To wait for Mach3 to finish what you've started

A frequent requirement is to have to wait for Mach3 when doing several commands, to keep them executing in order. The:

```
While IsMoving
Wend
```

Loop is the typical way of doing this.

ex:

```
code "G0X100"
While IsMoving()
Wend
```

However, this causes the system to make millions of calls to the subsystem to determine if Mach3 is finished. The CPU load rises terribly. A solution is to wait for 100ms or so each time you check unless you need a very tight response time. The solution is to use a syntax as follows..

```
Declare Sub Sleep Lib "Kernel32" (ByVal dwMilliseconds As Long)
.
.
.
Code "G0X100"
While ismoving()
Sleep 100
Wend
```

This will lower your CPU load a great deal and allow things to be much more robust.

For accessing the screen controls

As you may have seen in example code, a macro read and change the data in a DRO. It can also read the state of any LED and simulate the action of clicking a screen button. To access these operations on Mach2 controls you use the codes used internally by Mach3 and its Screen Designer for the DRO, LED or button operation you want to use.

There are, for historical reasons, two different ranges of numbers for each type of control. Some LEDs and DROs start at 800 and some buttons start at 1000. You do not need to worry about this. As we say it is historical!

DROs and LEDs can be defined that have no meaning to Mach3 being solely for you use. There are 255 of each denoted by codes 1000 to 1254. You must refer to them using special functions with "User" in the name to make it obvious that they are not controlling Mach2 itself.

Although we use literal values (like 14) in the examples you are strongly advised to assign the values you want to use to variables at the beginning of your macro and then use the variables in calls to the routines. This will make your program much easier to read. Thus the first LED example in a complete script would be:

```
JoyStickLED = 814
:
:
bJoy = GetOEMLed (JoyStickLEDFn)
```

LEDs

```
Function GetOEMLED (ledOEMCode as Integer) as Boolean
```

A common routine accesses system, OEM and User LEDs. *ledOEMCode* must be in the range 1000 to 1244 for user LEDs and the codes given in this wiki for system and OEM LEDs. The result is True (i.e. non-zero if converted to an integer) if the LED referred to is alight.

User LEDs, only, can be set on or off by:

```
Sub SetUserLED (ledUserCode as Integer, cond as Integer)
```

If `cond` = 1 the LED will be on. If `cond` = 0 then it will be Off

Examples:

```
bJoy = GetOELed (814)                    ' set variable bJoy if Joystick is enabled
If GetOEMLed (29) Then …..     ' see if a Fixture is in use
Call SetUserLED (1002, 1)                ' turn on user LED
```

DROs

```
Function GetOEMDRO (droOEMCode as Integer) as Double
```

Choose the appropriate codes depending on whether you want to access a "system", an OEM or user DRO. droOEMCode will be in the range 800 + for "system DROs and 1000 to 1254 for user DROss. The result is the current value displayed by the DRO.

```
Sub SetOEMDRO (droOEMCode as Integer, newValue as Double)
```

Choose the appropriate codes depending on whether you want to access a "system", an OEM or user DRO. droOEMCode will be in the range 800 + for "system DROs and 1000 to 1254 for user DROss. The routine sets the expression provided for newValue into the DRO. Not all DROs can be written. If you cannot type a value into the DRO on the screen (e.g. X Velocity = 806) then you cannot set it in a script.

```
Sub KillExponent (result as String, smallNumber as String)
```

This routine is provided to address the problem that VB Script is liable to represent small numbers (e.g. 0.0000012) in scientific (exponent) notation. The routine forces the string to be decimal.

Example:

```
Call SetOEMDRO (818, GetOEMDRO (818) * 1.1)       ' increase feedrate by 10%
```

Button Commands

```
Sub DoOEMButton (buttOEMCode as Integer)
```

Cal;l the routine to perform the action equivalent to "pressing" a button. Mach3 is instructed by the script to perform the function specified.

There is no provision for the trapping or reporting of errors but as most functions have an LED associated with them this can be inspected by the script code to check that the required action has been performed.

Very many "buttons" are toggles or cycle through a range of possible states or values. A loop containing inspection of an associated LED can be used to set a particular state. This example would be particularly suitable to be attached to a button.

Example:

```
Rem This sets the MPG jog on and the wheel to jog the Y axis
Rem There are actually more direct ways to do this in late releases
Rem of Mach2
JogTogButton = 174
JogMPGEn = 175
MPGJogOnLED = 57
MGPJogsY = 60
OK = False
For I = 1 to 2
          If Not GetOEMLED (MPGJogOnLED) Then
                    Call DoOEMButton (JogMPGEn)  ' try to enable
          Else
                    OK = True ' MPG is enabled
                    Exit For
          End If
Next I
Rem Could test of OK true here
OK = False
For I = 1 to 6      ' must get there after six axis tries
          If Not GetOEMLED (MPGJogsY) Then
                    Call DoOEMButton (JogTogButton) ' try next one
          Else
                    OK = True  ' got right axis selected
                    Exit For
          End If
Next I
Rem Could test OK here as well
```

Interrogating Mach's state

Interrogating Mach3 running modes

```
Function IsDiameter() as Integer
```

Returns a non-zero value if Mach3 is n Diameter mode (Turn only) otherwise zero for radius mode. Diameter mode affects th handling of all X coordinate values.

