

Getting started with DeltaCad macros

Jim Brown

jimbrown@telkomsa.net

Why am I writing this.....	2
So, what's a macro... ..	2
Let's run one.....	3
Our first macro... ..	4
Step 1- open the editor	5
Step 2- type in the code for a Message Box and test it.....	5
Step 3- Add your macro to the macro list.....	7
Step 4- Run the macro from within DC:	7
Variables and Return Values.....	8
Let's use a DC macro command	10
Using a return value on the fly	10
Allocate the return value to a variable.....	11
Comments in your macro.....	12
Getting input from the user... the InputBox	12
Controlling program flow	14
More on the MsgBox.....	14
If..... and GoTo.....	16
Other ways of controlling the flow	17
For ... Next.....	17
What next?	18

Why am I writing this....

Well when I started trying to work with DC macros, I didn't know where to start. I figured it out by digging through some existing macros, and looking in the help. I thought I could save you that bother, especially if you don't have much experience in programming in the first place.

This guide is exactly what it says- it's a *Getting Started*, it's not a definitive work on DC macros. I am not an expert on any of the following...

- DC
- DC macros
- Visual Basic
- Visual Basic macros
- Anything at all, really ☺

But I hope you get something out of this....

So, what's a macro...

It's always easy to start with a proper definition- let's see what we can find:

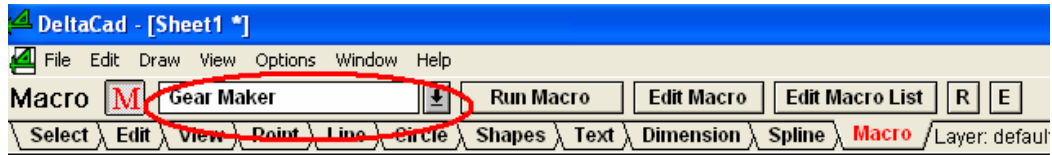
- Wikipedia says that the idea [of macros] is to make available to the programmer a sequence of computing instructions as a single program statement, making the programming task less tedious and less error-prone
- Chambers says that a macro is a single instruction that brings a set of instructions into operation.

In the context of DeltaCad, a macro would seem to be a short-hand way of lumping together a whole lot of things you can already do in DC, but under the umbrella of one single operation. If there's something you do often in DC, but find it's tedious to repeat a whole series of menu selections to do it, then a macro might be what you need.

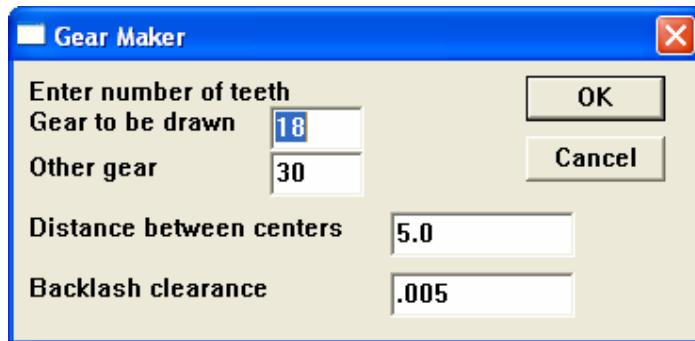
In short, a macro is a program which implements a series of DC capabilities every time you run the macro. It runs the same series of instructions each time, except that you may influence the outcome if the macro was written to allow that. For example, if there was a macro to draw a particular compound shape by constructing a series of individual DC lines and shapes, it might ask you where you want to place the shape and how big it should be. When it's finished, it might ask you if you want to do another one before it stops. (You might draw a simple house by sticking a triangle on top of a square, for example.)

Let's run one....

