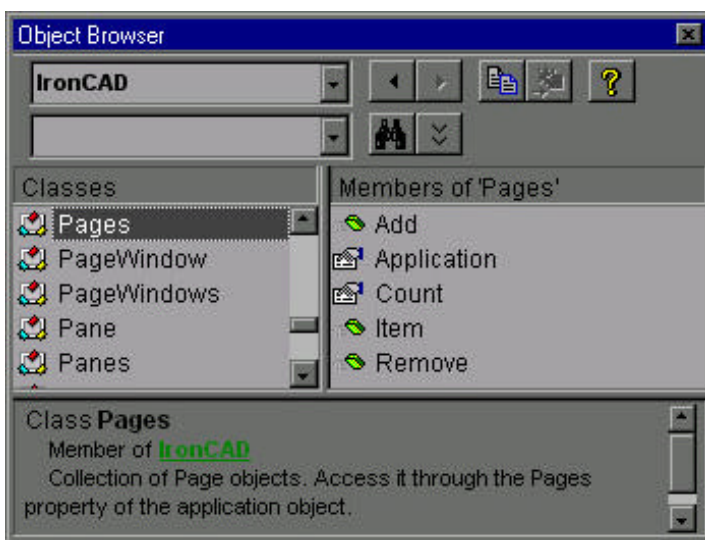
3 : Getting Access to IronCAD

In order to automate IronCAD the application object has to be set. What this essentially means is that two lines of code now have to be added to the Visual Basic project now for anything to happen. This application object will be the basis for everything that happens in automation as it concerns IronCAD. Below are the lines of code that do set the application object :

```
Dim objApplication As Object  
Set objApplication = GetObject(, triApp)
```

The first line declares the variable 'objApplication' and the second of line sets the variable to the 'Application' class in IronCAD. In the Object Browser under 'IronCAD' on the left side there will be a 'class²' called 'Application' and essentially that is what the variable is now. So for instance if we wanted to know the number of open scenes we could do the following :

```
Dim objApplication As Object  
Dim objPages As Object  
Dim lngPages As Long  
  
Set objApplication = GetObject(, triApp)  
Set objPages = objApplication.Pages  
lngPages = objPages.Count
```



Above we set 'objPages' equal to 'objApplication.Pages' on the second line of code after the variables are declared. Take a look in the Object Browser again, and click on the 'Application' class. On the right side you'll see a property called 'Pages' which is what we've set 'objPages' to. The help text for 'Pages' says

++. It's a type of variable such as a string or integer that has properties, methods and functions associated with it.

that it's the collection of pages and it's an object. So exactly what do we have now? Take a look in the object browser again and this time scroll down on the left side until 'Pages' is visible and click on it. Now on the right side of the object browser it shows the properties for this class. So on the last line when we set 'lngPages' equal to 'objPages.Count' the property 'Count' on the 'Pages' class was set to the variable 'lngPages'. To get used to working with the classes, below is a quick program to save a the active scene to a file :

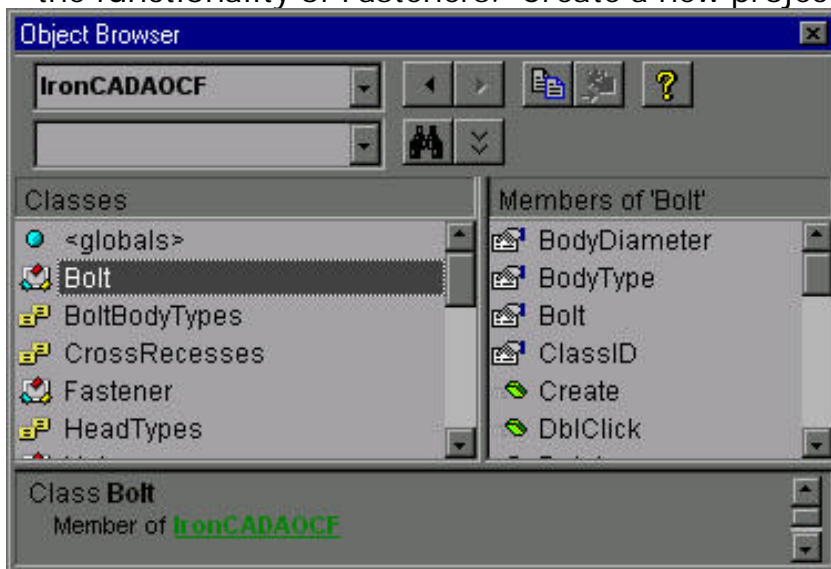
```
Dim objApplication As Object
Dim objPage As Object
Dim strFilename As String

Set objApplication = GetObject(, triApp)
Set objPage = objApplication.ActivePage
strFilename = "C:\Test.tmd"
objPage.SaveAs strFilename
```

