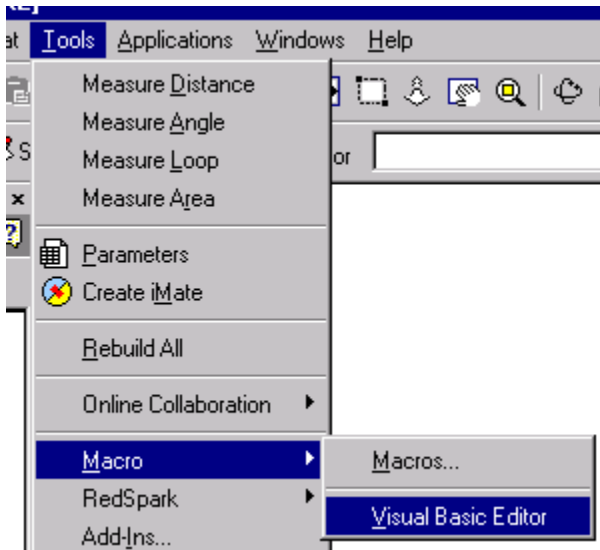


## Lesson 15 Visual Basic

Inventor Release 5 includes a Visual Basic Editor to allow users to customize Inventor. With VBA, you can create your own dialog boxes and interface tools. VBA does not create standalone applications, but always runs from inside Inventor.

You create IVB files by selecting Files->New Project. The advantage of an IVB file is that it's independent of the document.



The Visual Basic Editor is accessed from  
Tools->Macro->  
Visual Basic Editor



**TIP:** If you plan to do a lot of customization, it is worthwhile to invest in a full copy of Visual Basic from Microsoft. That will give you access to a suite of tools. The Visual Basic Editor inside of Inventor is a scaled-down version of the full application.

The Internet is a great source for Visual Basic tutorials, ActiveX controls, and tools you can use for creating super applications.



**TIP:** It is a good idea to create a sketch of how you want your dialog box to appear before you start to help you with laying out your control tools.

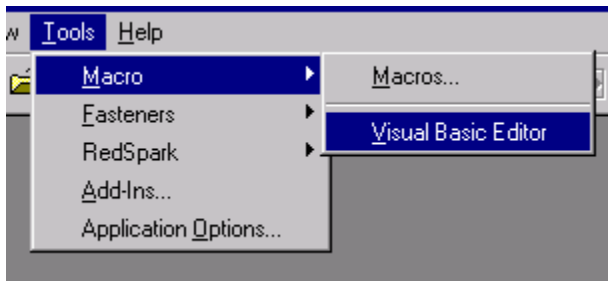
## Pipe Maker

Let's assume that you need to create pipes on a regular basis. You need a wide variety of pipe sizes. The interior diameter, wall thickness, and length can vary. You could handle this using Parameters or iParts, but the purpose of this exercise is to get familiar with the VBA tools.

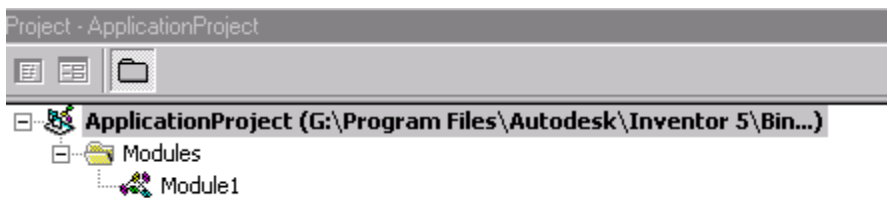
### Exercise 15-1 Dialog Box Layout

You do not need to have a file open to access the Visual Basic Editor. This project automatically will open a new part file and create the necessary geometry.

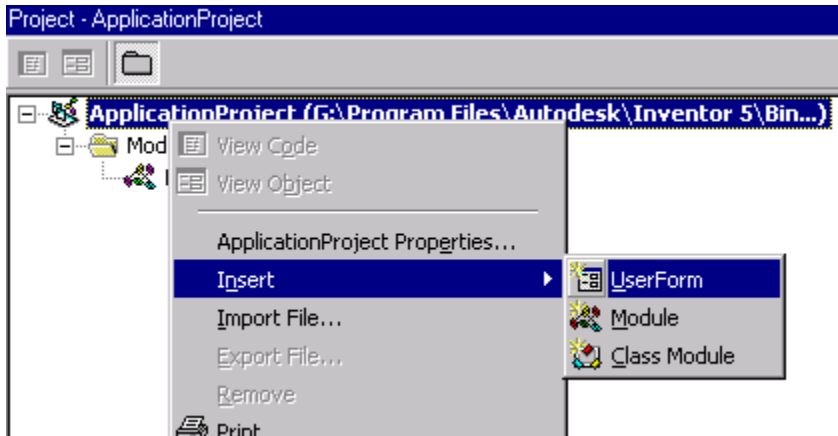
Estimated Time: 60 minutes



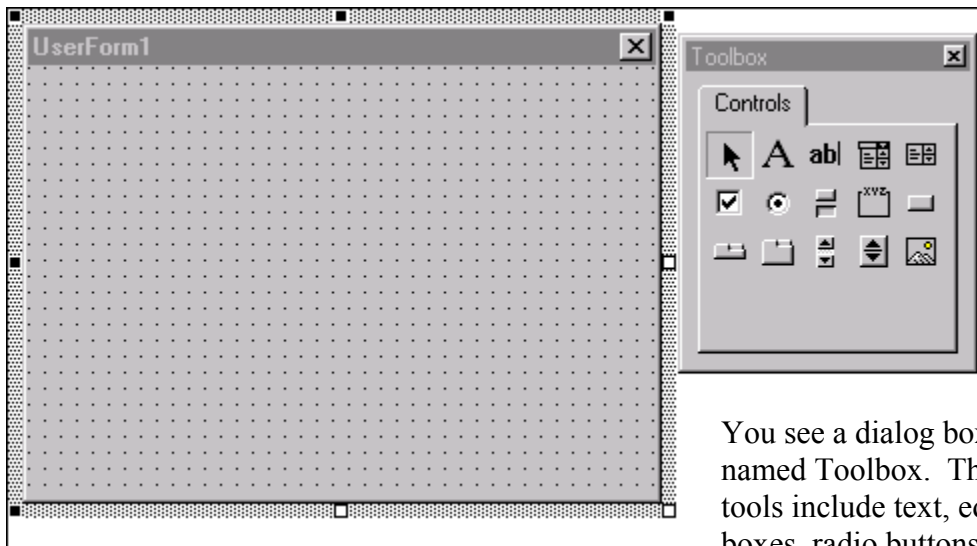
Start the Visual Basic Editor under the Tools Menu.



We see our project listed in the Browser.


















Highlight the project in the browser.  
Right click->Insert  
->UserForm.



You see a dialog box named Toolbox. The tools include text, edit boxes, radio buttons, and lists

Your screen now has several items in it. The UserForm is a canvas for the dialog box you will create.

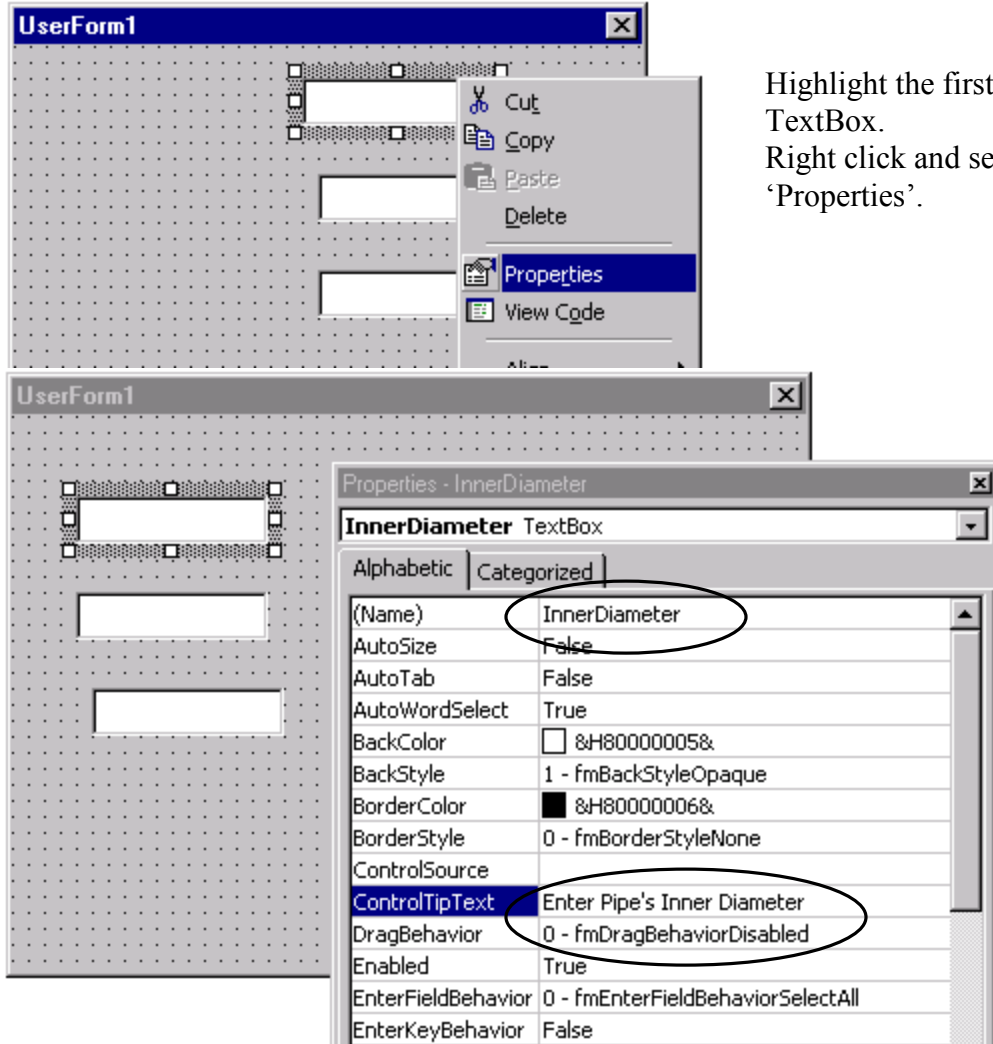
## Toolbox

Icon	Name	Function
	Select	Select Objects is the only item in the <b>Toolbox</b> that doesn't draw a control. When you select it, you can only resize or move a control that has already been drawn on a form.
	Text	Allows you to have text that you do not want the user to change, such as a caption under a graphic.
	Text box	This is a box where the user can enter alphanumeric characters; also known as a data entry box.
	ComboBox	Allows you to draw a combination list box and text box. The user can either choose an item from the list or enter a value in the text box.
	ListBox	Use to display a list of items from which the user can choose. The list can be scrolled if it has more items than can be displayed at one time.
	CheckBox	Creates a box that the user can easily choose to indicate if something is true or false, or to display multiple choices when the user can choose more than one.
	RadioButton	Allows you to display multiple choices from which the user can choose only one.
	ToggleButton	Creates a button that toggles on and off.
	Frame	Allows you to create a graphical or functional grouping for controls. To group controls, draw the frame first, and then draw controls inside the frame.
	CommandButton	Creates a button the user can choose to carry out a command.
	TabStrip	Allows you to define multiple pages for the same area of a window or dialog box in your application.
	MultiPage	Presents multiple screens of information as a single set.
	ScrollBar	Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale, or as an input device or indicator of speed or quantity
	SpinButton	A spinner control you can use with another control to increment and decrement numbers. You can also use it to scroll back and forth through a range of values or a list of items.
	Image	Displays a graphical image from a bitmap, icon, or metafile on your form. Images displayed in an <b>Image</b> control can only be decorative and use fewer resources than a <b>PictureBox</b> .

We need to place three data entry variables: Pipe Inner Diameter, Pipe Outer Diameter, and Pipe Length.

