sDotson.com

Autodesk Inventor Tutorials

by Sean Dotson
www.sdotson.com
sean@sdotson.com

VBA Functions in Parts Part Two

Latest Revision: 3/17/03
For R6

In the first part of this tutorial we setup a basic function to change the size of a part based on another parameter. If you do not understand all of the concepts that were presented in that tutorial I suggest you study it a bit longer as this one is going to be more advanced.

I also suggest you read the whole tutorial before trying to follow along as there are a few "gotchas"…

The data set for this tutorial can be downloaded from:
http://www.sdotson.com/tutparts/VBAFunctions2.zip

Unzip the dataset and move the **AirCyl.xls** Excl sheet to the root of your C:\ drive. (If you cannot access, or do not have a C drive, place the Excel sheet in another directory and later I will show you how to change the path to this file.)

Now open **AirCyl.ipt.** It will look like Figure 1.

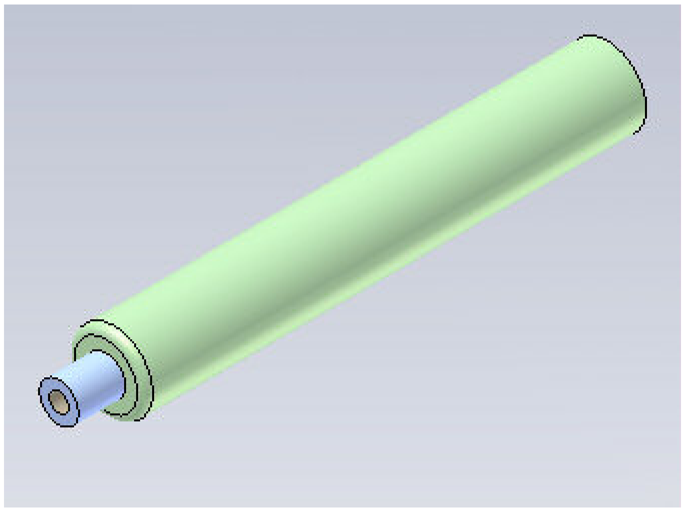
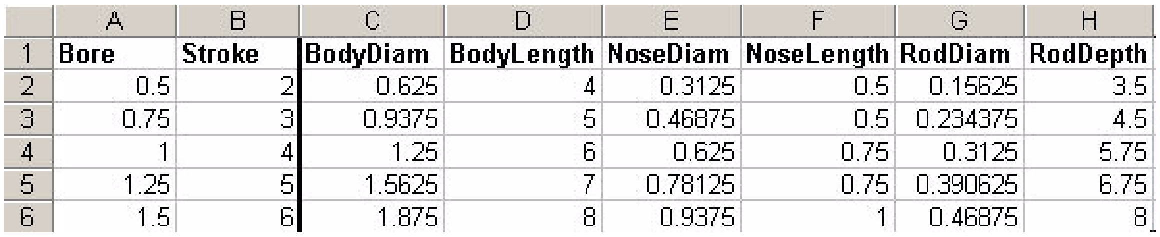


Figure 1 - Basic Air Cylinder

Now open the Excel file **AirCyl.xls** **(**See Figure 2)



	A	B	C	D	E	F	G	H
1	Bore	Stroke	BodyDiam	BodyLength	NoseDiam	NoseLength	RodDiam	RodDepth
2	0.5	2	0.625	4	0.3125	0.5	0.15625	3.5
3	0.75	3	0.9375	5	0.46875	0.5	0.234375	4.5
4	1	4	1.25	6	0.625	0.75	0.3125	5.75
5	1.25	5	1.5625	7	0.78125	0.75	0.390625	6.75
6	1.5	6	1.875	8	0.9375	1	0.46875	8

Figure 2 - Excel Sheet

In this sheet we enter the tabular data. We key off two different columns; **Bore & Stroke**. We shall call these the index columns. We enter the data for the other parameters in the columns next to the index columns. **BodyDiam**, **NoseDiam**, and **RodDiam** all key off**Bore** while **BodyLength**, **NoseLength**, and **RodDepth** key off

Stroke. There is no relationship in the Excel sheet for these keys, just a mental relationship we have made for use in the function.

Close the Excel sheet (it might help to print it out first to refer to as we create the function) and return to the part. Examine the sketches and features that make up the air cylinder. Note where the different parameters are being used to create certain features. Also keep in mind that the Excel sheet header cells do not have to be the same name as the parameters. We make them the same however to make it easier to keep track of the variables.