There's a new property above on the application object – 'ActivePage'. This property returns the active page in IronCAD which is what is most commonly wanted. The last line 'objPage.SaveAs strFilename' saves the scene the the filename specified on the line above. This should serve as a fairly strong base for getting started so now lets create a bolt!

4 : Creating A Fastener

The first thing that needs to be done before a fastener is created is that a reference needs to be added to the Fastener DLL. This will provide access to all the functionality of Fasteners. Create a new project then reference IronCAD



again and also place a check mark next to the item 'IronCAD – Fasteners (Tool)' to reference Fasteners. In the Object Browser now, there will be a new entry called 'IronCADAOCF' which is the Fasteners exposed classes. On the left side

of the object browser there will be a class for each type of fastener that there is in the Fastener tool. On the right side all the properties and methods are displayed for the particular fastener that has been clicked on along with help text to guide the user. So for an example let's create a bolt who's length is 3 inches and who's nominal size is 0.3 inches, has a round-square head and has a slotted castle nut on the bottom of it :

```
Dim objApplication As Object
Dim objActivePage As Object
Dim objTopShape As Object
Dim cBolt As New Bolt

Set objApplication = GetObject(, triApp)           '1
Set objActivePage = objApplication.ActivePage     '2
Set objTopShape = objActivePage.Shape             '3
cBolt.HeadType = HeadTypes.RoundSquare           '4
cBolt.Length = 3                                  '5
cBolt.NominalSize = 0.3                           '6
cBolt.Nut = NutTypes.SlottedCastle                '7
cBolt.Create objTopShape                          '8
```

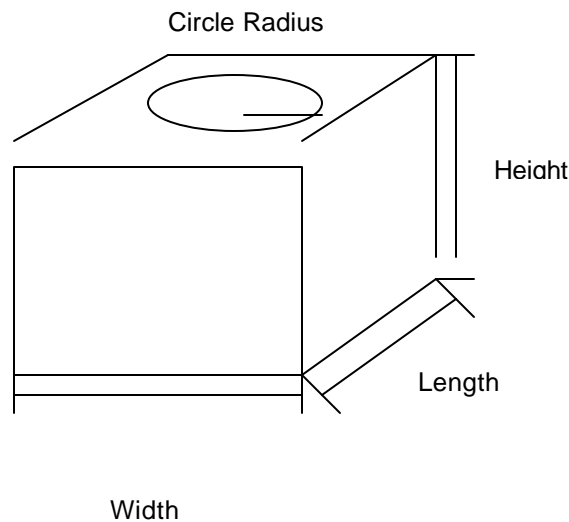
That's all there is to it. Other properties such as position, washer, whether it's metric or not can be set as well but for this example we'll keep it brief and discuss what is above.

The first new thing that is in this code is the 4th declaration near the top of the code 'Dim cBolt As New Bolt' – this code sets 'cBolt' equal to the 'Bolt' class that appears in the Fastener class library. The next new line of code we come across is on line #3 'Set objTopShape = objActivePage.Shape' which is where we set the top shape in the scene. The top shape is the shape that every shape is under at the scene level. Perhaps it could be thought of as the root level in the scene browser. The reason we set the top shape is because we need to specify a shape that the fastener is under. Line #4 through line #7 set the various properties of the bolt such as it's length, the type of head it has, nominal size and the nut. On Line #4 the type of head is set to 'HeadTypes.RoundSquare' which is a constant. 'HeadTypes' appears in the object browser as an enumerated type and basically they are just constants so the user knows what type of head they are setting, and the same is for the 7th line. The nominal size and length of a fastener must always be specified in inches because of the ANSI library it uses for dimensions. So for instance you'd like a

metric bolt with a length of 25.4 millimeters, you'll have to set the length to 1 since its length in inches is one inch, but set its Metric property to True and the conversion will happen internally. To set its metric property add the following line of code `'cBolt.Metric = True'`. Fasteners is actually another Visual Basic project that we've referenced so we still haven't actually created any shapes on our own. So for our big dive into IronCAD's automation system we'll create a program that allows a user to specify the size of a block they'd like to create, along with the dimension of a hole in the middle.