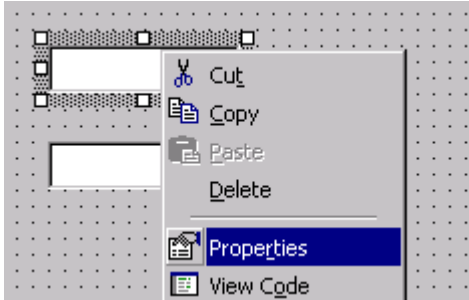
ab|

To place them, we use the Textbox tool. To insert the edit box onto your dialog box, just drag and drop the Textbox tool three times into your dialog box.

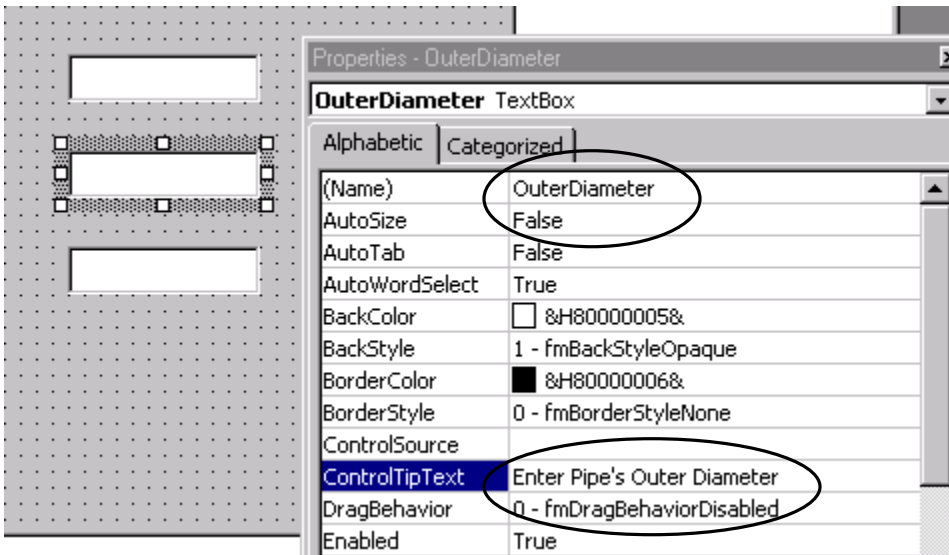


In the Properties dialog, you can set how the textbox is defined.  
Change the Name to InnerDiameter.  
Add a ControlTipText to help users know what you want to be entered in the text box.  
Close the Properties Dialog box.

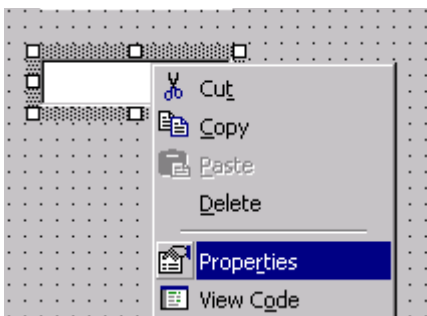
Lesson 15  
Visual Basic



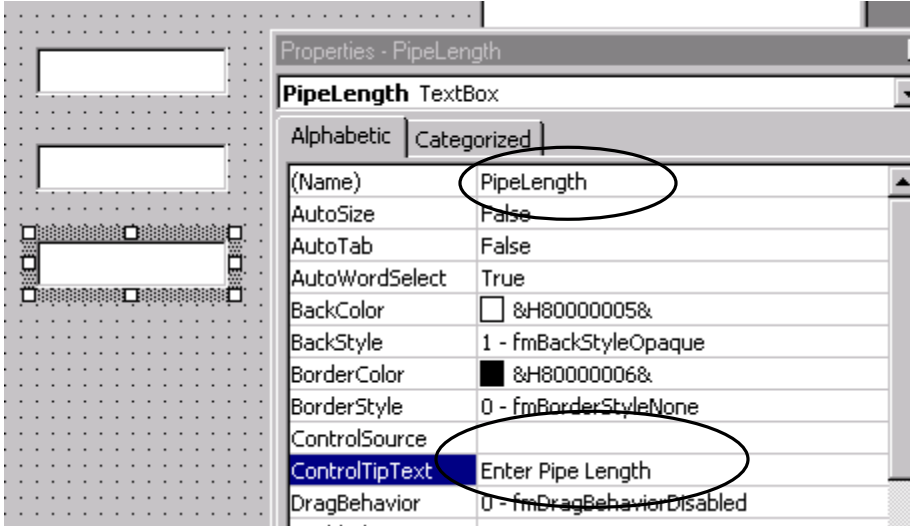
Select the second TextBox.  
Right click and select 'Properties'.



Set the Name of the TextBox as OuterDiameter.  
Set the ControlTipText to 'Enter Pipe's Outer Diameter'.



Select the third TextBox.  
Right click and select 'Properties'.



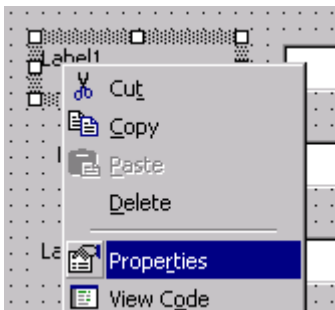
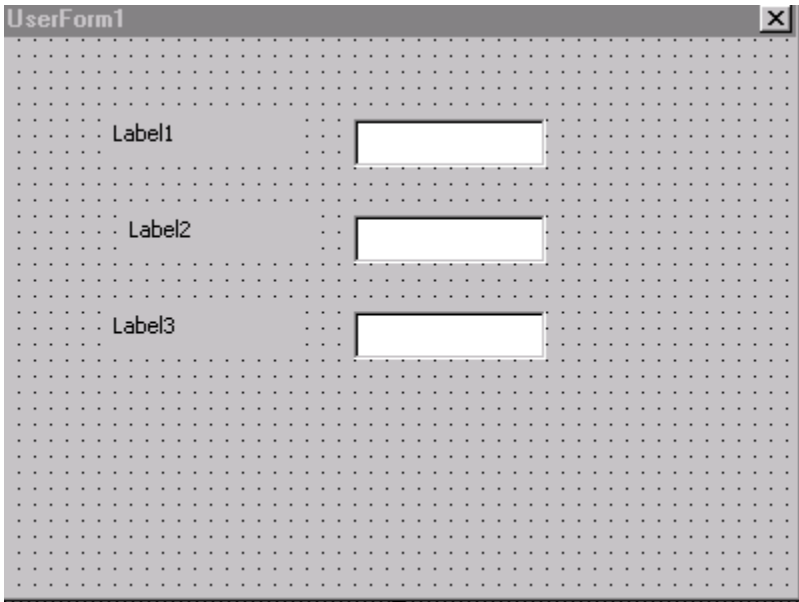
Change the Name to PipeLength.

Change the ControlTipText To 'Enter Pipe Length'.

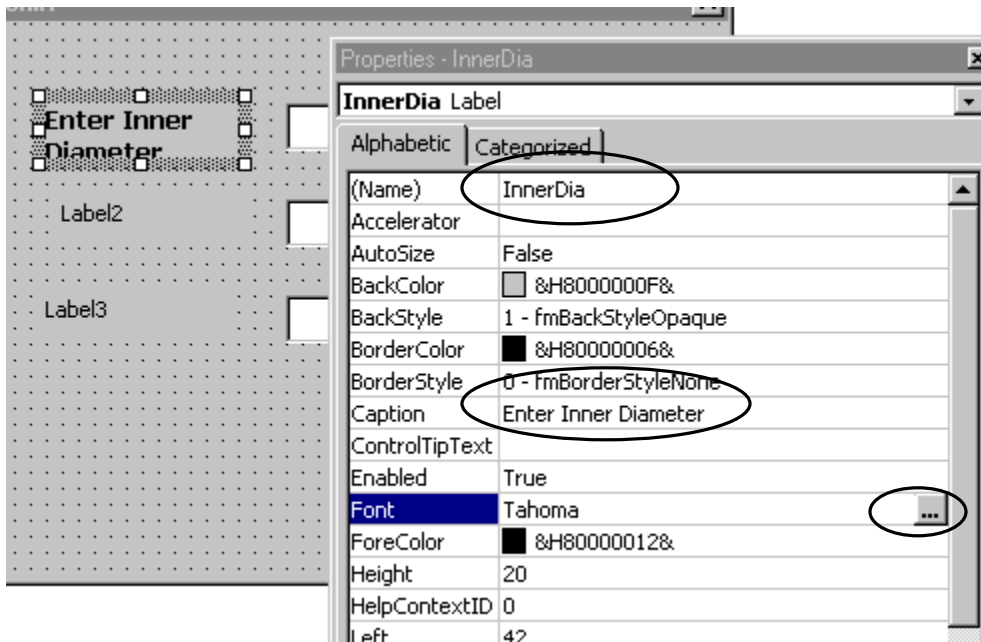
Next we add three text labels for the TextBoxes.

A

Select the Text tool and drag it into the UserForm three times.



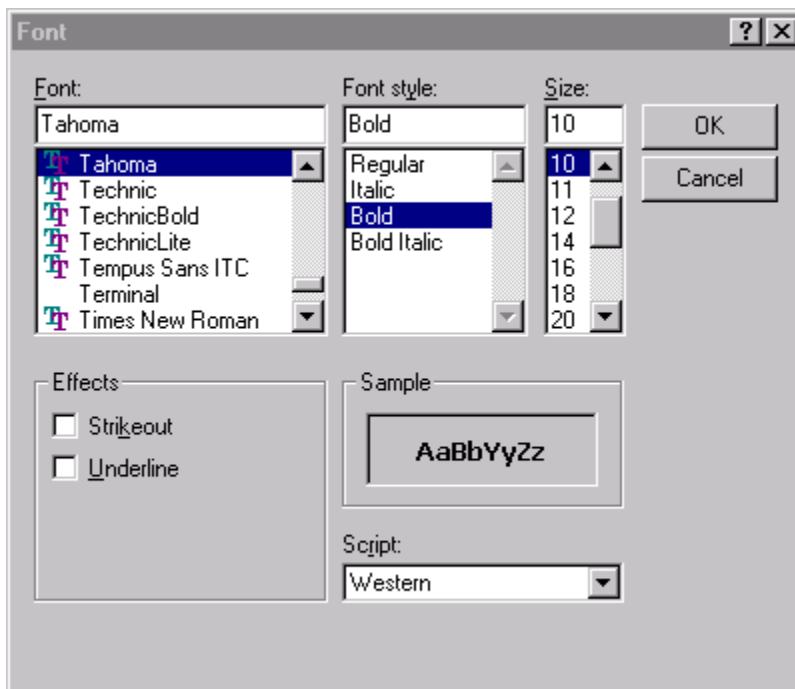
Select the first label.  
Right click and select 'Properties'.



Set the Name to InnerDia.

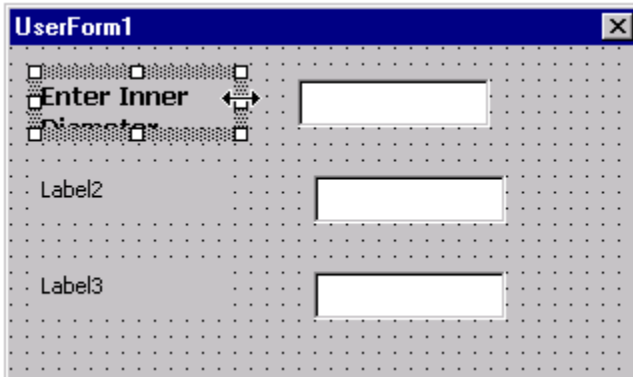
The Caption is what the user will see when the dialog box is activated.

To change the appearance of the Font, select the ... button in the Font row.