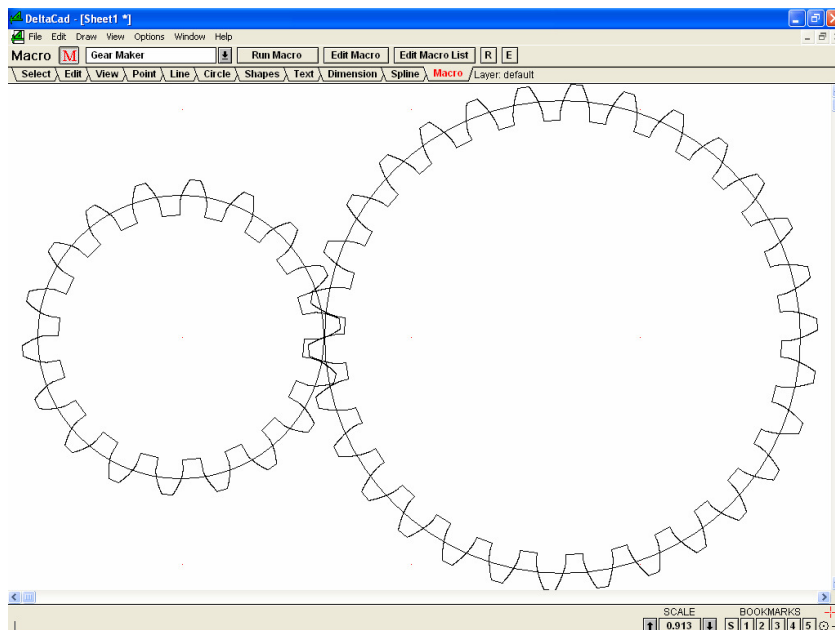
You get to DC's Macros by choosing the Macro tab- you should then get this...



You run a macro by choosing the one you want from the drop down menu- circled red above. (Ok, it's an oval, but I'm not sure there's such a word as "ovalled".) Then hit the Run Macro button- try it with the Gear Maker, and you'll be asked a few questions as shown below...



Then when you fill in the numbers and hit OK, you'll get something like this...

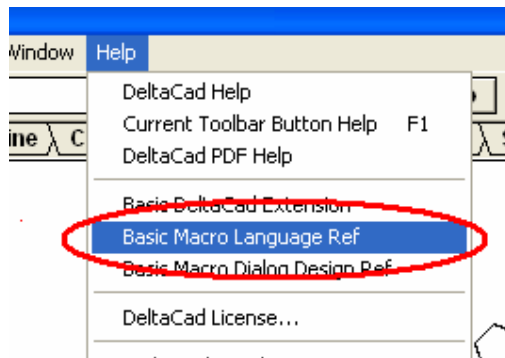


You can imagine the effort this would have taken had you done it manually with DC's lines, circles and shapes. That's exactly what macros are for...

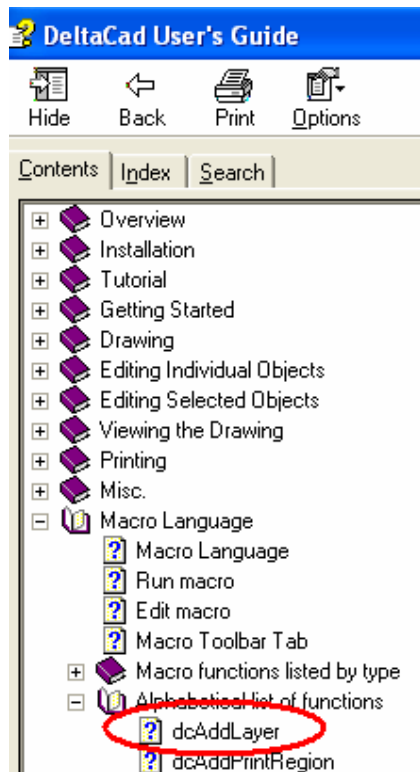
Our first macro...

We have to start somewhere... let's see how the Message Box works.

There are two kinds of commands in DC's macro language. The first, which we'll look at now, are not DC specific. They came bundled with the macro language engine that DC bought in from Cypress. The Message Box is one of these. You can read all about the generic commands in DC's Help, as Basic Macro Language Ref, as shown below...