5 : Creating Your First Shape

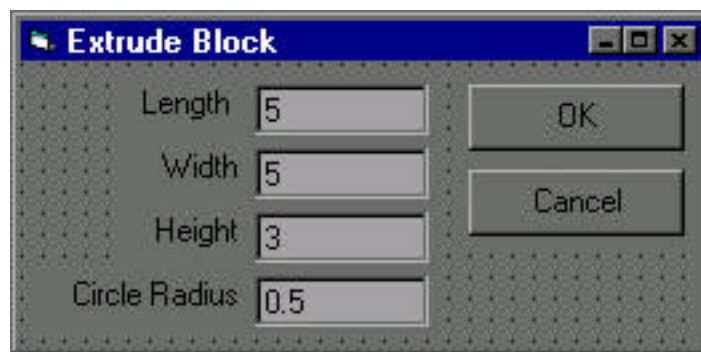
In this example we'll create a dialog allowing the user to specify the dimensions of a shape like the one pictured below with the various dimensions. We'll use the shape below to aid in our conversation of this example.



First we need to create the form that will gather the input. Create a form that looks similar to the one displayed on the next page. It should have 4 labels, 4 text boxes and 2 command buttons. The list below explains which properties should be set to each :

<i>Form Object</i>	<i>Name</i>	<i>Caption / Text</i>
Form	UIForm	Extrude Block
Text1	txtLength	5

Text2	txtWidth	5
Text3	txtHeight	3
Text4	txtRadius	0.5
Label1	lblLength	Length
Label2	lblWidth	Width
Label3	lblHeight	Height
Label4	lblRadius	Circle Radius
Command1	cmdOK	OK
Command2	cmdCancel	Cancel



Once the form is set up we can begin adding code to the 'OK' button that will become our program. So first, double-click the 'OK' button which will bring you to a page of text that will look similar to this :

```
Private Sub cmdOK_Click()  
  
End Sub
```

We're going to put our code between these two lines of code. So lets begin first by declaring a few variables. We know we'll need the application object, we'll need a variable for each dimension, we'll need the top shape. So lets add the declarations first...

```
Dim objApplication As Object           'The Application Object  
Dim objPage As Object                 'The Active Page  
Dim objTopShape As Object             'The Tope Shape  
Dim dblLength As Double               'The Length  
Dim dblWidth As Double                'The Width  
Dim dblHeight As Double               'The Height  
Dim dblRadius As Double               'The Hole Radius
```

Next we'll need to get the values from the form so let's do that...

```
dblLength = Val(UIForm.txtLength.Text)
dblWidth = Val(UIForm.txtWidth.Text)
dblHeight = Val(UIForm.txtHeight.Text)
dblRadius = Val(UIForm.txtRadius.Text)
```

After that's done, it's time to set the application object, the active page, and the top shape so add the 3 lines of code to do that...

```
Set objApplication = GetObject(, triApp)
Set objPage = objApplication.ActivePage
Set objTopShape = objPage.Shape
```

So far so good. Now we need to talk about adding a shape to the scene. In order for us to create an extrude, first we need to create a part to put it in. So what we're going to do is add a part to the scene, then put the extrude shape that we will be creating inside that part. First we need to declare a couple more variables, so put these declarations back at the top with the rest...

```
Dim objTopShapeHistory As Object
Dim objPart As Object
Dim objPartHistory As Object
Dim objExtrude As Object
```

And now add the following code at the bottom of the last code we added...

```
Set objTopShapeHistory = objTopShape.Components(triGroup).Item(1)
Set objPart = objTopShapeHistory.Shapes(triCompoundShape).Add
Set objPartHistory = objPart.Components(triHistory).Item(1)
Set objExtrude = objPartHistory.Shapes(triExtrudeShape).Add
```

Whew! That's a big jump from what we've added in the past, so let's take a second to look at it. On the first line the variable 'objTopShapeHistory' is set to 'objTopShape.Components(triGroup).Item(1)', and the third line is very similar to this line, only the component is 'triHistory' instead. Each shape has several components that are associated such as Group, History, Position, Anchor, TriBall, Surface Finish, and more. All together there are about 30 different components, we'll only discuss a few here though. The group and history components return the 'GroupComponent' and 'HistoryComponent' classes of

IronCAD respectively, while the position component returns the generic 'Component' class. On the second line we added a part to the scene. When adding IntelliShapes, they must be added under a part. Not only must they be added under a part, but they must be added under the parts history component. So on line 3 we set the history component of the part to the variable 'objPartHisotory' and on line 4 we add the actual extrude IntelliShape to the part.