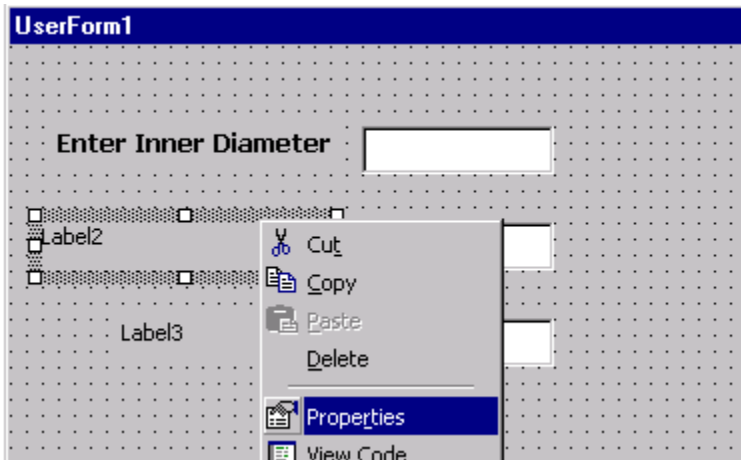


Set the Font to  
Tahoma, Bold, with a  
Size of 10.  
Set OK.

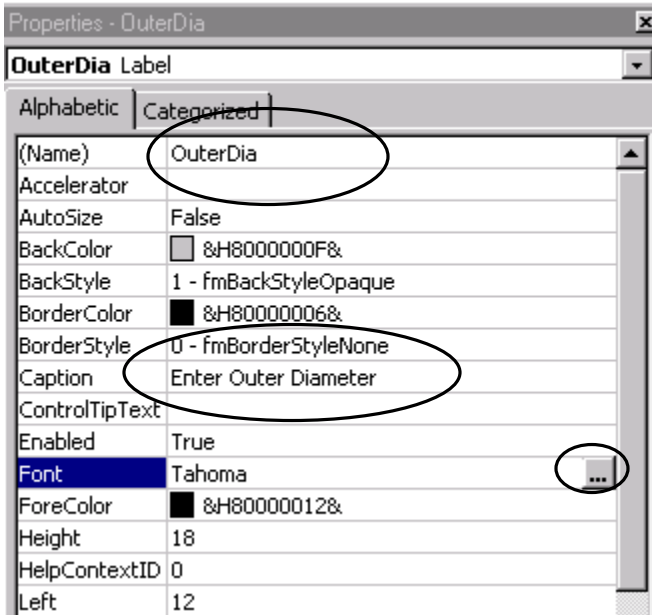




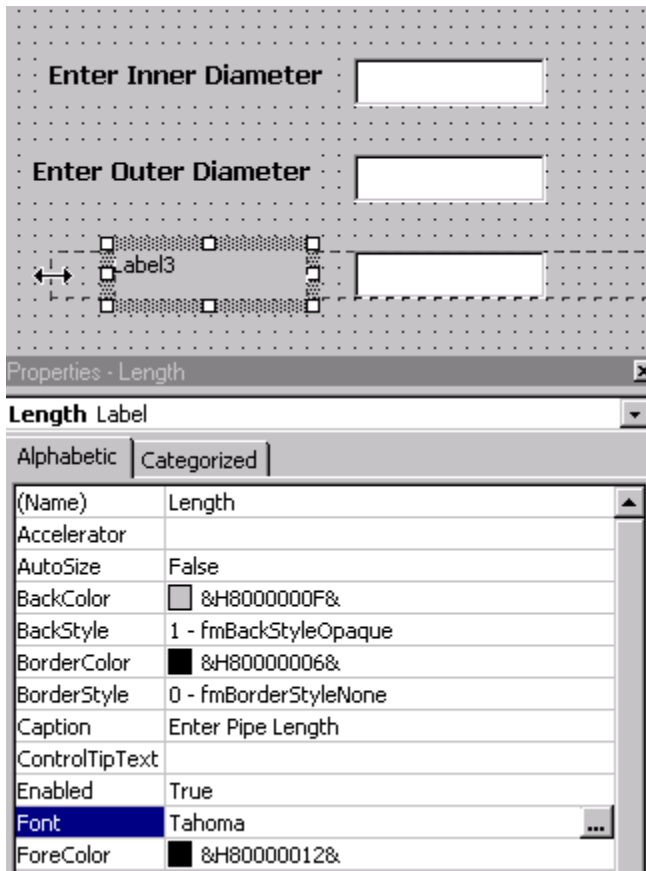
You can expand the label by using the grips when it is selected.



Select Label2.  
Right click and select 'Properties'.



Set Name to OuterDia.  
Set Caption to  
Enter Outer Diameter.  
Set the Font style to  
Tahoma, Bold, Size 10.



You can adjust the width, height, scale and location of the labels by selecting on the label and using your mouse to stretch and move it. Pay attention to the cursor cues – they tell you whether you are in MOVE mode or RESIZE mode.

Change the Properties of the Label3.

Set Name to Length.

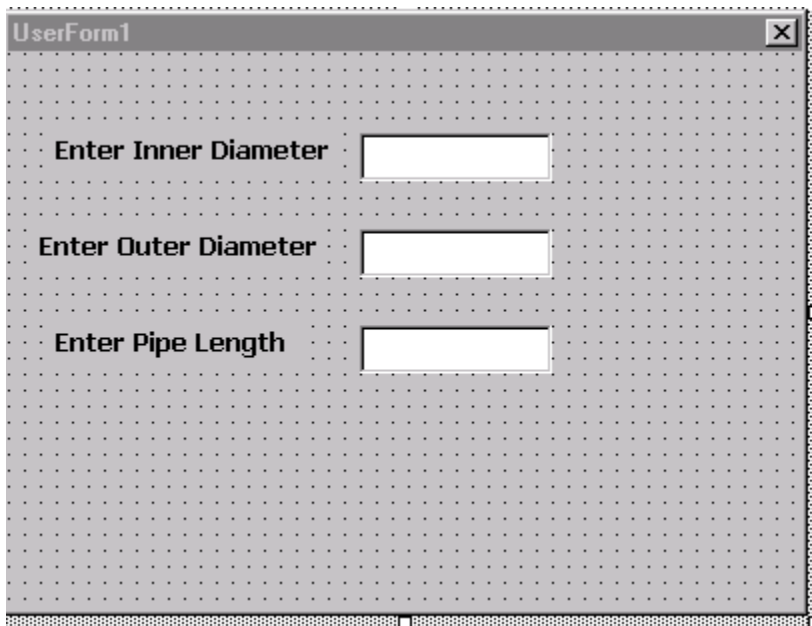
Set Caption to

Enter Pipe Length.

Set the Font to

Tahoma, Bold, 10.

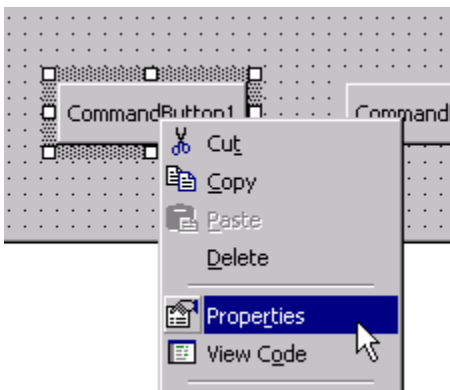
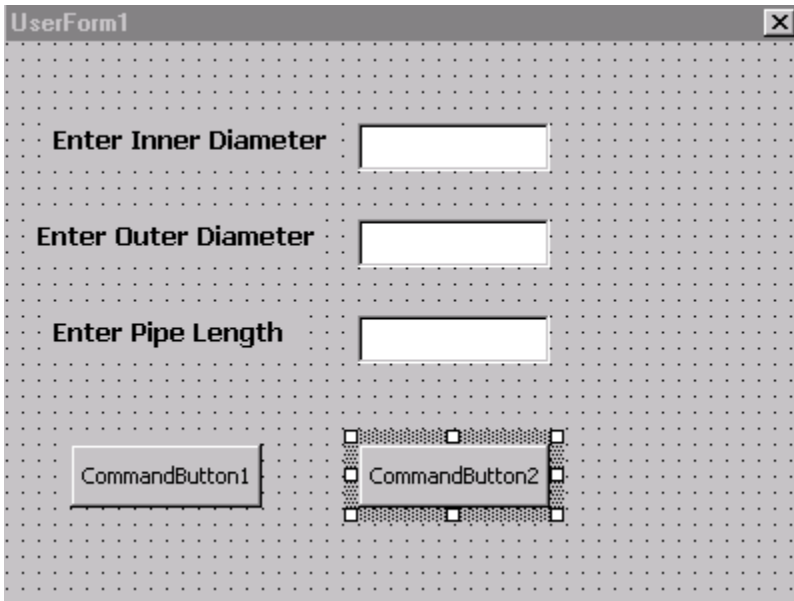
Our dialog box, so far.



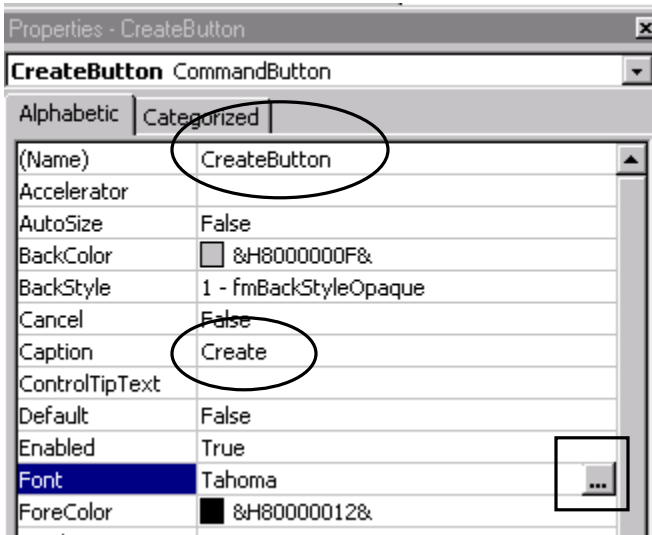
We need two command buttons. One will create the pipe and one will exit the dialog box.



Drag and drop two command buttons onto your dialog box.

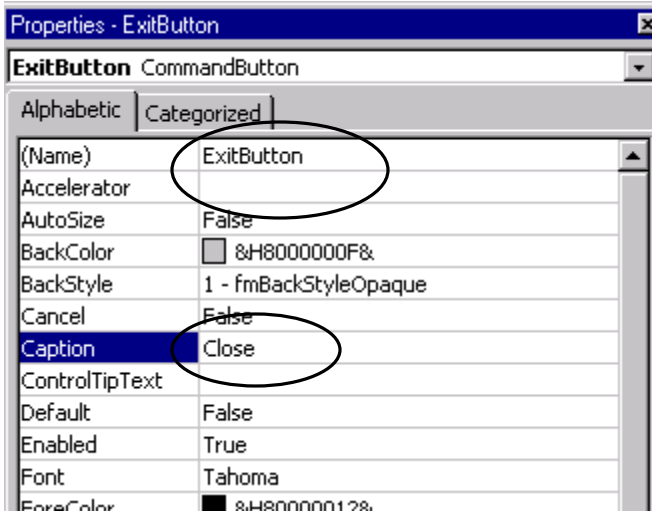


Select CommandButton1.  
Right click and select Properties.



Change the Name to  
CreateButton.  
Change the Caption to  
Create  
Change the Font to  
Tahoma, Bold, 10.

Lesson 15  
Visual Basic

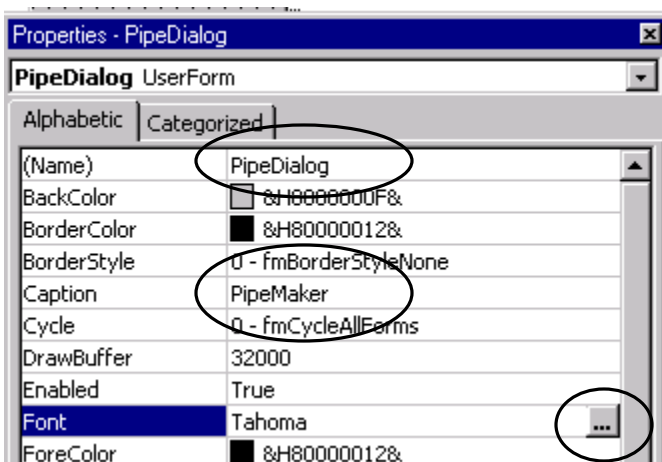


Change the Properties of the CommandButton2.  
Set the Name to ExitButton.  
Set the Caption to Close.  
Set the Font to Tahoma, Bold, 10.