Rather than have you type out the function, I have already created it for you. Go to **Tools>Macros>Visual Basic Editor** and find the function associated with this part file. Copy the text in the **Function.txt** (included in the data set) file into the Visual Basic Editor.

The code for the functions is listed below:

```
Public Function ExcelLookup(arg1 As Double, arg2 As Double, arg3 As Double) As Double

    Dim sDocName As String
    Dim iRow As Long
    Dim XL As Variant
    Dim xlWS As Variant

    sDocName = "c:\AirCyl.xls"
    Set XL = GetObject(sDocName)

    If XL Is Nothing Then
        MsgBox "Failed to open '" & sDocName & "'", vbCritical
        Exit Function
    End If

    Set xlWS = XL.ActiveSheet

    Dim colXLparams As New Collection

    iRow = 2 'skip the header row
    Do While xlWS.Cells(iRow, arg1).Value <> ""

        If xlWS.Cells(iRow, arg1).Value = arg2 / 2.54 Then
            ExcelLookup = xlWS.Cells(iRow, arg3).Value
            Exit Function
        End If

        iRow = iRow + 1

    Loop

    On Error Resume Next

    'detach from XL
    Set xlWS = Nothing
    Set XL = Nothing

End Function
```

Let’s look at each line of code:

```
Public Function ExcelLookup(arg1 As Double, arg2 As Double, arg3 As Double) As Double
```

This line defines the name of the function and three arguments (arg1,arg2,arg3)

```
    Dim sDocName As String
    Dim iRow As Long
    Dim XL As Variant
    Dim xlWS As Variant
```

These lines dimension some variables we will require in our function. We define the sDocName as a string (this will be our Excel filename), iRow as a Long (this will be the row Number), and XL and xlWS as Variants. At this point I will not go into details about these Variants.

```
    sDocName = "c:\AirCyl.xls"
```

This line sets the variable sDocName to the location and name of the Excel sheet. In this case I have named the file AirCyl.xls and placed it in the root of the C drive. If you want to change the location of this file you can do so in this line.

```
    Set XL = GetObject(sDocName)
    If XL Is Nothing Then
        MsgBox "Failed to open '" & sDocName & "'", vbCritical
        Exit Function
    End If

    Set xlWS = XL.ActiveSheet

    Dim colXLparams As New Collection
```

This section checks to make sure a valid Excel file is attached and end the function if it's not. Again, I am focusing on the Inventor specific part of this code. You should be able to reuse all of this code without changing it.

```
    iRow = 2 'skip the header row
    Do While xlWS.Cells(iRow, arg1).Value <> ""

        If xlWS.Cells(iRow, arg1).Value = arg2 / 2.54 Then
            ExcelLookup = xlWS.Cells(iRow, arg3).Value
            Exit Function
        End If

        iRow = iRow + 1

    Loop
```

This is the true heart of the code. We set the first **iRow** to 2. This means we will ignore the first row of the Excel sheet. We can use this space for column headers.

We then loop through the rows and look at the row and the column that is defined by **arg1**.

Let's step back for a second to discuss how this function is suppoed to work:

Arg1 is the column of the value we are keying off of. If we wanted to find values that keyed off of the **Bore** we would set arg1 equal to 1 (the column that represents the values for Body Diam). **Arg2** is the value in the **Arg1** column we are looking to match. **Arg3** is the column from which we want to select the resulting value.

For example if we wanted to use this function to find the resulting **NoseDiam** for a **Bore** of 1" the values of the arguments would be:

Arg1 = 1 (the **Bore** column)
Arg2 = 1" (value we want to match)
Arg3 = 5 (the **NoseDiam** column)

So in our function we loop through the rows until we find a value in column **arg1** that matches the value as specified by **arg2** and return the value in this same row under column **arg3.** It's a bit confusing but read it a few times and it will start to make sense.

Again we divide **arg2** by 2.54 to obtain the correct units of cm, Inventor's internal units.

In our example we would return a value of 0.625" for our given arguments.

```
On Error Resume Next
    'detach from XL
    Set xlWS = Nothing
    Set XL = Nothing
    End Function
```

The rest of the code detaches Inventor from Excel, empties some variables and then ends the function.