The next thing we need to do is set the IntelliShape's anchor at 0, 0, 0 along the sizebox of the IntelliShape. To do this, first we need to get the Anchor component which is the generic 'Component' class in IronCAD. To do this we'll need to declare a new variable again, and then set the anchor component. So first declare the variable up with the others declarations :

```
Dim objAnchor As Object
```

And then set the anchor component...

```
Set objAnchor = objExtrude.Components(triAnchor).Item(1)
```

Now we have successfully set 'objAnchor' to the anchor of the new extrude. What we need to do though is set the position of the anchor to 0, 0, 0 so lets do that with the following 3 lines of code :

```
objAnchor.Variable(triX).Value = 0  
objAnchor.Variable(triY).Value = 0  
objAnchor.Variable(triZ).Value = 0
```

Now the position of the anchor along the shape is 0, 0, 0. Let's look at 'objAnchor' though – it has 'Variable(triX).Value' appended to it so why is that? Don't forget that 'objAnchor' is the generic class 'Component' so if we look in the Object Browser at the 'Component' class we see that it has 4 properties – Application, Shape, Type, and Variable. So essentially we're setting a variable on the anchor, but Variable is a class of its own and if we take a look at the 'Variable' class in the Object Browser we'll see that it has 5 properties – Application, Expression, Lock, Parent and Value. This enables us to set an expression such as "(Sizebox\Length * 0.5)" or just a normal value like we did.

Okay, the anchor is set now the next thing that we need to do is set how far it needs to be extruded. To do this we need to get the extrude shapes Sweep component. Every IntelliShape except for a loft shape has a sweep component which is where a lot of variables such as extrude depth, IntelliShape shelling, and whether it's a solid or it's a hole are kept. So, first we need to declare another variable with the other declarations :

```
Dim objSweepComponent As Object
```

And then we need to set the sweep component...

```
Set objSweepComponent = objExtrude.Components(triSweep).item(1)
```

The sweep component is also a generic 'Component' class. It is fairly easy to recognize if a component is a generic class, if you don't see the name of the component as a class on the left side of the Object Browser, then it is the generic 'Component' class. Now lets set the extrude height :

```
objSweepComponent.Variable(triExtrudeDistUp).Value = dblHeight
```

Notice that we're setting the extrude distance up. We could also extrude the shape down by replacing 'triExtrudeDistUp' with 'triExtrudeDistDown'. We're setting the extrusion depth to the form value for height as well.

The final thing we need to do (not to make it sound minimal ☺) is to get the profile of the shape and add the path of the circle and the path of the box. To do this we'll go ahead and add the last variable declarations now with the others, so add the following lines of code to the previous declarations :

```
Dim objProfileComponent As Object  
Dim objProfile As Object  
Dim objBoxPath As Object  
Dim objCirclePath As Object
```

So now that we've declared the last variables, we should finish this up, right? Well, first we need to set the profile component. This is where the profile for the extrude shape is kept, and then we'll set the profile :

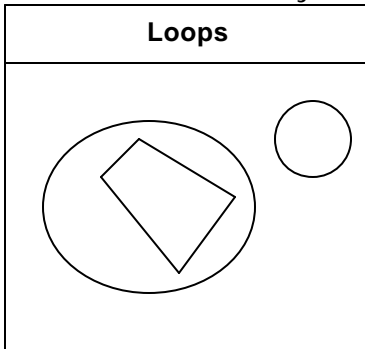
```

Set objProfileComponent = objExtrude.Components(triProfile).Item(1)
Set objProfile = objProfileComponent.Profile

```

On the first line we set 'objProfileComponent' to the 'ProfileComponent' class of IronCAD. This time the component is not the generic 'Component' class, and on the left side of the Object Browser the class 'ProfileComponent' will be there along with 5 properties – Application, Profile, Shape, Type and Variable. So on the second line we set 'objProfile' equal to the Profile of the profile component.