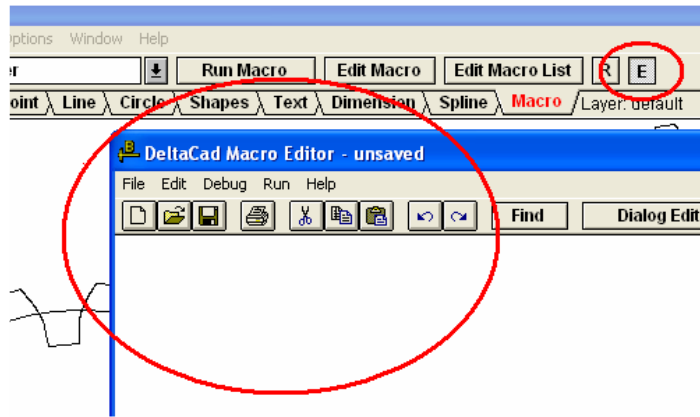
The DC specific commands are explained in the DeltaCad User's Guide- these commands all start with the letters dc, as you can see here...



We'll look at them later.

Step 1- open the editor

This is easy, just click the E button, and the editor opens up...



(Don't confuse this with the Edit Macro button, which opens the current macro (the one visible in the drop down) for editing.)

Step 2- type in the code for a Message Box and test it

Have a look at the MsgBox command in the manual-

```
MsgBox ( msg, [type] [, title])
```

Whatever's in the brackets is optional, so the simplest message box is

```
MsgBox msg
```

where msg is the text of the message you want to display.

Look further down the MsgBox help, till you get to the example. Ignore most if it... for now notice that a macro needs a beginning and an end. You'll see there's a command at the top:

```
Sub Main
```

and one at the bottom

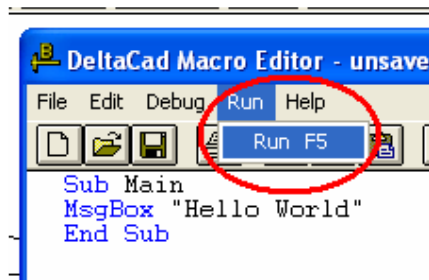
```
End Sub
```

and we need to put our MsgBox command in between them. If we want to display the message "Hello World" in a message box, we need to type the following code into the editor- note the quotes:



Now save your work (File | Save As), because you don't want to lose all that typing.

Run the macro as shown here:



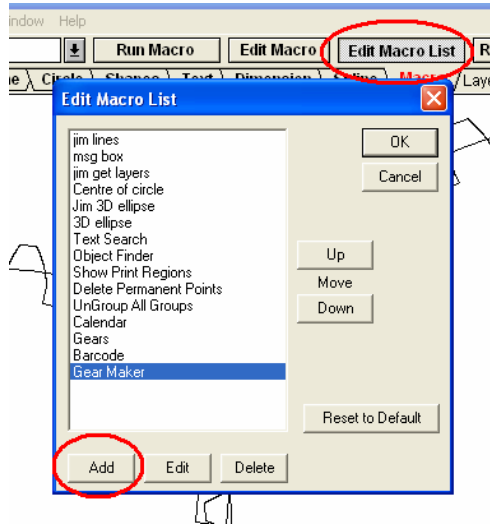
You should get this on the screen:



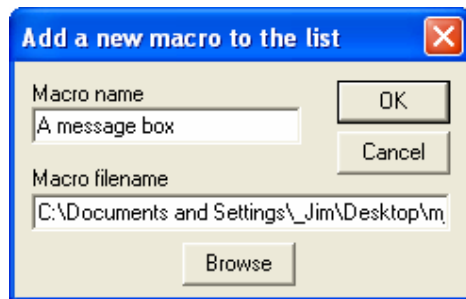
Disappear the message by hitting OK, and you'll go back to the editor, which you should then close in the normal Windows fashion.

Step 3- Add your macro to the macro list

Right now, your macro is invisible to DC- you need to add it to DC's macro list by choosing Edit Macro List and Add as shown:



Then browse to where you saved it, and give it a name by which it will appear in the list:



Step 4- Run the macro from within DC:

Your macro will now be in the drop down, hit Run Macro and Voila!



Variables and Return Values

If you have never used a programming language before, you may not have encountered these terms- here's a brief explanation.