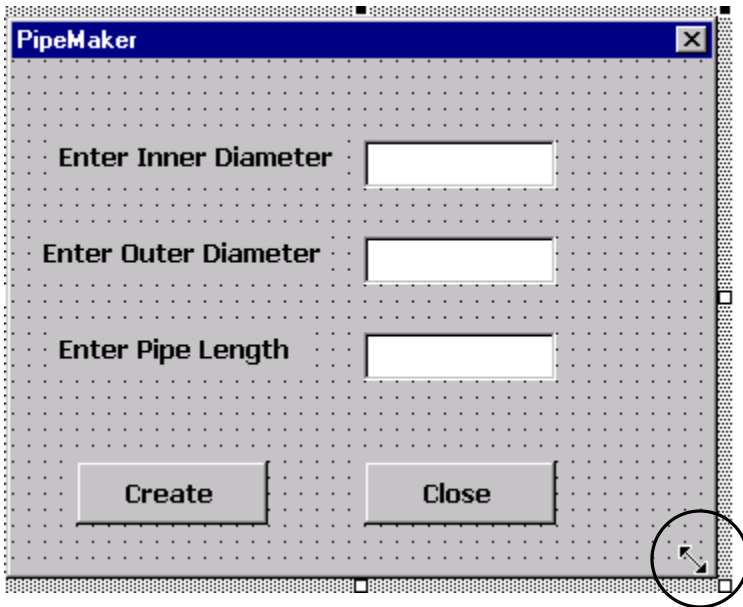


**TIP:** When defining the labels for your command buttons, use words and syntax similar to Inventor standard dialogs. That way your custom dialog box will work and look like an Inventor dialog box.

Select the entire dialog box by picking on one of its edges.  
Press F4.  
This brings up the Properties dialog box.

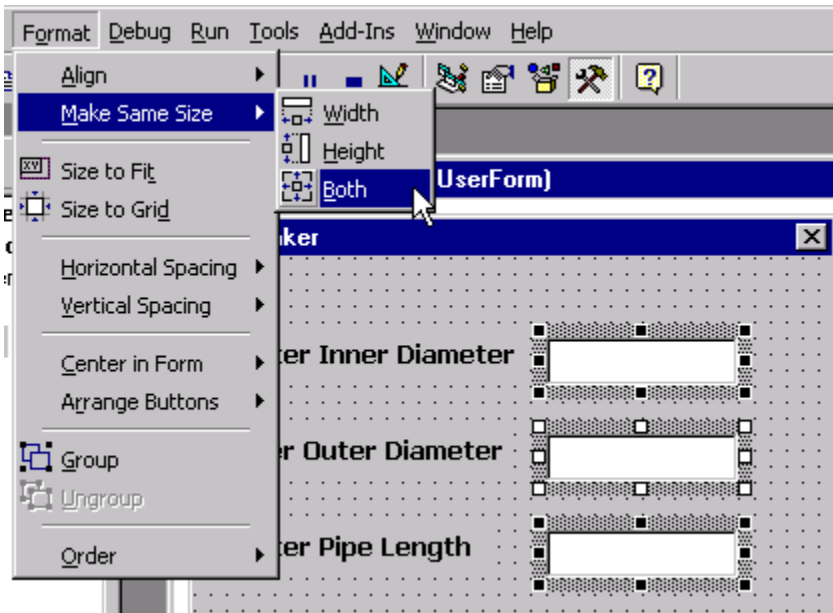


Change the Name to PipeDialog.  
Change the Caption to PipeMaker.  
Set the Font to Tahoma, Bold, 10.



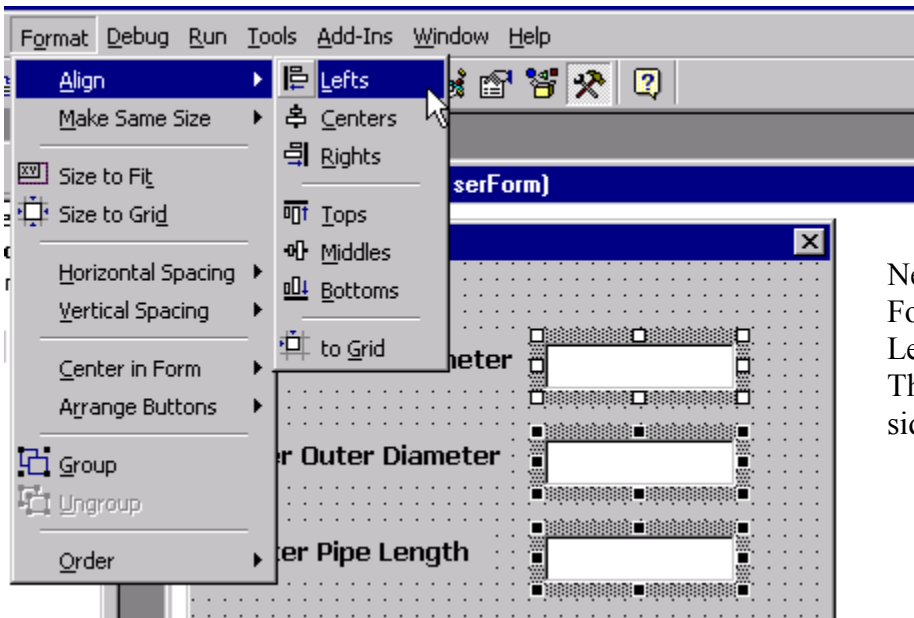
Our dialog box, so far.

We can resize the dialog so it doesn't have so much blank space by grabbing one of the corners and scaling.

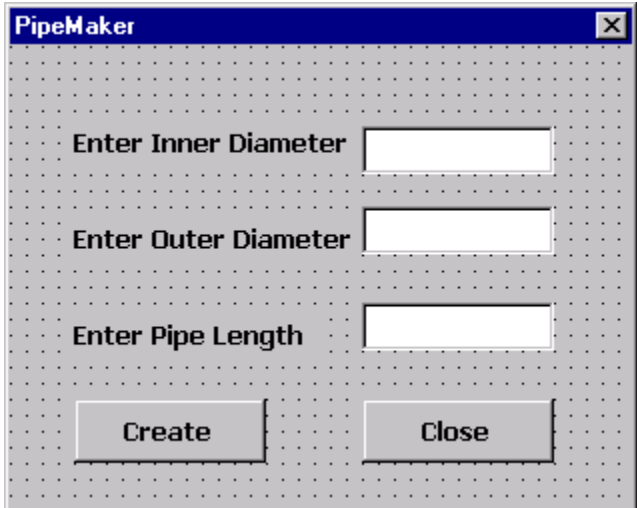


We can clean up the appearance of the dialog by going to the Format menu. Select all three Edit boxes. You can select more than one edit box by holding down the Shift key. Select Format-> Make Same Size-> Both.

Lesson 15  
Visual Basic

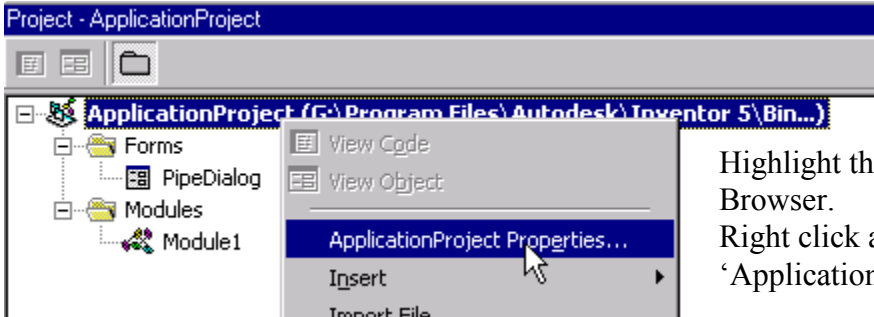


Next go to  
Format->Align->  
Left.  
This will align the left  
side of the edit boxes.

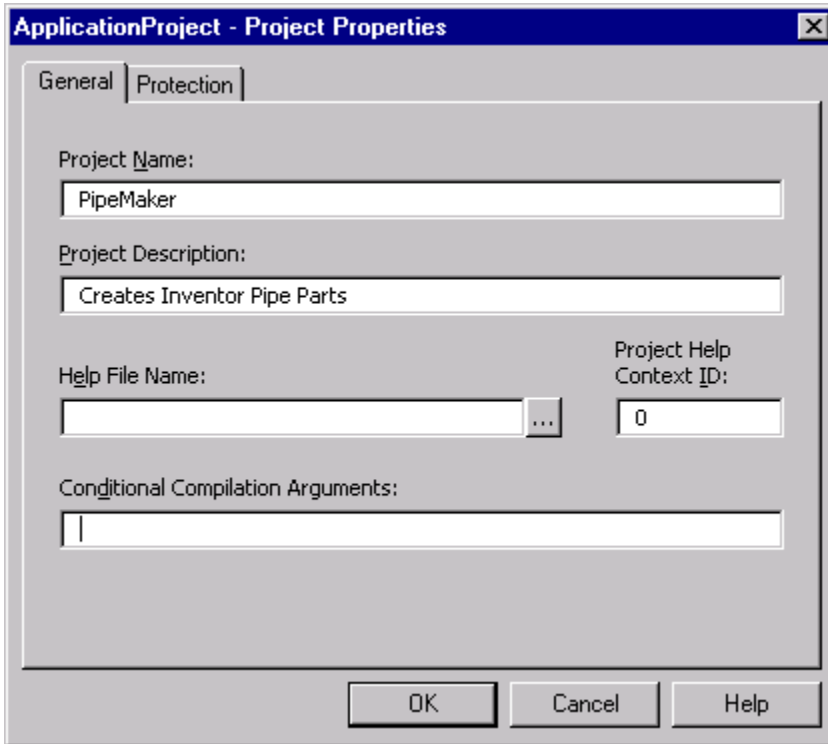


Our dialog box.

In the browser, we see the  
dialog box we are building  
under 'Forms'.

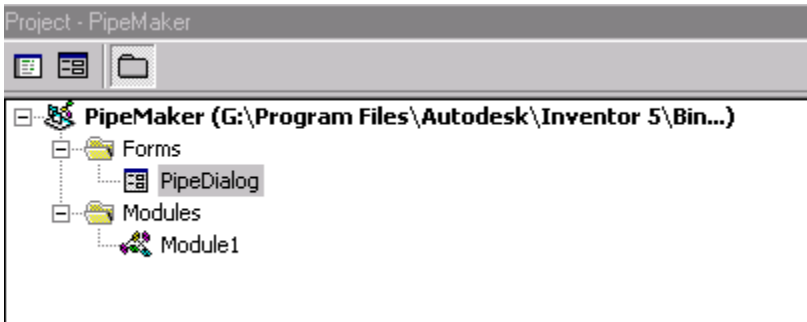


Highlight the project in the  
Browser.  
Right click and select  
'ApplicationProject Properties'.

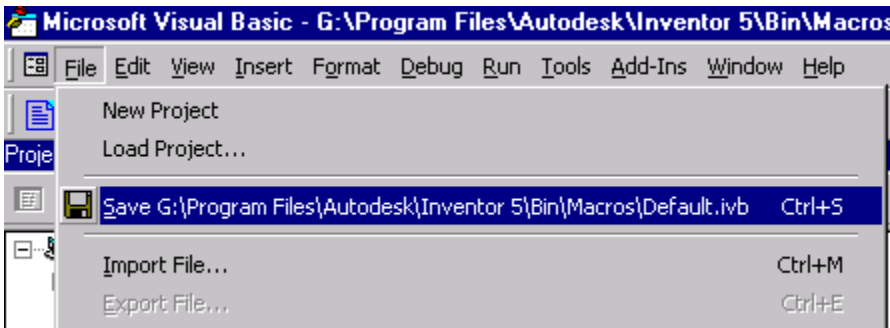


Fill in the Project Name As PipeMaker. Enter a Project Description. For advanced applications, you can create a \*.hlp file and link it to the project. Creating help files is fairly easily done. Microsoft has a free download in their developer's area to allow you to do this.