Now, it's necessary to talk about the concept of loops before we continue. In



the picture to the left there are 3 loops. To create the profile geometry in IronCAD a path is added. There needs to be a path added for each loop. If more than one loop exists on a single path then the shape will be invalid and a default shape will be created – the dreaded block. Of course that's great if you wanted a block 😊 So with that being said, we know that there needs to be two paths because our block is one loop

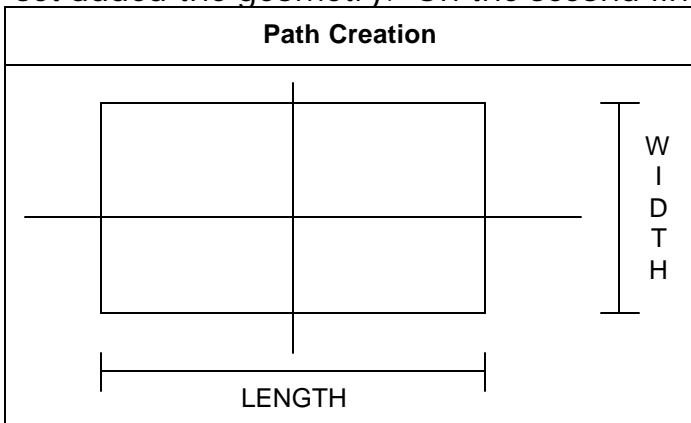
circle is another loop. Here's the code we need to add in order to create the correct path for our extruded block with the hole in the middle :

```

Set objBoxPath = objProfile.Components(triPath).Add
objBoxPath.SetStartPoint -(dblLength / 2), (dblWidth / 2)
objBoxPath.AddLineTo (dblLength / 2), (dblWidth / 2)
objBoxPath.AddLineTo (dblLength / 2), -(dblWidth / 2)
objBoxPath.AddLineTo -(dblLength / 2), -(dblWidth / 2)
objBoxPath.AddLineTo -(dblLength / 2), (dblWidth / 2)

```

Now let's look at what we just did. We added the outside path for the box, and set added the geometry. On the second line we set the point to start adding



corner of the block, then we add picture below to better understand We started at the top left and corner. Then from there we added to the lower left hand corner and now the next step is to add the circle

path to the profile so that the hole is in the block. To do this we need to add another path and then create the geometry for the circle. The following lines of code will do just that :

```
Set objCirclePath = objProfile.Components(triPath).Add
objCirclePath.SetStartPoint 0, 0
objCirclePath.AddEllipse 0, 0, dblRadius
```

On the first line just like we did before we added the new path. Next the start point was set and finally the circle was added by the radius 'dblRadius' that was specified from the form. Everything that needs to be done to define the shape geometry is completed now the shape has to be updated and the sizebox reset along with updating the page so the new shape gets rendered so for our final three lines of code we that is what happens :

```
objExtrude.Update
objExtrude.ResetSizebox
objPage.Update
```

Finished! The form could also be unloaded since we the creation is finished since there is no need to keep it around so lets add the line of code to do that :

```
Unload UIForm
```

Now what we need to do is compile it into an EXE and place it on the **Tools** menu so that it can be distributed. So first lets build the EXE by going to the menu **File | Make Extrude.exe...**, then pick a directory and name to make it in and click 'OK.' Once that is completed successfully (there will probably be a few compile bugs the first time - nothing works 100% the first time you know?) go to IronCAD and then go to **Tools | Add-on Tools...** to bring up the Add-on Tools configuration dialog box. Next click the 'Add' button since we are adding a new tool, and once the 'Add Tool' dialog appears, click the 'Executable' option the type in the full path and filename of the executable that was just made. Click 'OK' and once the dialog disappears and the 'Add-On Tools' re-appears it's possible to change the menu text. When everything is done click the 'Close' button to return to IronCAD. Now after all this hard work lets try it. Go to **Tools | <Your Tool Name>** and run the application.