A **variable**, in the world of programming, is a pigeon-hole in the computer memory where something can be stored. As the name variable implies, the contents of the pigeon hole can change, although the name of the pigeon hole will remain constant. If you go to the bank and ask the teller for your balance, they'll ask you for your account number. In the bank's system, there will be a variable called AccountNumber or AcctNum or AccNo or something- while the teller is dealing with you, the content of that pigeon hole will be your account number, say 12345. Other variables might be CurrentBalance and CreditLimit, which will temporarily hold your balance and limit. When the teller has finished with you, the contents of AccNo might change to 54321 for the next client, and the other variables will then hold that client's balance and so on.

In most programming languages, there is a requirement or at least an ability to allocate a pigeon hole before it gets used. If I have a need to put the value 555-9999 into a pigeon hole which I want to be known as TelNum, I could in some languages say something like

```
TelNum = 555-9999
```

and the system would allocate a pigeon hole, call it TelNum and put 555-9999 into it all in one go. Other programming languages insist you allocate and name the pigeon hole first, then put the value in. In the process of allocating the pigeon hole, it is usual to tell the system what kind of thing you're going to put in it, eg a bunch of characters like a name or address (usually referred to as a string), or a whole number (referred to as an integer) or some other kind of value.

DC (as a type of Basic), uses the command Dim, and before I use the variable TelNum, I can declare it (as they call this allocation) by saying something like:

```
Dim TelNum  
TelNum = 555-9999
```

Read up on the Dim statement in the help, to see what types of variable you can declare. The vanilla use of Dim as above, declares the variable as a Variant, which is a sort of catch-all, but if you specifically want the variable to be an Integer, you would say

```
Dim YearOfBirth as Integer.
```

When you fire up the macro editor and run a macro, DC isn't fussed about whether or not you declare your variables before you use them. Most computer programmers say it's good practice to declare them first- won't go into that now- and there is a way you can get DC to force you to. Above the Sub Main line of your macro type the command

Option Explicit

Then DC will not allow you to use an undeclared variable. If you then try to put a value into a variable which doesn't exist, DC will shout at you and your macro won't work. It's up to you.

Right, so what's a **return value**? Some commands will do something and then bring you an answer- that answer is the return value. It's the value which the command returns to you.

In the banking example, there might be a command which calculates the balance for an account whose number we feed to the command...

```
Dim AccountNumber
Dim AccountBalance

AccountNumber = 12345
AccountBalance = WhatIsTheBalance (AccountNumber)

Display AccountNumber; AccountBalance
```

That says....

- Allocate space for two variables, AccountNumber and AccountBalance
- Give AccountNumber a value... this would normally be keyed in on the screen, not given as a line of program like I've got it here
- Run the command WhatIsTheBalance, on the account whose number is the current value of the variable called AccountNumber
- Put the answer (the return value) into the variable AccountBalance
- Show the values on the screen.

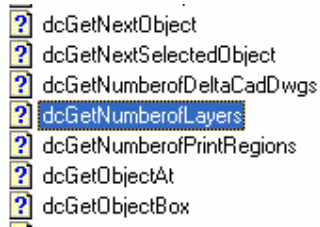
Some DC commands return values, some don't. When they do, the help will tell you.

Ok, let's move on to use some of this new knowledge and write a more useful macro....

Let's use a DC macro command

The MsgBox was a pretty trivial example, but served to show how to create, test and save a macro, and how to put it into DC's list. That macro didn't do much though so let's use one of the DC extension commands to give us some information about the drawing.

If you glance down the list of DC extensions, you'll see some of them "get" something for you; we'll use the dcGetNumberOfLayers one here.



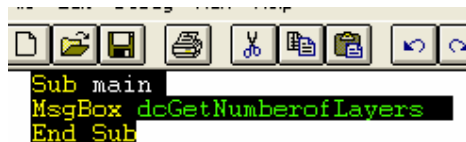
First have a look at the write-up on dcGetNumberOfLayers in the help, then read on....

The dcGetNumberOfLayers command goes and finds how many layers there are, and it's return value is that number.

There are two ways we can use that return value...

Using a return value on the fly

We can simply use the return value in for example a message box. Create a new macro or edit your previous one to look like this...