So now we need to create the parameter equations that will call this function. We need to do this in a very particular manner due to the way parameters are evaluated. In Inventor the parameter is evaluated after each keystroke. This causes Inventor to try to run the function before you have completely typed the equations and can cause errors. For this reason I recommend you type the equation in a text editor such as Notepad and then paste it into the parameter equation box at one time.

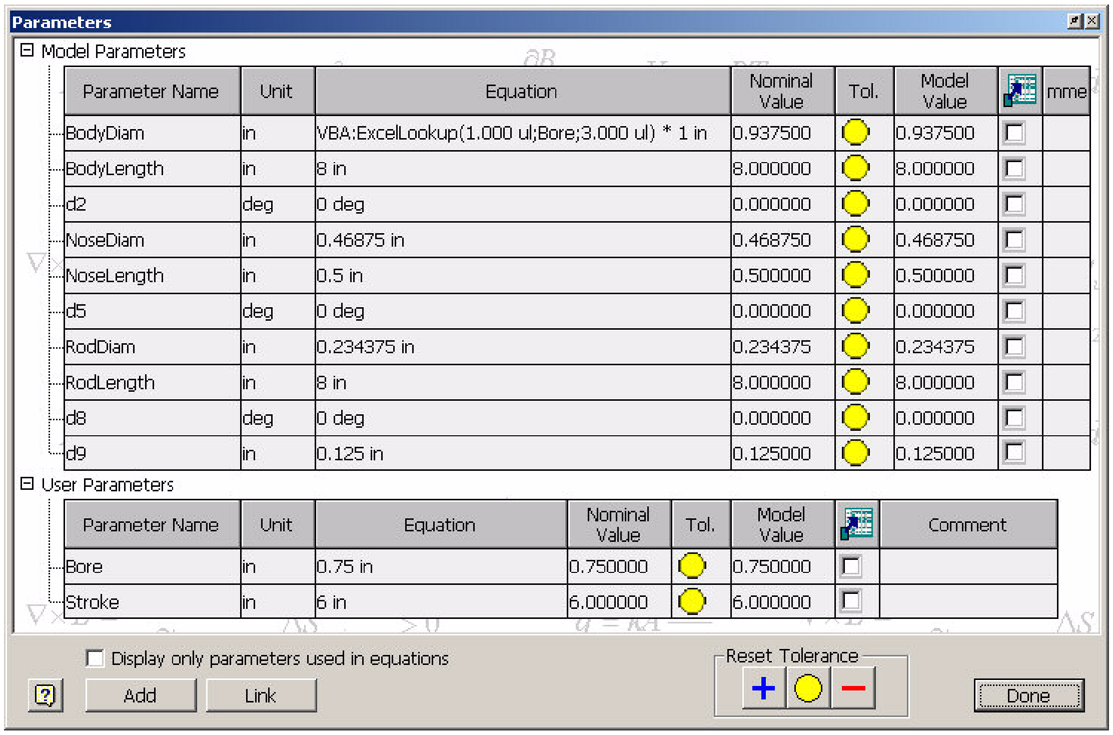
Open the file **Equations.txt**. Copy the value of the equation for the parameters **BodyDiam** and then return to Inventor. Open the parameters dialogue and paste in the value in the equation box. (See Figure 3)

Let's analyze what we pasted into the part file.

VBA:ExcelLookup(1.000 ul;Bore;3.000 ul) * 1 in

Arg1 is set to 1 as we are keying off the Bore. **Arg2** is set to **Bore** (another parameter name) so it will take the value of the User Parameter **Bore** and use it to find a matching value in the Excel sheet's **Bore** column. (Note the column name and parameter do not have to be the same, it just makes it easier to keep track of the variables.) **Arg3** is set equal to 3 as we want to find the resulting **BodyDiam** for the Bore of 1". Notice that **BodyDiam** in the Excel sheet is indeed the third column. We multiply this by 1 in to convert it from centimeters (the default value return by the function) into inches.

Continue to copy and paste the other values in the text file into the parameter dialogue. See Figure 4.