6 : Clean Up

Hopefully this has been helpful in understanding more about automation in IronCAD. As Add-On tools get more complicated a lot more attention to units and positioning have to happen. These examples didn't worry about units because when drawing a profile the geometry is in the scene units. When Add-Ons start dealing with surface finishes and methods of positioning units have to be in centimeters. The most common mistakes that happen with automation in IronCAD result from components not existing or values set in the wrong units. In the previous examples it wasn't necessary to worry about whether a component existed because it was a critical component and it was always assumed that it did. But for example if we were applying a surface finish to the part, and we had used the following line :

```
Set objSurfaceFinish = objPart.Components(triSurfaceFinish).Item(1)
```

it would have immediately crashed on the next line since 'objSurfaceFinish' would not have been set. The reason is because the surface finish component isn't added by default when a shape is created, first we would have needed to add it. But for safety precautions always use the following syntax when working with components :

```
If (objShape.Components(<COMPONENT>).Count = 0) Then
    Set objComponent = objShape.Components(<COMPONENT>).Add
Else
    Set objComponent = objShape.Components(<COMPONENT>).Item(1)
End If
```

This ensures that the variable 'objComponent' will be set for the next line. As far as the values being set in the wrong units, there are several functions in IronCAD that require units in Centimeters such as the 'Shift', 'GetPointTransformUp' and 'GetPointTransformDown' methods on the shape class. The advice on these is that if you get an unexpected result with positioning, try metric units.

'The complete code for the Extrude Example

```
Private Sub cmdOK_Click()  
  
    'First set of declarations  
    Dim objApplication As Object           'The Application Object  
    Dim objPage As Object                 'The Active Page  
    Dim objTopShape As Object            'The Tope Shape  
    Dim dblLength As Double              'The Length  
    Dim dblWidth As Double              'The Width  
    Dim dblHeight As Double             'The Height  
    Dim dblRadius As Double             'The Hole Radius  
  
    'Second set of declarations  
    Dim objTopShapeHistory As Object  
    Dim objPart As Object  
    Dim objPartHistory As Object  
    Dim objExtrude As Object  
  
    'Third declaration  
    Dim objAnchor As Object  
  
    'Fourth declaration  
    Dim objSweepComponent As Object  
  
    'Final declarations  
    Dim objProfileComponent As Object  
    Dim objProfile As Object  
    Dim objBoxPath As Object  
    Dim objCirclePath As Object  
  
    'Get the form values  
    dblLength = Val(UIForm.txtLength.Text)   'Set length  
    dblWidth = Val(UIForm.txtWidth.Text)     'Set the width  
    dblHeight = Val(UIForm.txtHeight.Text)   'Set the height  
    dblRadius = Val(UIForm.txtRadius.Text)   'Set the circle radius  
  
    'Set the application, active scene and top shape  
    Set objApplication = GetObject(, triApp)  
    Set objPage = objApplication.ActivePage  
    Set objTopShape = objPage.Shape  
  
    'Add the compound and extrude shape  
    Set objTopShapeHistory = objTopShape.Components(triGroup).Item(1)  
    Set objPart = objTopShapeHistory.Shapes(triCompoundShape).Add  
    Set objPartHistory = objPart.Components(triHistory).Item(1)  
    Set objExtrude = objPartHistory.Shapes(triExtrudeShape).Add  
  
    'Set the anchor and position it  
    Set objAnchor = objExtrude.Components(triAnchor).Item(1)  
    objAnchor.Variable(triX).Value = 0  
    objAnchor.Variable(triY).Value = 0  
    objAnchor.Variable(triZ).Value = 0  
  
    'Set the extrude distance  
    Set objSweepComponent = objExtrude.Components(triSweep).item(1)
```

```
objSweepComponent.Variable(triExtrudeDistUp).Value = dblHeight

`Set the profile
Set objProfileComponent = objExtrude.Components(triProfile).Item(1)
Set objProfile = objProfileComponent.Profile

`Set the paths
Set objBoxPath = objProfile.Components(triPath).Add
objBoxPath.SetStartPoint -(dblLength / 2), (dblWidth / 2)
objBoxPath.AddLineTo (dblLength / 2), (dblWidth / 2)
objBoxPath.AddLineTo (dblLength / 2), -(dblWidth / 2)
objBoxPath.AddLineTo -(dblLength / 2), -(dblWidth / 2)
objBoxPath.AddLineTo -(dblLength / 2), (dblWidth / 2)

Set objCirclePath = objProfile.Components(triPath).Add
objCirclePath.SetStartPoint 0, 0
objCirclePath.AddEllipse 0, 0, dblRadius

`Update the shape, reset its sizebox and update the page
objExtrude.Update
objExtrude.ResetSizebox
objPage.Update

`Unload the form
Unload UIForm
```

End Sub

Shape Component System (Common components and Variables)