Notice a subtle difference between this coding and the previous MsgBox- there are no quotes around dcGetNumberOfLayers, whereas there were earlier when we had "Hello World". Reason is, we wanted the string Hello World to be the message text; now we want the command dcGetNumberOfLayers to run, and feed its return value to the message box. Put quotes around dcGetNumberOfLayers and see what happens....

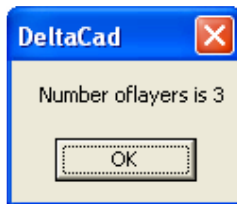
Now, when you run the macro, you will get something like this. (Before you run it you might like to add a new layer or two so you can see that works ok.)



Just throwing the answer 3 onto the screen might seem a bit cryptic, so modify your code as shown below. The & character joins the text in the quotes to the return value from dcGetNumberOfLayers. Note the space before the closing quote.

```
Sub main
MsgBox "Number oflayers is " & dcGetNumberOfLayers
End Sub
```

Now your message box is a bit more helpful:



Allocate the return value to a variable

It might be that you need to use the number of layers more than once in your macro- to make it simpler, we can put the return value into a variable, then simply drop the variable in when we need it. That will save you typing, and probably also a tiny amount of computing power since it won't need to run the get command every time you want the value.

Try code like this... where we put the return value into a variable called Jim and then use that value more than once.

```
Sub main
Dim Jim As Integer
Jim = dcGetNumberOfLayers
MsgBox "Number of layers is " & Jim
MsgBox "Number of layers is still " & Jim & " :)"
End Sub
```

Danger of course is, that the actual number of layers might change and the value in the variable will be out of date.

Comments in your macro

It's good practice to put some comments in your code. One reason- the selfish one- is so that when you look at the code in a year or two, the comments will remind you what you had in mind at the time. The other reason- the altruistic one- is so that if someone else looks at your code, they'll understand what you were doing. A comment is typing in your macro that DC will ignore when the macro runs, but which is there when you read it.

A comment can be a whole line or a can be tacked onto the end of a line. Comments are denoted by the apostrophe ' – anything after a ' is ignored by DC.

Here's an example...

```
' see what the scale is at the moment... THIS IS A COMMENT
drawingscale = dcGetDrawingScale ' call the code to get the scale SO IS THIS
MsgBox "Drawing Scale = " +Format(drawingscale) ' put answer in a message box AND THIS
'now set the scale to mm
dcSetDrawingScale 25.4
```

By now you will have noticed that DC colour codes the typing in the macro editor- comments are green.

Ok it's time we moved to new level- how can we get the macro to take some info from us....

Getting input from the user... the InputBox

There are at least two ways of doing this, in Getting Started I'm going to use the InputBox. Read up on the InputBox command in the manual. It says that the InputBox returns a string- and that the command is InputBox\$. From my experimenting with it, it doesn't seem to matter and even if you get a string from an InputBox you can use it as a number, like for drawing a line. However, don't take that as a definitive statement...

(Later on you can read up on another method, the Dialog Designer, in the help.)

Here's the format of the InputBox command from the manual-

```
N$ = InputBox$(Prompt$, Title$, Default$, X%, Y%)
```

The important thing is that the command returns a value- that's the whole point after all. In the example below, 4 separate input boxes ask for the beginning and end coords of a line, and then DC draws the line for you.

The prompt is the question you want asked- eg, "What is the x value of the start of the line"; the title is supposed to be a title in the top line of the input box, but when I tried it is always seems to say "DeltaCad" and I can't get it to work. (For that matter, nor does the one in the manual- they want it to say "Greetings" but their example says "InputBox Dialog".) Try the following code...

```
Option Explicit
'this is good practice
'note it goes above the Sub Main line

Sub Main
'dim the variable to hold the drawings scale
Dim ds As Single
'dim the co-ords of the line we're going to draw
Dim xa, ya, xb, yb As Single
'dim the text for the input boxes
Dim Mylinetitle, Mylineprompt As String

'set the scale to mm
dcSetDrawingScale 25.4
' show that the scale is mm
ds = dcGetDrawingScale
MsgBox "Drawing Scale = " & ds

' now draw a line
Mylinetitle = "Where do you want the line?"

Mylineprompt = "What is xa?"
xa = InputBox(Mylineprompt, Mylinetitle)
' the title thing doesn't seem to work....

Mylineprompt = "What is ya?"
ya = InputBox(Mylineprompt, Mylinetitle)

Mylineprompt = "What is xb?"
xb = InputBox(Mylineprompt, Mylinetitle)

Mylineprompt = "What is yb?"
yb = InputBox(Mylineprompt, Mylinetitle)

dcCreateLine xa, ya, xb, yb

End Sub
```