Parameters

⊟ Model Parameters

Parameter Name	Unit	Equation	Nominal Value	Tol.	Model Value		mme
BodyDiam	in	VBA:ExcelLookup(1.000 ul;Bore;3.000 ul) * 1 in	0.937500	●	0.937500	☐	
BodyLength	in	8 in	8.000000	●	8.000000	☐	
d2	deg	0 deg	0.000000	●	0.000000	☐	
NoseDiam	in	0.46875 in	0.468750	●	0.468750	☐	
NoseLength	in	0.5 in	0.500000	●	0.500000	☐	
d5	deg	0 deg	0.000000	●	0.000000	☐	
RodDiam	in	0.234375 in	0.234375	●	0.234375	☐	
RodLength	in	8 in	8.000000	●	8.000000	☐	
d8	deg	0 deg	0.000000	●	0.000000	☐	
d9	in	0.125 in	0.125000	●	0.125000	☐	

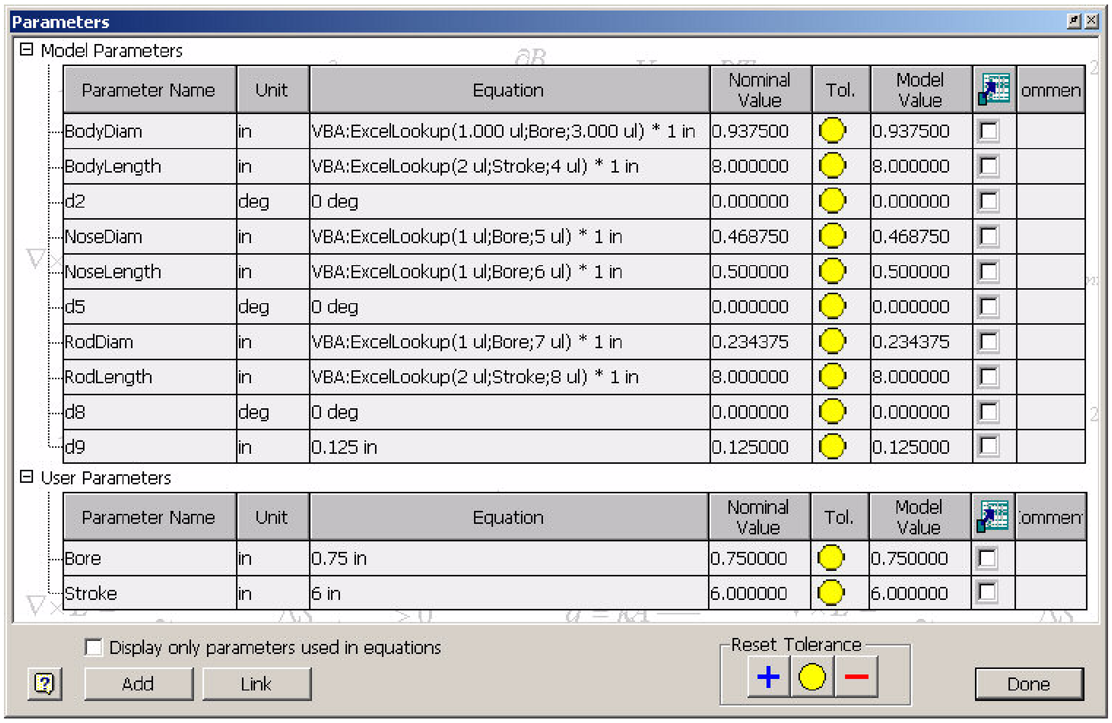
⊟ User Parameters

Parameter Name	Unit	Equation	Nominal Value	Tol.	Model Value		Comment
Bore	in	0.75 in	0.750000	●	0.750000	☐	
Stroke	in	6 in	6.000000	●	6.000000	☐	

☐ Display only parameters used in equations

Add Link Reset Tolerance Done

Figure 3 - Entering Parameter Equations



Parameters

⊟ Model Parameters

Parameter Name	Unit	Equation	Nominal Value	Tol.	Model Value		ommen
BodyDiam	in	VBA:ExcelLookup(1.000 ul;Bore;3.000 ul) * 1 in	0.937500	●	0.937500	☐	
BodyLength	in	VBA:ExcelLookup(2 ul;Stroke;4 ul) * 1 in	8.000000	●	8.000000	☐	
d2	deg	0 deg	0.000000	●	0.000000	☐	
NoseDiam	in	VBA:ExcelLookup(1 ul;Bore;5 ul) * 1 in	0.468750	●	0.468750	☐	
NoseLength	in	VBA:ExcelLookup(1 ul;Bore;6 ul) * 1 in	0.500000	●	0.500000	☐	
d5	deg	0 deg	0.000000	●	0.000000	☐	
RodDiam	in	VBA:ExcelLookup(1 ul;Bore;7 ul) * 1 in	0.234375	●	0.234375	☐	
RodLength	in	VBA:ExcelLookup(2 ul;Stroke;8 ul) * 1 in	8.000000	●	8.000000	☐	
d8	deg	0 deg	0.000000	●	0.000000	☐	
d9	in	0.125 in	0.125000	●	0.125000	☐	