Press 'OK'.



Our Browser updates with the new project name.

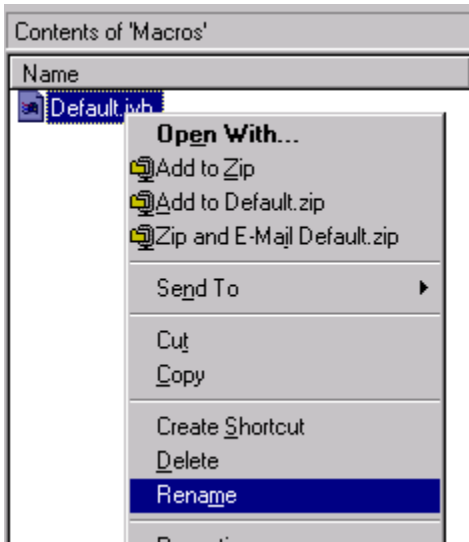


To save, go to File->Save or Ctrl-S.

Notice that the macro we are writing is being stored in the Macros subdirectory as 'default.ivb'.

Lesson 15  
Visual Basic

---



If you close Inventor, you can rename the default.ivb file to a name that is easier to remember.

Locate the default.ivb file in the Macros subdirectory using Windows Explorer.

Right click and select 'Rename'.

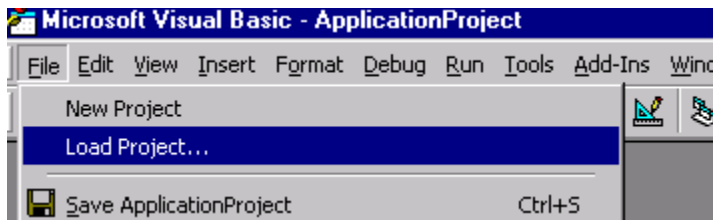
Rename the file to 'PipeMaker.ivb'.



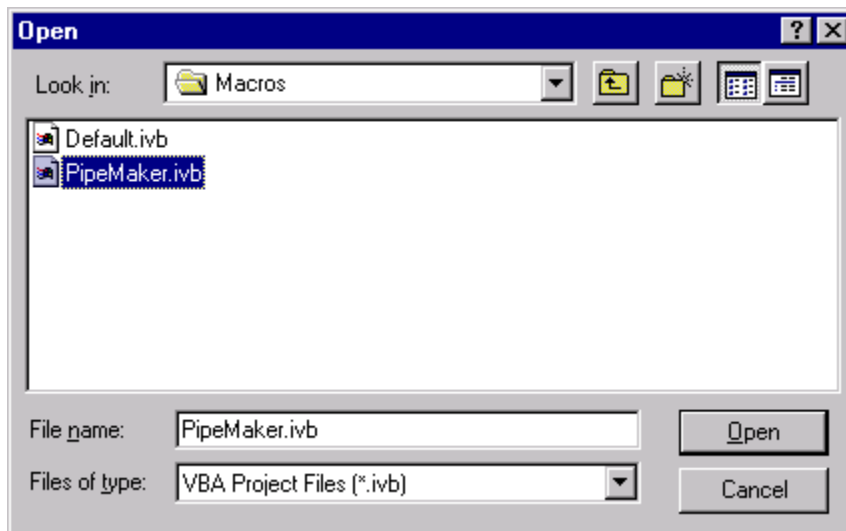
## Exercise 15-2 Assigning Subroutines

File Name: PipeMaker.ivb  
Estimated Time: 60 minutes

Creating the dialog box was the easy part, we still need to make it work. For our next exercise, we write the code.

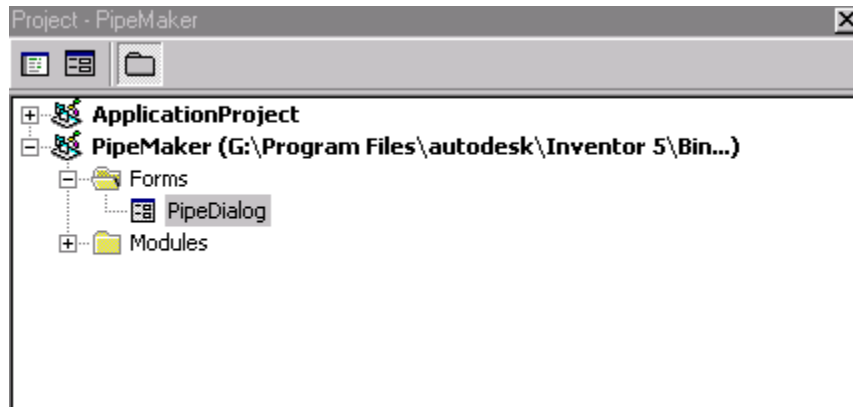


Start Inventor.  
Go to Tools->Macros->Visual Basic Editor.  
Under File->Load Project.

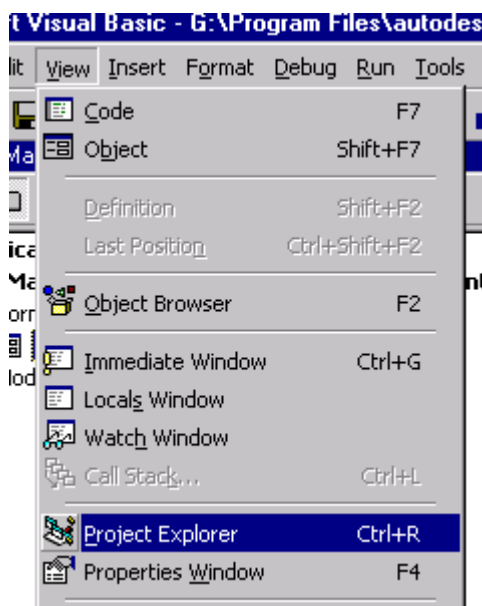


Locate the PipeMaker.ivb we created in the previous lesson.

## Lesson 15 Visual Basic

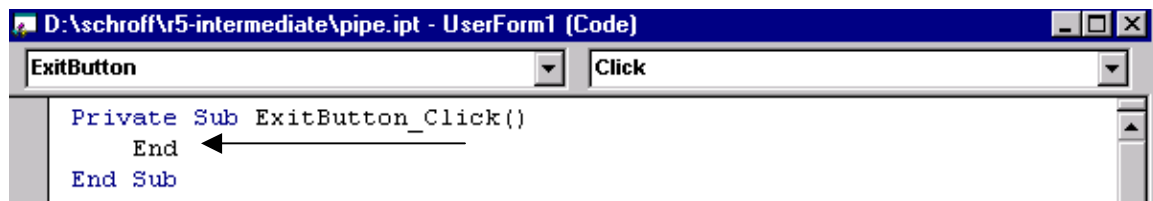


The project loads.



If you don't see the browser in your window, go to View->Project Explorer  
Or  
Control-R.

Double-click on the Close command button in your dialog box.



A new window pops up. This window controls the software code that runs the button. Type the word End between the Private Sub and End Sub lines. Close the window.

This means that when the user presses the Exit button, the program will automatically end.

The Create Command button will initiate the key actions in this program; it will create the pipe.

Double-click on the Create button.

```
Private Sub CreateButton_Click()
```

```
End Sub
```

---

```
Private Sub ExitButton_Click()
```

```
End
```

```
End Sub
```

We see the subroutine we already defined for the ExitButton. Above that is the subroutine for the CreateButton.

```
Private Sub CreateButton_Click()
```

```
    'Define Variables
```

```
    Dim CenterRadiusO As Double
```

```
    Dim CenterRadiusI As Double
```

```
    Dim DistanceExtent As Double
```

```
End Sub
```

---

```
Private Sub ExitButton_Click()
```

```
End
```

```
End Sub
```

To create a comment, place a ' before the line.

We start by defining our Variables.

We have three variables:

outer radius, called CenterRadiusO

inner radius, called CenterRadiusI

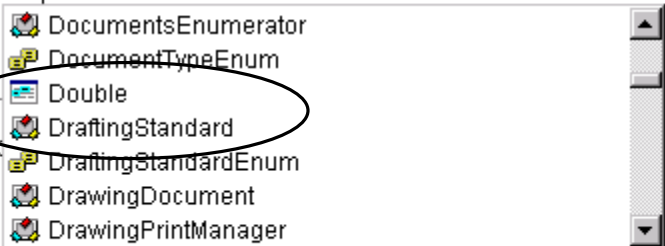
pipe length, called DistanceExtent

## Lesson 15

### Visual Basic

---

```
Private Sub CreateButton_Click()  
  
    'Define Variables  
    Dim CenterRadiusO As Double  
    Dim CenterRadiusI As Double  
    Dim DistanceExtent As  
  
End Sub  
  
Private Sub ExitButton_  
    End  
End Sub
```

A screenshot of the Visual Basic IDE's type dropdown menu. The menu is open, showing a list of types: DocumentsEnumerator, DocumentTypeEnum, Double, DraftingStandard, DraftingStandardEnum, DrawingDocument, and DrawingPrintManager. The 'Double' type is circled in red. A horizontal line points from the 'As' keyword in the code above to the dropdown menu.

As you type, Visual Lisp will bring up a help box to provide you with the possible definitions available to you. Simply scroll down until you locate the desired definition. Double Left Pick to select.

Each Dim statement created a variable where we store information.

```
'Define variables  
Dim CenterRadiusO As Double  
Dim CenterRadiusI As Double  
Dim DistanceExtent As Double  
  
'Set Variables from values in edit boxes  
CenterRadiusO = OuterDiameter / 2  
CenterRadiusI = InnerDiameter / 2  
DistanceExtent = PipeLength
```

Next, we define how we set the variables using the values from the edit boxes.

Be sure to set the variables so that you use the same variable names that you defined under the edit box property. If you use a different name or misspell a name, then the program will not work properly. This is one of the most common errors when programming.

```
'Set Variables from values in edit boxes
CenterRadiusO = OuterDiameter / 2
CenterRadiusI = InnerDiameter / 2
DistanceExtent = PipeLength

'error message if InnerDiameter is greater than OuterDiameter
If InnerDiameter > OuterDiameter Then
    CreateButton.Enabled = False
    MsgBox ("Inner Diameter Must be a Smaller Value than Outer Diameter.")
    Exit Sub
End If

End Sub
```

---

We do some error checking. We need to make sure the user enters a larger value for the outer diameter than the inner diameter.

CreateButton.Enabled = False means that the program will not create the part. Instead, we will see a message box with the message we see inside the parentheses.

Any If statement defined must be closed with an 'End If'.

```
'error message if InnerDiameter is greater than OuterDiameter
If InnerDiameter > OuterDiameter Then
    CreateButton.Enabled = False
    MsgBox ("Inner Diameter Must be a Smaller Value than Outer Diameter.")
    Exit Sub
End If

' Create new part document
Dim oDoc As PartDocument
Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

End Sub
```

---

If the inner diameter is less than the outer diameter, we proceed with the program.

Our first step is to open a new part file.

```
' Create new part document
Dim oDoc As PartDocument
Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)

' Get component definition from part document
Dim oCompDef As ComponentDefinition
Set oCompDef = oDoc.ComponentDefinition
```

Inventor internally handles the units in centimeters.

## Lesson 15

### Visual Basic

---

When executing the following line,

```
Dim oDoc As PartDocument
```

```
Set oDoc = ThisApplication.Documents.Add(kPartDocumentObject)
```

it automatically defaults to centimeters.

If you intend to have the units in either inches or mm, then you need to give it its full path. The full path can also be accessed from 'TemplateDir' property of the 'Preferences' object.

