

# Fasteners

## Introduction :

The purpose of this document is to provide a technical description of Fasteners pertaining to its class structure and how it interacts with IronCAD. This document will also discuss the basics of Automation in IronCAD along with providing a couple of examples. This document should be a great help to the novice programmer or to the professional programmer wanting to create complex Add-Ons to IronCAD. It is assumed that the user understands the Visual Basic 5.0 environment and language, along with creating references to type libraries. This document only covers the 3D scene and not 2D layout.

## Automation Overview:

In order to understand Fasteners, first there needs to be an understanding of the shape system in IronCAD. Each shape is a child under what is considered the “top shape” which is the page, or scene that’s currently open. In order to get the top shape, first the application needs to be set using GetObject(). Lets use the following piece of code for discussion :

```
Dim objApp As Object           `1
Dim objPage As Object         `2
Dim objTopShape As Object     `3

Set objApp = GetObject(, triApp) `4
If (objApp.Pages.Count > 0) Then `5
    Set objPage = objApp.ActivePage `6
    Set objTopShape = objPage.Shape `7
End If                         `8
```

The code above shows how to get the active page in IronCAD if there is one and then get it’s top shape. Line #4 shows how to get the IronCAD application, which returns the “Application” class in IronCAD’s type library. Once we have the application we can determine the number of scenes open currently as demonstrated on line #5. So that there are no errors down the line, if there is currently a page open then, Pages.Count will be greater than zero, and we’ll set objPage to the active page in IronCAD. Once the page is set we use line #7 to set the top shape of the application.

Each shape has either a group or history component which allows the sub-shapes to be set. Shapes in the root level of the scene browser are located in the group component of the top shape. Parts within assemblies are located in the assembly’s group component, and Intellishapes® are located in the history component of a part. So, for the next example, we’ll use a picture of the scene browser and code to get a visual interpretation of how components work. We’ll use ‘objTopShape’ from the example above and assume that that’s the top shape of the current page.

```

Dim objAssy As Object '1
Dim objPart As Object '2
Dim objIntelliShape As Object '3
Dim objGroupComponent As Object '4
Dim objHistory As Object '5

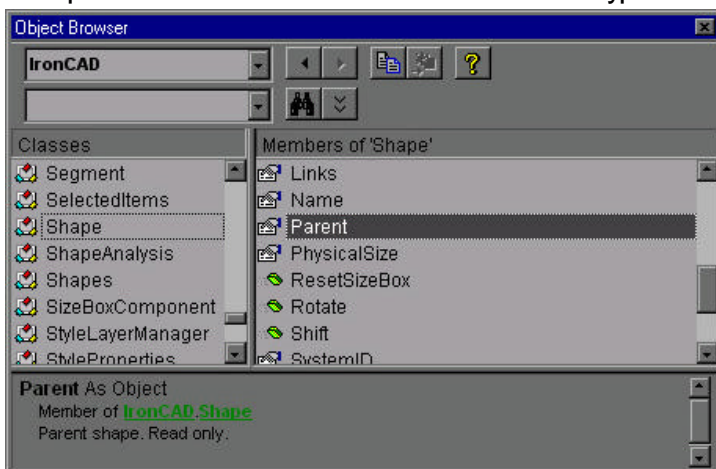
Set objGroupComponent = objTopShape.Components(triGroup).Item(1) '6
Set objAssy = objGroupComponent.Shapes.Item(1) '7
Set objGroupComponent = objAssy.Components(triGroup).Item(1) '8
Set objPart = objGroupComponent.Shapes.Item(1) '9
Set objHistory = objPart.Components(triHistory).Item(1) '10
Set objIntelliShape = objHistory.Shapes.Item(1) '11

```



Now, let's explain. On line #6, we set the group component of the assembly in the scene. This is equivalent to the component class in the IronCAD type library. There are many component types, but for right now we'll concentrate on the history and group components. Once we have the group component, we are able to get the shapes within it. So, on line #7 we set the assembly shape which is shape 1 since there is only one shape in the top shapes group component, the root level of the scene browser. The process of getting the part is exactly the same as getting the assembly since the assembly has a group component as well. However parts have history components, so instead of getting the IntelliShape® through the group component of a part, the IntelliShape® has to be set through the history component of the compound shape, which is why 'triHistory' is used instead of 'triGroup' on line #10.

That's a brief overview of the shape system as it relates to going down the tree, but there's often a need to get the parent of a shape. In fasteners the parent of the shape is set often so that based on what type of shape it is, a different action is



performed. Unfortunately, it is not possible to get the parent of a shape using the standard IronCAD type library. If we take a look at the shape class in IronCAD's type library for a second, we notice it has a property called 'Parent', however this is not the owner of the history or group component that the shape is in, this is the definition shape. Simply, the shape that appears in the scene is a mirrored image of the definition shape, and any actions

we perform on a shape happen to it's definition shape, not the shape itself. So, in order to get the parent shape a utility DLL had to be made. In order to use the utility DLL, in Visual Basic 5.0 add a reference to 'IronCAD – Tool Utilities'. This DLL contains a few helper functions that have been decided to be public since many have been asked for,

however the code is not being released so a brief definition of it's classes are listed below.