⊟ User Parameters

Parameter Name	Unit	Equation	Nominal Value	Tol.	Model Value		ommen
Bore	in	0.75 in	0.750000	●	0.750000	☐	
Stroke	in	6 in	6.000000	●	6.000000	☐	

☐ Display only parameters used in equations

Add Link Reset Tolerance Done

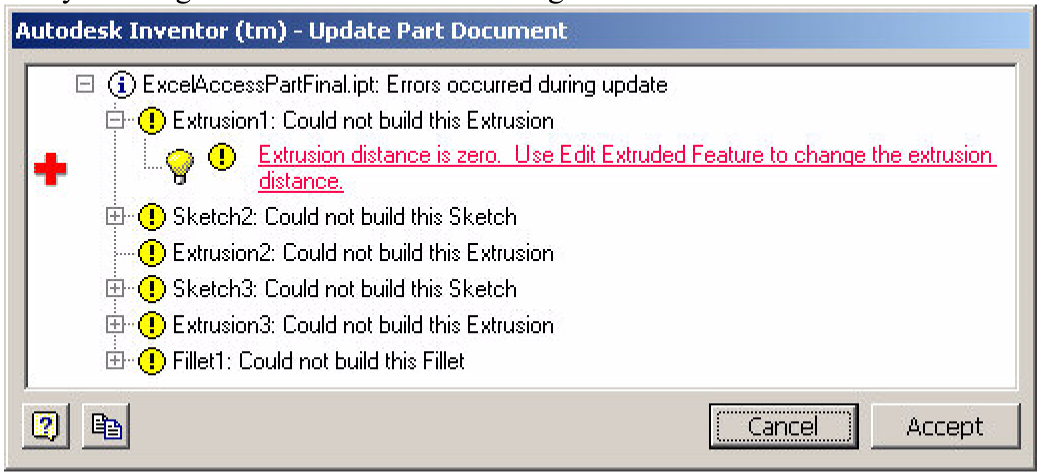
Figure 4 - All Equations Entered

Notice that in the equation for the parameter **BodyLength** we set **Arg1** equal to 2. This is because we want to key off the **Stroke** column, not the **Bore** column. You could continue this logic for any number of "key" columns.

Once all parameters have been pasted. Click Done. Notice that the update icon is highlighted. Click on it and the model should update. It will not change as the current values of **Bore** and **Stroke** have not been changed. Enter the parameter menu once again and change **Bore** to 1" and **Stroke** to 5".

Now exit and update the model. It should have changed to reflect the values as shown in the Excel sheet.

So we now have a part that uses a lookup table to determine its values based on two user parameters. If you enter a value that is not on the lookup chart the function will return zero and you will get an error like the one in Figure 5.



Autodesk Inventor (tm) - Update Part Document

ExcelAccessPartFinal.ipt: Errors occurred during update
Extrusion1: Could not build this Extrusion
Extrusion distance is zero. Use Edit Extruded Feature to change the extrusion distance.
Sketch2: Could not build this Sketch
Extrusion2: Could not build this Extrusion
Sketch3: Could not build this Sketch
Extrusion3: Could not build this Extrusion
Fillet1: Could not build this Fillet

Figure 5 - Error Message (Non-Valid Entry)

I feel this is a good way to let the user know we may have entered an incorrect value. You could get more involved in the function and present the user with a message box letting them know they entered an incorrect value and allow them to change it before updating the model. The code to do this would be

```
If ExcelLookup = 0 Then
MsgBox "You have entered a value that is not in the lookup table."
End If
```

Placed between the **`Loop`** and **`On Error Resume Next`** lines. See the file **ExcelAccessPartFinal.ipt** for an example of a finished part with this code.

You could then place this part on your network or in your templates folder and generate new parts off of it while maintaining one Excel file. The Excel file can reside on the network and you do not have to worry about changing sources for linked Excel sheets. You do, however, have to remember to package this Excel file along with the part if you are moving the part to a location where it cannot reach the Excel sheet. You will also have to edit the macro to point it to the new location.

In the third part of this tutorial I will discuss how to use these VBA functions to make complex motion much easier. (Of course the easiest was is to use **Animator** ☺)