Interrogating Mach3 internal variables

The current value of Mach3 internal variables can be read using the `GetParam` function.

```
Function GetParam (name as String) as Double
```

This returns a numeric value corresponding to the name of the given variable which is provided as a string (constant or variable)

The corresponding routine `SetParam` sets the value of the variable to newVal.

```
Sub SetParam (name as String, newVal as Double)
```

A list of recognised strings can be found at Get/SetParam() Vars

Examples:

```
Rem interrogate drive arrangements
mechProp1 = GetParam ("StepsPerAxisX")
Rem make C acceleration be same as X for slaving
Call SetParam("AccelerationC", GetParam ("AccelerationX"))
```

Notice that the word "Param" is used here in a different sense to the Machine Parameters accessed by the # operator from within a part program and in accessing the Q, R & S word "parameters" to a macro call.

Access to the machine G-code parameter block

Mach3 has a block of variables which can be used in part programs. They are identified by # followed by a number (the parameter address). The contents of the Tool and Fixture tables are in these parameters but there are many values that can be used by the writer of a part program. These machine variables can be accessed within macros by GetVar and SetVar.

```
Function GetVar (PVarNumber as Integer) as Double
Sub SetVar (PVarNumber as Integer, newVal as Double)
```

The predefined parameter variables are defined the manuals *Using Mach3Mill* and *Using Mach3Turn*.

Examples:

```
FixNumb = GetVar (5220)       ' get current fixture number
Rem set X offset of fixture 2 to be same as fixture 1
Call SetVar (5241, GetVar (5221))
Rem increment a counter, say in a multiple part layout
Call SetVar (200, GetVar (200) + 1))
```

Arguments of macro call

When a macro is called from the MDI line or within a part program then data can be passed to it by P, Q, and S words on the line. The values of these words are "read" in the macro using the Param functions.

```
Function Param1 () as Double ' gets P word
Function Param2 () as Double ' gets Q word
Function Param3 () as Double ' gets S word
```

NOTE: There was a Bug in the S parameter that would cause the spindle to run at the S words number.

So as of Mach3 version: 3.042.011 that has been changed/Fixed The NEW Param3 is "R".

```
Function Param3 () as Double ' gets R word
```

Information to and from the user

Scripts can communicate with the operator by displaying a dialog box with a prompt into which the user can type numeric data. The Question function prompts for one item. The GetCoord routine prompts for the values of X, Y, Z and A coordinates.

Refer here to MsgBox etc built-in VB Script calls !!!

The other strategy, probably more suited to scripts attached to buttons, is to provide DROs of a screen into which data is set before running the macro. These can of course also display results from the script.

User Intelligent Labels and Tickers enable messages to be displayed.

Dialog boxes

```
Function Question (prompt as String) as Double
```

The string in `prompt` is displayed in a modal dialog titled "Answer this. The dialog contains an edit box. The value of the function is set to the number in this when OK is clicked.

```
Sub GetCoord (prompt as String)
```

As with Question, a modal dialog titled "Enter Coordinates" displays prompt. This has four edit boxes labelled X, Y, Z and A into which values can be typed. GetCoord itself does not return the values to the macro code. These must be fetched by GetXCoor, GetYCoor etc.

```
Function GetXCoor () as Double
Function GetYCoor () as Double
Function GetZCoor () as Double
Function GetACoor () as Double
```

Outputting text, warnings etc.

```
Sub Message (text as String)
```

Writes the message on the `Error` intelligent label and in the History log file.

```
Sub PlayWave (pathname as String)
```

Plays a Windows .WAV file (e.g. a chime to warn of an event or error). This feature must be enabled in Config>Logic

```
Sub Speak (text as String)
```

Speaks a text string. Requires Speech to be enabled in Config>Logic and a suitable sppech engine to be installed (e.g. the one supplied with Microsoft Office).

User defined DROs and LEDs

This technique is mainly applicable to wizards and scripts which are run from a user defined screen button.

A block of DRO OEM codes is allocated to 255 USER DROs (Ranged from 1000 through 1254) which are not used by Mach3 itself. These DROs, suitably labelled, can be placed on a screen.

The operator enters data into the DRO(s) before pressing a button or series of buttons to run the macro or macros. The macro(s) access the data using `GetUserDRO` as explained above. The macro can also use `SetUserDRO` to update the data or return a result in another DRO.

In addition there are 255 user LEDs which can be read and (unlike normal LEDs) written using `GetUserLED` and `SetUserLED`.

This technique can, for example, be used to implement a totally personal scheme to extend the Mach3 offset setting by Touch with Correction. Suppose you have a probe with a 5 mm tip diameter which only trips in sideways movement (i.e. for X and Y)