First of all, once the type library is referenced, it will appear as 'IRONCADA0' in the object browser. All of the classes are able to be created, so in order to use any of them you must declare a variable as a new class. In the example below, from the Intellishape® we set in the previous example, we'll get the part that it belonged to :

```
Dim objSceneUtils As New SceneUtils
Dim objPart As Object

Set objPart = objSceneUtils.GetShapeParent(objIntelliShape)
```

Well, that was pretty easy wasn't it? Another useful function is GetShapePath which returns the full path of a shape in the scene. As Add-Ons get more complicated in IronCAD, there needs to be a great amount of attention paid to units. There is a class called 'UnitUtils' that helps a lot with this and is used vastly in Fasteners. For an example, lets say that you want to create a block that's 5"X5"X5", however you don't know the scene units. When you start to create the block you will need to specify the extrude distance and then specify the cross section in terms of the scene units.

```
Dim objUnitUtils As New UnitUtils
Dim dblSceneValue As Double

dblSceneValue = objUnitUtils.ConvertUnits(objPage, 5, UNIT_INCHES,
                                         UNIT_PAGEUNITS)
```

Above, 'objPage' is the page we got in the first example. After the code is run, 'dblSceneValue' holds the value of 5" in the current scene units. So for instance if the units of the current scene were in centimeters, 'dblSceneValue' would be 12.7, which is 5 inches.

This overview should be enough to get us started on the Fasteners Add-On. More things will come up as we go along but they'll be explained in the process.

## Fasteners :

### 1 : Initialization

The first part in understanding fasteners is understanding the initialization that happens when a user drops the “Fastener” icon from the catalog. In the catalog, there is a dummy shape (a compound shape) that exists that has an Add-On component. The Add-On component is assigned the Class ID of the ‘Fastener’ class in the Fasteners project. Let’s first take a look at the code that created the dummy shape :

```
Dim objApp As Object
Dim objScene As Object
Dim objCompoundShape As Object
Dim objAddon As Object
Dim objBehavior As Object
Dim objCatalog As Object
Dim objEntries As Object
Dim objEntry As Object

Set objApp = GetObject(, triApp) '1
Set objScene = objApp.ActivePage.Shape.Components(triGroup).Item(1) '2
Set objCompoundShape = objScene.Shapes(triCompoundShape).Add '3
Set objAddon = objCompoundShape.Components(triAddOn).Add '4
objAddon.Variable(triClassId) = "{51A63FE9-C5A2-11D1-8FF2-00A0240B6602}" '5
Set objBehavior = objCompoundShape.Components(triUIBehavior).Add '6
objBehavior.Variable(triDropEvent).Value = triExecuteAddon '7
Set objCatalog = objApp.ActiveCatalog '8
Set objEntries = objCatalog.Entries '9
Set objEntry = objEntries.Add(objCompoundShape) '10
objEntry.Label = "Fastener" '11
```

Keep in mind that this code is not in the Fasteners project, this code was temporary code to create the Fasteners catalog entry. All Add-Ons use a method similar to this one except for Pattern which is a little more complicated. From line #1 to line #2 all we do is get the application, then set the scene to the active page. On line #3 we add a compound shape in the scene. This compound shape has no Intellishapes® in it, it is purely a void shape, in reality it is nothing except for an anchor and a sizebox, with no geometry associated with it. On line #4 we get add an Add-On component to the shape and set it to objAddOn. This adds the Add-On component to a shape. On line #6 the ClassID is specified. This is the class ID of the ‘Fasteners’ class in the Fasteners project. Next we need to add a behavior to the shape so that when it’s dropped it calls the Add-On DLL which is what happens in lines 6 and 7. So what is essentially done is that we’ve specified it to execute the Add-On on when a user drags and drops it from the catalog. Finally, lines 8 through 11 simply add the compound shape to the active catalog, and change it’s caption to “Fastener”. So, now when a user drops the Fastener from the Catalog it calls the class “Fastener” and looks for the “Drop” function in the class. In the Fastener project the following sub routine is in the ‘Fastener’ class :

```
Public Sub Drop(objFastener As Object, X As Long, Y As Long)

    Initialize.OnDrop objFastener

End Sub
```