The important part of this code is the four uses of the InputBox, which allocates the 4 coordinates to the 4 variables xa, xb, ya and yb. Those 4 values are then used in the dcCreateLine command which obviously draws the line.

Before that, I decided to use the dcSetDrawingScale command to change the scale to mm, and then proved that was done by using a MsgBox to display the scale with a dcGetDrawingScale.

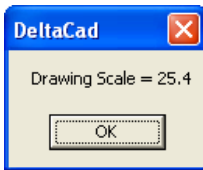
Lastly in this Getting started, I'd like to touch on how you can control the flow through the macro. For example, you might like the above program to ask you if you want to draw another line, or quit the program.

Controlling program flow

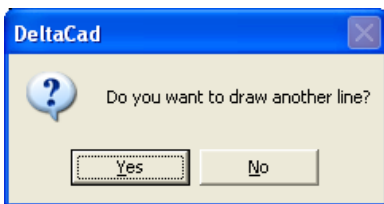
If we want to ask the user if there are more lines to be drawn, one way is to use the MsgBox in a more advanced way....

More on the MsgBox

When we used the MsgBox before, it only had an OK button like this:



The message box has more to that that, and we can easily get it look like this:

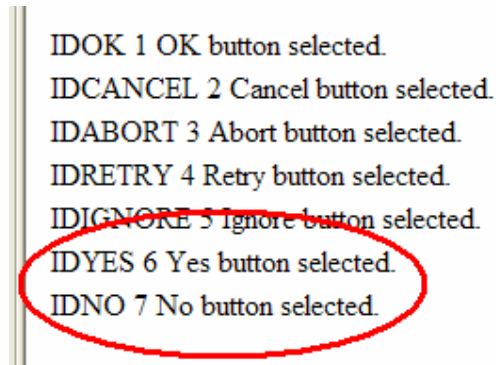


How to do that is based on the message box type, which is explained in the manual. Each button style is allocated a value, such as 0 for only an OK button; each type of icon is also allocated a number such as 48 for an Exclamation. To get the box shown above, we want 4 for a Yes / No and 32 for a Question Mark, so we add 4 to 32 to get 36 and in the MsgBox command we put a type of 36.

```
MB_OK 0 Display OK button only.  
MB_OKCANCEL 1 Display OK and Cancel buttons.  
MB_ABORTRETRYIGNORE 2 Display Abort, Retry, and Ignore buttons.  
MB_YESNOCANCEL 3 Display Yes, No, and Cancel buttons.  
MB_YESNO 4 Display Yes and No buttons.  
MB_RETRYCANCEL 5 Display Retry and Cancel buttons.  
  
MB_ICONSTOP 16  
MB_ICONQUESTION 32  
MB_ICONEXCLAMATION 48  
MB_ICONINFORMATION 64
```

```
'now see if there's another line to draw  
Dim Response As String  
Response = MsgBox ("Do you want to draw another line?", 36)
```

Note that here we have a MsgBox returning a value we have called Response; the value depends on which button was pressed. The return values are also explained in the manual; in our case we will get either a 6 for Yes or a 7 for No.



So, we have a variable Response which now holds either of the values 6 or 7.

(You might like to prove this by adding some code like this...

```
MsgBox ("Return value is " & Response)
```

Then you should get something like this..



or this



depending which one you pressed.)

What do we do with the return value- how does it affect the program flow?
There are a number of ways to control the sequence in which program lines are acted upon.

If..... and GoTo.....