Next, we state our intention to create a component definition.

```
' Get component definition from part document
Dim oCompDef As ComponentDefinition
Set oCompDef = oDoc.ComponentDefinition

' Create a new sketch on the X-Y work plane.
Dim Sketch1 As PlanarSketch
Set Sketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))
```

End Sub

---

Before we can place our geometry, we need to define which work plane to use.

```
' Create a new sketch on the X-Y work plane.
Dim Sketch1 As PlanarSketch
Set Sketch1 = oCompDef.Sketches.Add(oCompDef.WorkPlanes.Item(3))

' Set a reference to the transient geometry object.
Dim oTransGeom As TransientGeometry
Set oTransGeom = ThisApplication.TransientGeometry
```

End Sub

---

The number (3) in the WorkPlanes.Item indicates the XY plane.

The oTransGeom variable will be used to locate the center point of the two circles.

```
' Set a reference to the transient geometry object.
Dim oTransGeom As TransientGeometry
Set oTransGeom = ThisApplication.TransientGeometry

' Draw a circle on the sketch.
Dim OuterCircle As SketchCircle
Set OuterCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadiusO)
Dim InnerCircle As SketchCircle
Set InnerCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadiusI)
```

End Sub

---

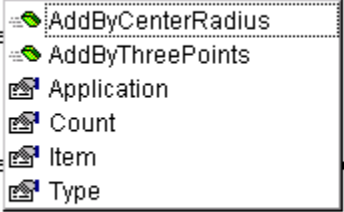
Finally, we create the inner and outer circles.

You can use the Visual Lisp help to assist you in completing the lines.

We locate both circle center points at 0,0.

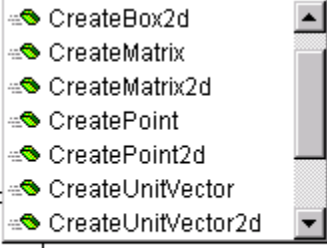
```
' Set a reference to the transient geometry object.
Dim oTransGeom As TransientGeometry
Set oTransGeom = ThisApplication.TransientGeometry

' Draw a circle on the sketch.
Dim OuterCircle As SketchCircle
Set OuterCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadius)
Dim InnerCircle As SketchCircle
Set InnerCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadiusI)
```



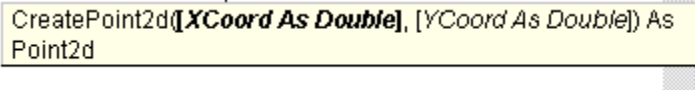
The Help shows us that if we elect to create a sketch circle using the Add By Center Radius method, we need to define the center point's location and the radius value. OTransGeom is the variable we defined to be used to help locate the center point.

```
try object.
ntGeometry
AddByCenterRadius (oTransGeom.CreatePoint2d(0, 0), CenterRadius)
AddByCenterRadius (oTransGeom.CreatePoint2d(0, 0), CenterRadiusI)
```



Locate the CreatePoint2d in the help pop-up.

```
Radius (oTransGeom.CreatePoint2d(0, 0), CenterRadius)
Radius (oTransGeom.CreatePoint2d(0, 0), CenterRadiusI)
```



We are then prompted for what point to use. We enter 0,0 to use the origin.

We complete the line of code by defining the Center Radius as the variable CenterRadiusI.

```
'Draw a circle on the sketch
Dim OuterCircle As SketchCircle
Set OuterCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadius)
Dim InnerCircle As SketchCircle
Set InnerCircle = Sketch1.SketchCircles.AddByCenterRadius(oTransGeom.CreatePoint2d(0, 0), CenterRadiusI)
```

## Lesson 15

### Visual Basic

---

```
' Draw a circle on the sketch.
Dim OuterCircle As SketchCircle
Set OuterCircle = Sketch1.SketchCircles.AddByCenterRad
Dim InnerCircle As SketchCircle
Set InnerCircle = Sketch1.SketchCircles.AddByCenterRad

' Create a profile.
Dim Profile As Profile
Set Profile = Sketch1.Profiles.AddForSolid
```

`End Sub`

---

Next we define the profile.

```
' Create a solid extrusion.
Dim Extrusion1 As ExtrudeFeature
Set Extrusion1 = oCompDef.Features.ExtrudeFeatures.AddByDistanceExtent(Profile,
DistanceExtent, kSymmetricExtentDirection, kJoinOperation)
```

```
' Create a solid extrusion.
Dim Extrusion1 As ExtrudeFeature
Set Extrusion1 = oCompDef.Features.ExtrudeFeatures.AddByDistanceExtent(Profile, 3, kSymmetricExtentDirection,
```

We extrude the profile.  
(Profile,DistanceExtent, kSymmetricExtentDirection,kJoinOperation)  
define all the variables we usually set in the Extrude dialog.

Profile refers to the Profile we defined earlier  
DistanceExtent is the PipeLength Variable  
kSymmetricExtentDirection indicates a mid-plane extrusion.  
KJoinOperation indicates a Join.

```
Set oExtrude = oCompDef.Features.ExtrudeFeatures.AddByDistanceExtent( _
oProfile, 1, kNegativeExtentDirection,
kJoinOperation)
```

In the above example, we extrude a distance of 1 unit in the negative direction.

The syntax for extrusions is outlined as follows:



AddByDistanceExtent  
 (Profile As Profile, Distance As Variant,  
 ExtentDirection As PartFeatureExtentDirectionEnum,  
 Operation As PartFeatureOperationEnum,  
 TaperAngle As Variant = 0) As ExtrudeFeature

Profile	Input Profile object used to define the shape of the extrusion. If the Operation argument is anything except kSurfaceOperation, then the input profile must have closed paths. Open paths are valid when creating surfaces.
Distance	Input Variant that defines the length of the extrusion. This can be either a numeric value or a string. A parameter for this value will be created and the supplied string or value is assigned to the parameter. If a value is input, the units are centimeters. If a string is input, the units can be specified as part of the string or it will default to the current length units of the document.
ExtentDirection	Input constant that indicates which side of the sketch plane to extrude toward. Valid input is kPositive, kNegative, or kSymmetric. kPositive defines the offset direction to be in the same direction as the normal of the sketch plane.
Operation	Input constant that indicates the type of operation to perform. Valid input is kJoinOperation, kCutOperation, kIntersectOperation, kSurfaceOperation.
TaperAngle	Optional Input Variant that defines the angle of the taper. If not supplied, the feature will be created with a taper angle of zero.  This can be either a numeric value or a string. A parameter for this value will be created and the supplied string or value is assigned to the parameter. If a value is input, the units are radians. If a string is input, the units can be specified as part of the string or it will default to the current angle units of the document.

```
' Create a solid extrusion.
Dim Extrusion1 As ExtrudeFeature
Set Extrusion1 = oCompDef.Features

' fit the view
ThisApplication.ActiveView.Fit
```

End Sub

Next, we zoom the view to fit the part.

Lesson 15  
Visual Basic

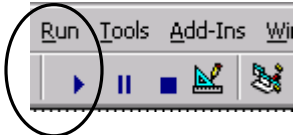
---

```
'close dialog  
End
```

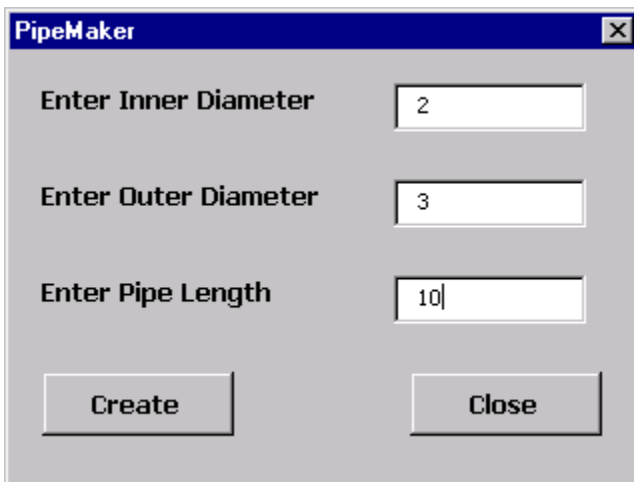
To close the dialog, simply type  
'End'.



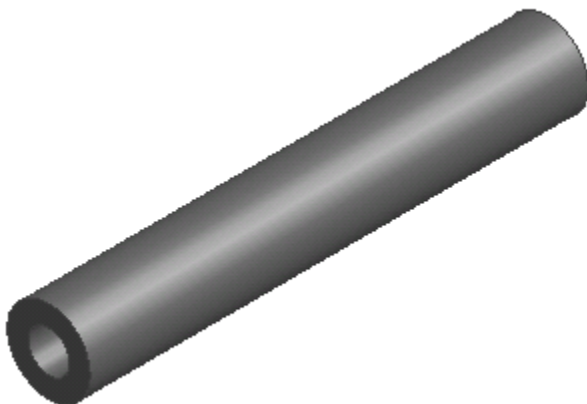
Save the project.



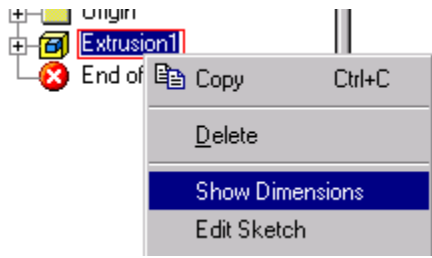
To test your program, press Run in the Menu or the Run arrow.



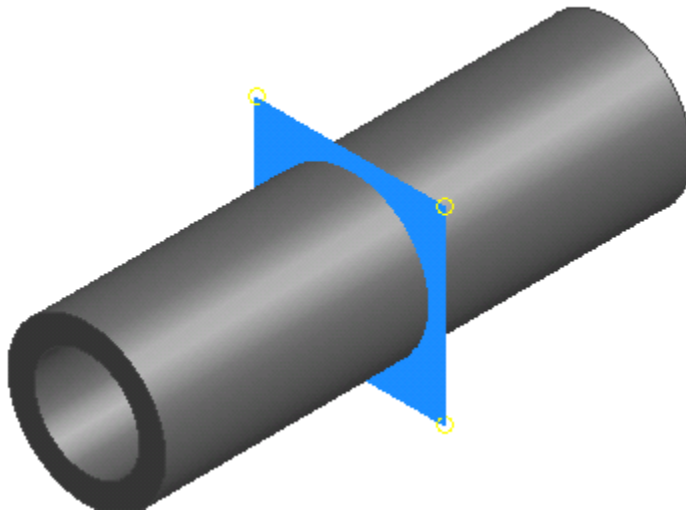
Fill in some values and press 'Create'.  
Remember that our values are in centimeters by default.



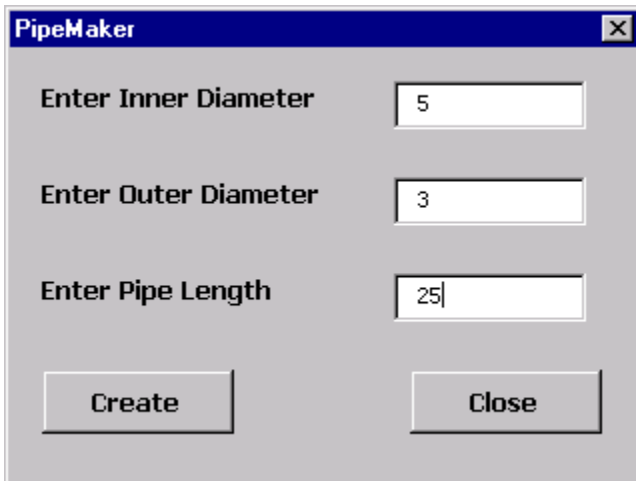
A part file is opened and our pipe is  
created.



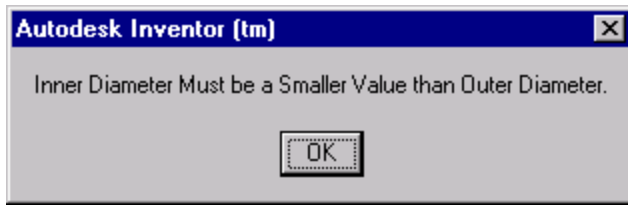
Check your part to see if it uses the dimensions you assigned.



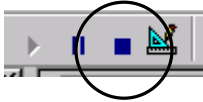
If you highlight the X-Y work plane you see that the pipe was created as a mid-plane extrusion.