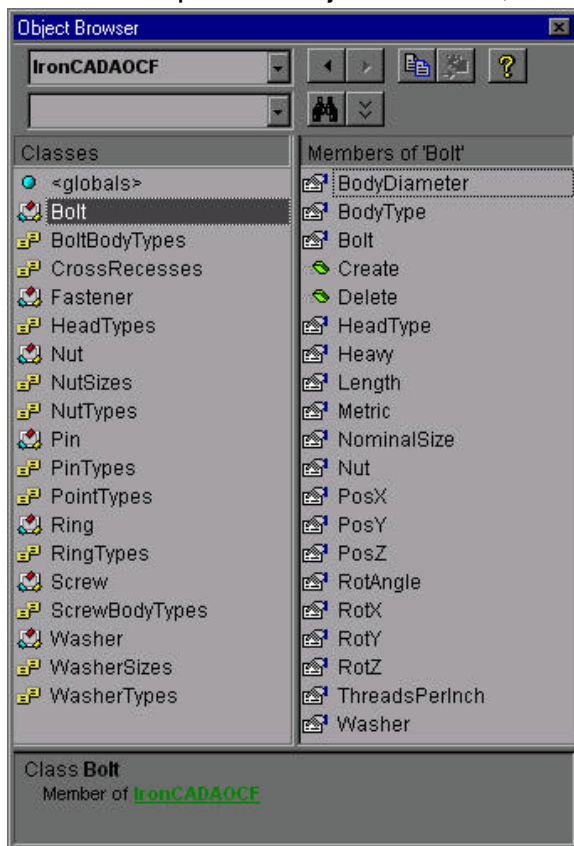
All three parameters must be specified in Drop for the function to be called. In the argument list, 'objFastener' is the shape that is dropped, in our case it's the dummy compound shape. The arguments 'X' and 'Y' are the X and Y position on the screen that the icon was dropped in Pixels. The reason a dummy shape is dropped is because we need to get the position of where the fastener needs to be, otherwise we wouldn't know where to position it. So what happens in the Initialize module, the function OnDrop() is called and what it essentially does is gets the position of the dummy shape, sets the application and deletes the dummy shape, all before the form loads. When using GetObject() that will return the first instance of the application that was opened, but if the user has more than one application open, the probably wouldn't want to drop the fastener in one application and the fastener actually appear in another application so instead of using GetObject() we use the property "Application" on the dummy shape. This returns the instance of the application the shape is in.

This pretty much describes the initialization process of Fastener and how the correct positioning information is collected as far as IronCAD is concerned. OnFormLoad() also reads in a defaults settings file and sets the data on the form.

## 2 : Class Structure

The next important thing to recognize is the class structure of Fasteners. Each fastener is it's own class which enables us to edit each fastener in a separate way and allows Fasteners to be referenced from another Visual Basic project. Lets familiarize ourselves with the class structure first.

Create a new Visual Basic project and reference "IronCAD – Fasteners (Tool)" and then open the object browser, and view the "IRONCADAOCF" type library. It's



easy to see the class structure of Fasteners for the purpose of discussion now. You'll notice that there is a separate class for each fastener that there is, Bolt, Nut, Pin, Ring, Screw and Washer. Lets create a fastener in the middle of IronCAD from a Visual Basic project to get familiar with the process now. Create a new project, reference Fasteners, and the following lines to the Click() action of a command button:

```
Dim objApp As Object
Dim objShape As Object
Dim objBolt As New Bolt

Set objApp = GetObject(, triApp)
Set objShape = objApp.ActivePage.Shape

objBolt.NominalSize = 0.25
objBolt.Length = 3.25
objBolt.HeadType = HeadTypes.Hex
objBolt.Washer = WasherTypes.Plain

objBolt.Create objShape
```

Now, go ahead and run the program and watch as it creates the bolt in the scene and adds a washer. That is all that needs to be done to create a bolt, it's pretty simple. Only a couple of properties were used but it's easy to see that the position, whether it's metric or not, the threads per inch, its body diameter and more can all be specified. In fact, someone referencing Fasteners has exactly the same control of a fastener as they do through the dialog which was our intent. Things to note however is that the nominal size must ALWAYS be specified in inches despite whether the fastener is metric or not. The reason for this will become visible as we take a deeper look into the project. Another thing to notice is that even though the head type HexFlange exists, if you specify that type of head for a nothing will be created, only heads associated with the bolt on the dialog can be specified with the bolt via automation. It is suggested that a little time is spent experimenting with the Fasteners type library before continuing, it is essential to understand.

### 3 : Creation

The way shapes are created differ vastly depending on whether it's an extrude or spin or loft or sweep shape and examples of all are in Fasteners. No examples of shape creation are given in this document since the examples all require at least 100 lines and there are more than 500 examples in the Fasteners project with comments to help explain the creation process.