The first way is the simple “if” statement. We want the code that draws the line to be run again if the user presses Yes, otherwise we want the program to end.

In the manual under Control Structures you will find the If statement. It has a number of formats and the simplest is just...

If condition Then Statements...

That means that if (and only if) a certain condition is met, activate the statement. The statement will be ignored otherwise, and the next line of code will be run.

This simple If should work for us- we want the program to go back to the top if Yes is pressed, otherwise we want it to run through to the end.

How do we get it to go to the top? The easiest way, although it’s frowned on by the computer purists, is to use the “GoTo” command. As its name implies, a GoTo goes to a certain part of the program. You have to put a label at the place you want to go to; labels look like this:

ThisIsALabel: ‘ note the colon.

So for our code to jump back to the part of the program that draws the lines, we have to first put a label at the right spot, like this:

```
ds = dcGetDrawingScale
MsgBox "Drawing Scale = " & ds
MyLabel:
' now draw a line
Mylinetitle = "Where do you want the line?"
```

Then we add an If with a GoTo, like this...

```
'now see if there's another line to draw
Dim Response As String
Response = MsgBox ("Do you want to draw another line?", 36)
MsgBox ("Return value is " & Response)
If response = 6 Then GoTo MyLabel
```

Try it and you should find that if you answer “Yes”, you get asked for the details of another line; if you answer “No”. the program will end as before.

Other ways of controlling the flow

Read the Control Structures part of the help, and you will see there are many more ways of controlling flow.

Firstly, the If method has a number of enhancements. For example if there is more than one statement you want to run (we only had the Goto) then you can spread them on many lines after the If; you end the whole thing with an End If. Also, you can use a whole series of Else Ifs- we only had two possibilities (Yes or No) but if you had a variety of possible values for Response, you would need a similar number of elses.

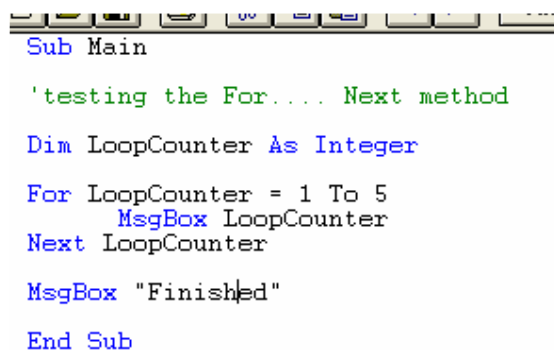
Secondly there is the Select ... Case method- this is similar to the If, in that it offers a number of escape routes. Each possibility is a "case" and has its own code.

Lastly, there are a variety of looping methods, which allow a section of code to run while or until a certain condition is met.

There is also the For.. Next which I'll explain next.

For ... Next...

Here, we want to loop through some code for a range of values of a variable. So we need to declare a variable as our counter. Then we specify what the range is from start to end. A simple example follows, in which we declare a counter, then loop through a MsgBox 5 times, each time showing the counter's value. Each time the Next is encountered, it checks to see if the range is used up- if not it loops again, if it is, it drops out...



```
Sub Main
    'testing the For... Next method
    Dim LoopCounter As Integer
    For LoopCounter = 1 To 5
        MsgBox LoopCounter
    Next LoopCounter
    MsgBox "Finished"
End Sub
```

A more useful example is shown below, where we use a For .. Next to loop through the number of layers in a DC drawing and tell us the name. As before, we use dcGetNumberOfLayers- here we use that number as the upper limit of our loop.

```
Sub Main

Dim NumOfLayers As Long
Dim LoopCounter As Long
Dim LayerName As String
Dim msg As String

NumOfLayers = dcGetNumberOfLayers

For LoopCounter = 1 To NumOfLayers
    LayerName = Space(255)
    dcGetLayerName LoopCounter , LayerName

    msg = "Layer " & LoopCounter & " of " & NumOfLayers & " is " & LayerName
    MsgBox msg
Next LoopCounter
End Sub
```

What next?

Well that's up to you- I hope you've got something out of this short tutorial. I welcome your feedback, especially if you find some errors or have other suggestions to make.

Happy programming-

Jim