Check your error trapping by entering in a larger inner diameter than outer diameter. Then press Create.



Our message box appears.



Select the Stop button.

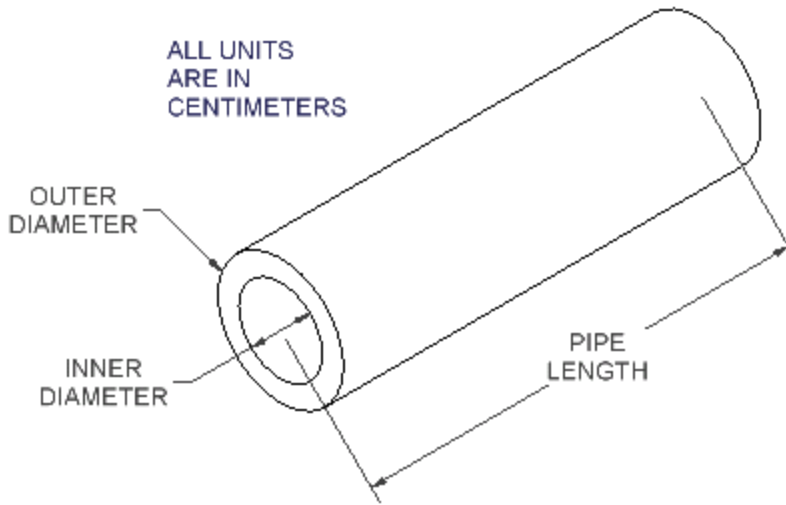


Save the project.

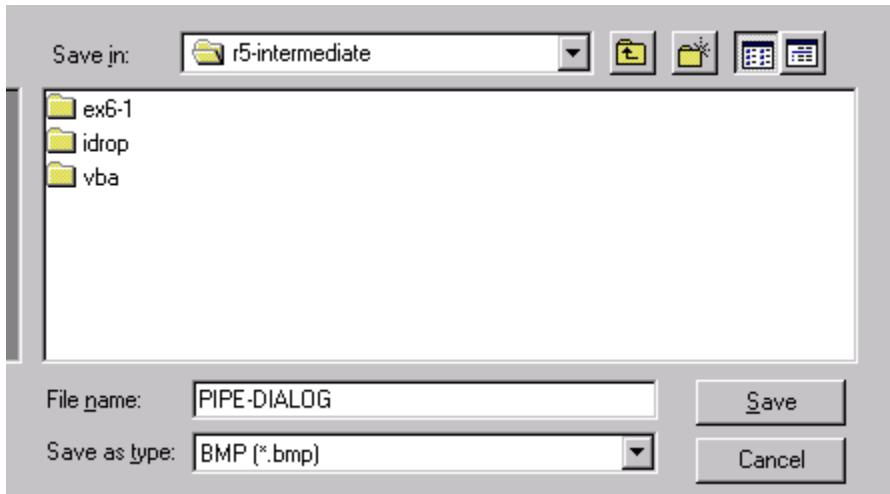
### **Exercise 15-3: Adding a Picture**

File Name: PipeMaker.ivb  
Estimated Time: 60 minutes

Create a drawing from one of the pipes you create with the dialog.



We can then use this drawing to add a picture to the dialog box. The image will help users figure out what the pipe will look like.

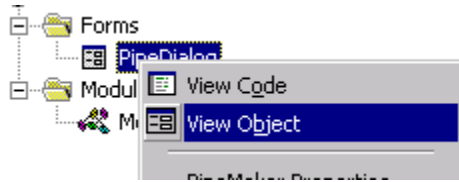


You can use the Save As option to save the drawing as a bmp or use a screen capture program to create your image.

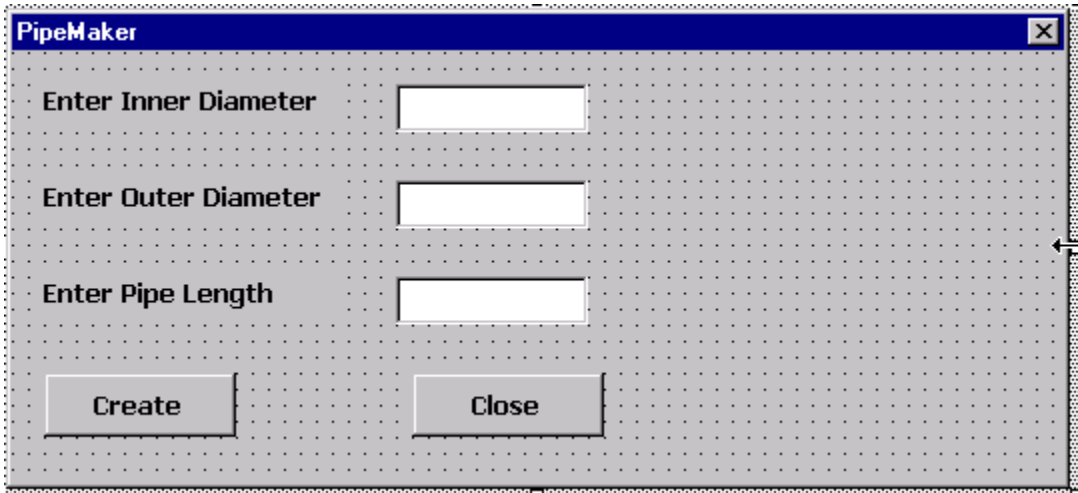
Lesson 15  
Visual Basic

---

Activate the Visual Basic Editor.  
Load the pipemaker.ivb project.



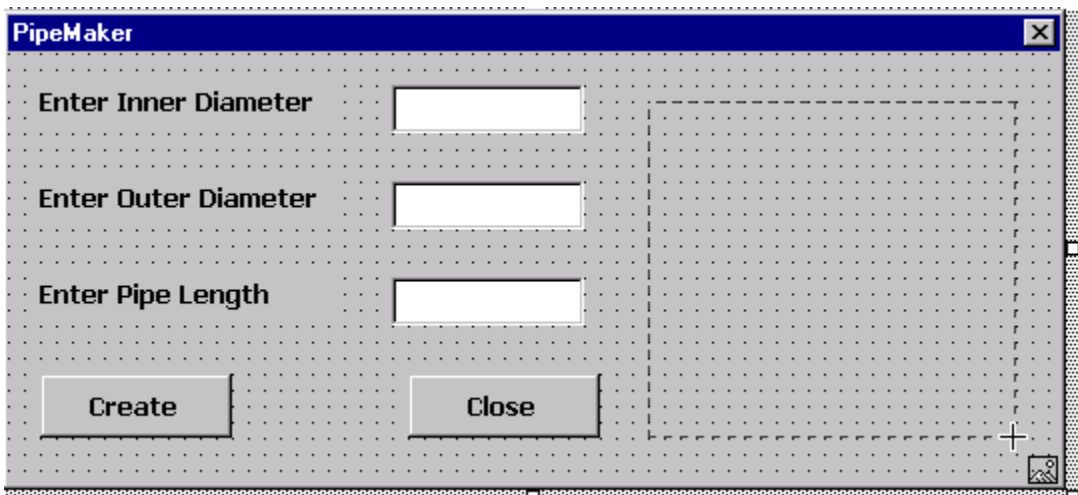
If you don't see the dialog box, you can open it.  
Highlight PipeDialog under Forms.  
Right click and select 'View Object'.



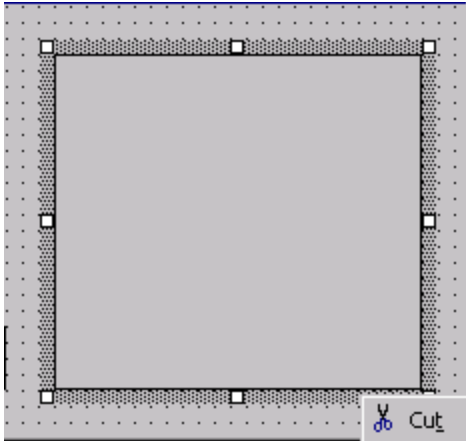
Stretch out the dialog box to make room for a graphic.



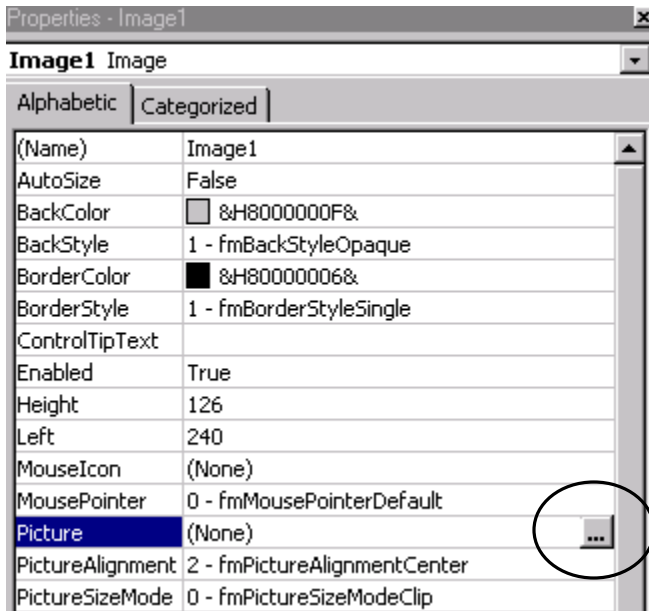
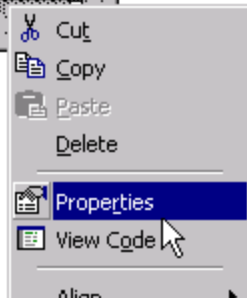
Select the Image tool from the Toolbox dialog.



Select the two corners that will define the image window.

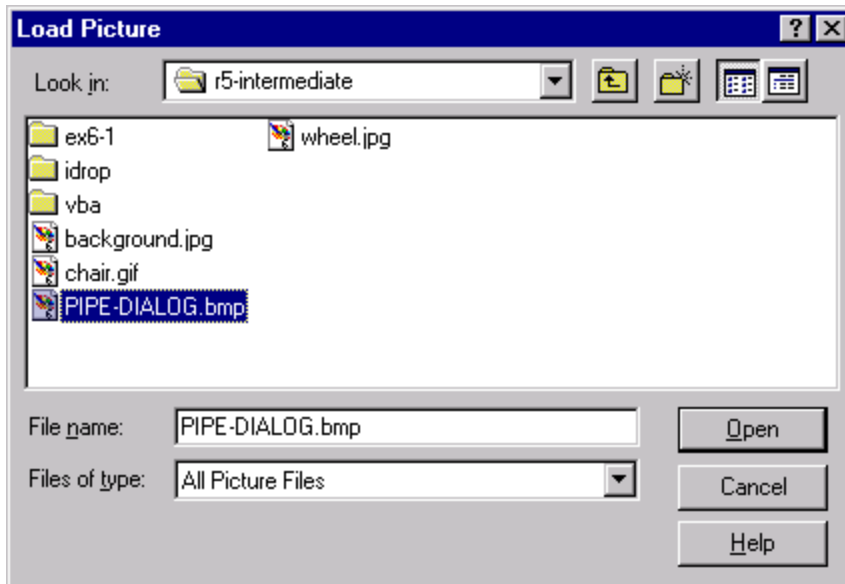


Highlight the image frame.  
Right click and select 'Properties'.