When the user clicks 'OK' the type of fastener the user wants is dispatched to the correct fastener and then the classes take over the creation process. There are probably less than 100 lines of code between clicking OK and calling the create method of the fastener. The main code exists in the classes. It is necessary to discuss how it's possible to edit a fastener. Once a fastener is created, the fastener calls a function located in the 'Utils.bas' module which assigns Add-On properties to a shape. Near the end of the create method for each fastener (Bolt, Nut, Screw...) there is a line that is similar to the following :

```
Utils.SetBehavior objBoltCompound, ClassID
```

For screws 'objBoltCompound' is 'objWasherCompound', and for screws it is 'objScrewCompound', and so on. 'ClassID' is a property of each class and it goes to the Windows registry and retrieves the Class ID from HKEY\_CLASSES\_ROOT where the Class ID is stored in the form of IronCADAOCF.Bolt or IronCADAOCF.Screw and so on. The subroutine SetBehavior() is commented very well so not much discussion will be given here about the precise things that it does, that's discussed in the code itself.

What this does now is that each shape has an Add-On component whose ClassID is set to the Class ID of the class in the Fasteners project. So when a user double clicks on a fastener, IronCAD looks for the function "DbClick()" in the class. If we take a look at the code for the Bolt class we'll see the following function :

```

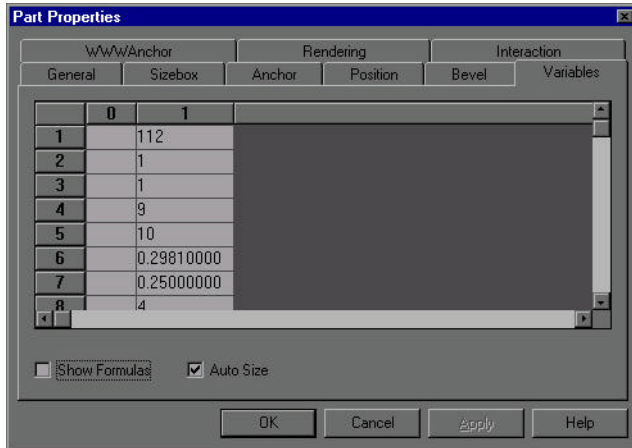
Public Sub DblClick(Cshape As Object , X As Long, Y As Long)

    Edit.DispatchEditProcess FASTENER_BOLT, Cshape

End Sub

```

Each class has a similar function, which calls ‘Edit.DispatchEditProcess().’ It is now visible to see that editing a fastener is not done through the Class, it’s done through a lot of interface code especially if it’s a Bolt. Editing a fastener is generally very simple if it is not a bolt. What happens is when a fastener is created, all the parameters of the fastener are stored in the fastener’s variable component. As an example, create a bolt



and right click on the bolt compound and click “Part Properties.” This will bring up a dialog similar to the one here, and it shows all the variables that the current shape has. This becomes a very powerful feature to save variables to a shape because it allows us to have an initial set of values we can store. When a user double clicks on a shape, we load all the current values of the bolt, and when the user clicks ‘OK’, if the data on the form when the user clicks ‘OK’ is identical to the data that we loaded

initially, we don’t edit the bolt, but if the dimensions of the bolt have changed, we delete the bolt and create a new bolt with the new dimensions.

Editing does bring up some very complicated situations however. Suppose the user creates a bolt and adds a nut and a washer. The user then double clicks the nut and changes it from a hex nut to a slotted castle nut. If the user then double clicks on the bolt assembly, the change needs to be detected. An even more critical situation is if the user deletes the nut and the bolt assembly expects there to be a nut, then the program needs to double-check the existence of the nut and if the nut is not there the option needs to be unchecked.

The code is commented substantially so as far as the involvement of IronCAD goes no more will be discussed here that includes IronCAD’s involvement in the Fastener process. So for the final section the error checking is discussed.

#### 4 : Error Checking

The error checking in Fasteners happens on GotFocus(), not LostFocus() as most programs do it. When a control that needs validation gets focus, two public variables are set in the ‘Interface.bas’ module, one is ‘bItemSelected’ which is true if an item is selected that needs validation, and ‘objItem’ which is set the the item that needs to be validated. These variables are set through the function “OnBoxGotFocus()” which gets called when a drop box gets focus. If the user clicks on an option or a field that

doesn't need validation on LostFocus() then the function "OnOptionClick()" gets called and `'bItemSelected'` is set to false so that the field will not be validated when it loses focus. When a control gets focus, if `bItemSelected` is true, then it calls the function "ValidateInput()" located in the 'Interface.bas' module. If the input is incorrect, the function will set focus on the box that just lost focus. Before focus is set the public variable `'bInValidation'` is set to true so that when the box that was invalid gets focus it doesn't try to validate itself again, otherwise two error messages would pop up which is not very user friendly.

## **Conclusion :**

The examples provided have been commented thoroughly as mentioned before, it's impossible to explain everything in full detail in a paper. The comments should cover and explain what has not been explained here. If there are any questions please contact one of our technical marketing people who will be able to provide more support about problems that may arise. Good Luck!