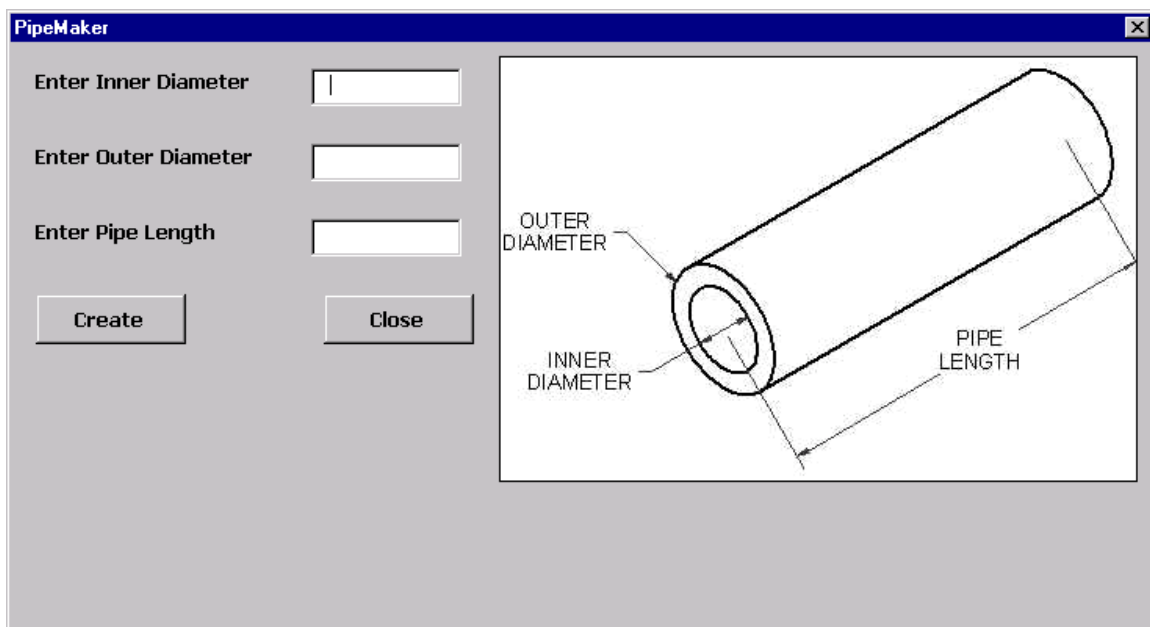


Locate Picture in the list.  
Select the browse button.

Lesson 15  
Visual Basic



Locate the bmp file you created from the drawing.



I found that jpeg files provide a better quality than a bitmap file. You may want to experiment if you decide to use graphics in your dialogs to see which file type gives the best results.

Save your project file.

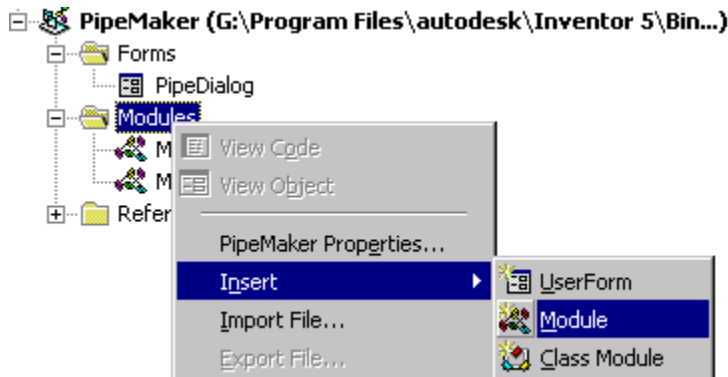


## Exercise 15-4: Creating a Macro

File Name: PipeMaker.ivb  
Estimated Time: 60 minutes

We can load and run a macro without having to open VBA.

Load the PipeMaker.ivb project in the Visual Basic Editor.



Highlight the Modules folder.  
Right click and select  
Insert->Module.

```
Sub CreatePipe()  
    PipeDialog.Show  
End Sub
```

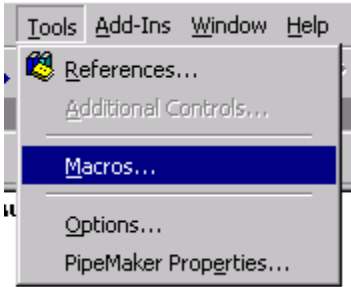
We will call our macro CreatePipe.  
All our macro does is start the dialog box we created.  
The code for the dialog box will take it from there.

Type in the three lines:

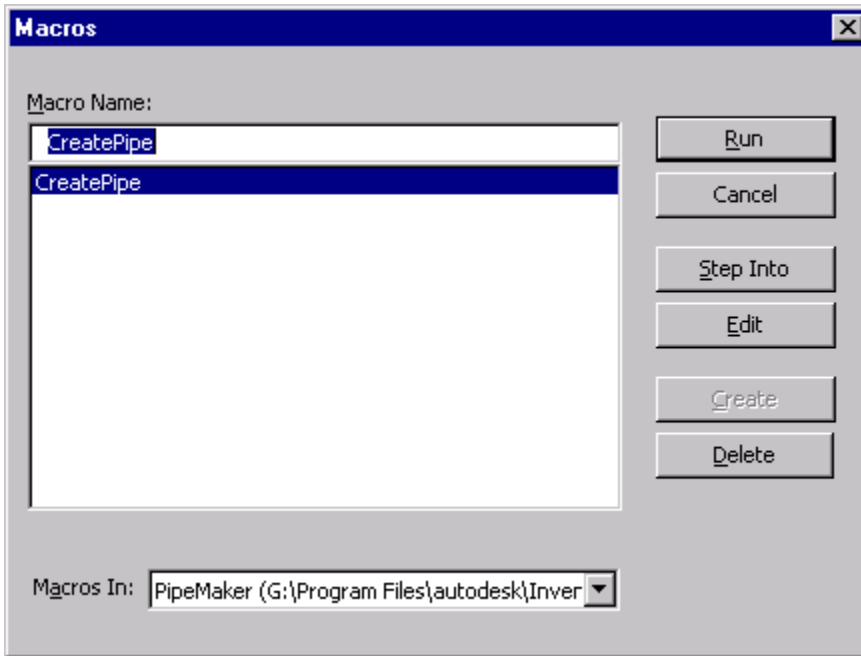
```
Sub CreatePipe()  
    PipeDialog.Show  
End Sub
```

Lesson 15  
Visual Basic

---



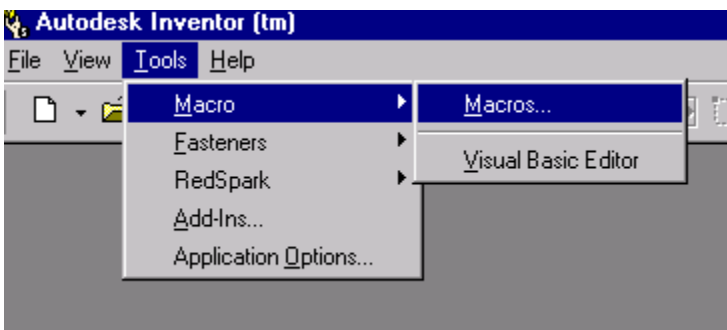
Go to Tools->Macros.



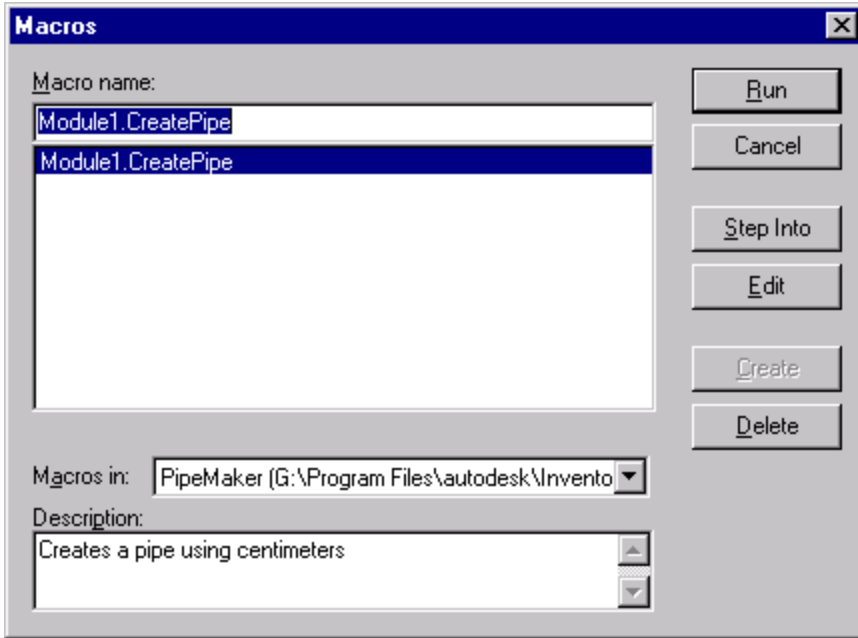
We see the CreatePipe Macro in the list.  
Press 'Run' to run the macro.

Save the project file.

Close the Visual Basic Editor.



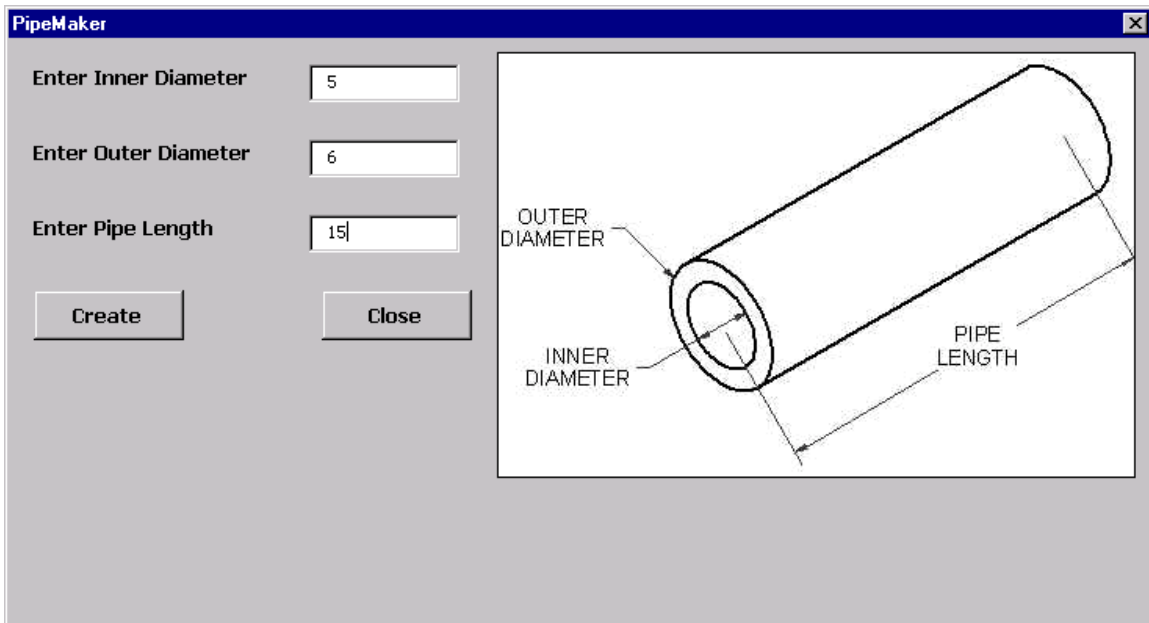
Close any open part or  
assembly files.  
Go to Tools->Macros->  
Macros.



Our Macro appears in the Macro list.

Press 'Run'.

Your dialog appears and your program should work fine.



You can continue to run the macro regardless of whether or not you have other files open. Each run of the macro will create a new part file with a pipe.

## ***Recommended Resources:***

### **Using Visual Basic with AutoCAD by Andrew G. Roe**

This book contains several projects that can be used to help you get started with Inventor. The syntax will be different for Inventor parts, but you will get some good guidance. Mr. Roe uses a similar project for making pipes in his text and you can compare what he does in AutoCAD versus how the Inventor VBA works.

### **Visual Basic 6 for Dummies by Wallace Wang**

This book is a good choice for users who are not familiar or comfortable with programming. Most of the examples are not appropriate for CAD work, but you learn how the VBA tools work.

### **Learn to Program with Visual Basic 6 by John Smiley**

I really enjoy the conversational tone used by the author. Again, the examples in the book are not applicable to CAD, but you get several good examples of how the code works.

### **VBA for Dummies by Steve Cummings**

A good resource to fill in the blanks between the other books since this book is specifically about VBA, which is a limited version of Visual Basic.

Inventor also comes with several sample programs and a help manual specifically for VBA. While the on-line documentation is not as good as any of these texts, it is the only place for you to find the objects and methods specific to Inventor.