Mach specific Subroutines/Functions grouped by purpose

From MachCustomizeWiki

This section describes the most important standard routines (i.e. functions and subroutines) that can be called from VB Script on buttons or in macros.

This version is for Mach 1.90.033 and later

Contents

To execute G or M-codes from a script

The simplest and most powerful routine is:

```
Sub Code (text as String)
```

The *text* argument is a string expression (including, or course a constant or simple variable) which is any line of G or M codes that you could enter into MDI. It will be passed to Mach3 for execution. The only restriction is that you are advised not to call another script from within a script.

Examples, using both versions of the syntax for subroutine calls:

```
Code "G0X0" ' X to zero in current coords
Code "G1X10" & Feed ' variable Feed has been set to something like
              ' "F150"
```

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 in 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) then you might use a 1 mm slip or piece of shim-stock to manually feel the Z touch. You could define a pair of scripts attached to two buttons to apply the fixed, 5 mm, X and Y correction and a third button that uses a Z-correction DRO to set the thickness of the shim or slip which is in use.

Such features can be made to appear to the operator to be exactly like built-in Mach3 functionality.

User Button captions, Labels and Tickers

```
Sub SetButtonText (text as String)
```

This will change the caption text of the button to which the VB script is attached to the given string. This may only be called "from" a button rather than in a macro.

If the current screen has a label whose, case sensitive, text is in the range `UserLabel1` to UserLabel255 then the actual text displayed can be set in a Script by calling

```
Sub SetUserLabel (number as Integer, text as String)
```

This will display the given `text` in the label corresponding to the `number` given.

```
Sub SetTicker (number as Int, text as String)
```

Accesses the 255 tickers `Ticker1` to `Ticker255`. In a ticker the text of the message scrolls through the box so a very long message can be given in a small area of screen at the expense of some inconvenience for the user.

e.g.

```
SetUserLabel 12, "You must enter a whole number of holes"
SetTicker 205, "This is a very long error message because you seem to have done something very silly"
```

Handling files of Part Programs

This group of functions deals with loading and running G-code and features for the Teach MDI and wizard systems

```
Sub Loadfile (pathname as String)
```

Loads the given file of G-code.

```
Sub LoadRun (pathname as String)
```

Loads the given file of G-code and starts its execution.

```
Function AppendTeachFile (pathname as String) as Integer
Function OpenTeachFile (pathname as String) as Integer
Function OpenSubroutineFile (name as String) as Integer
```

Opens the given file for G-code and starts writing of commands executed (e.g. by MDI) to it. *Open* is an empty file, *Append* assumes that the file already exists and preserves its content. Returns non-zero value if operation is sucessful.

??? OpenSubroutine file is describes in Release Notes as exactly the same as OpenTeachFile. It is not exactly clear whsat the significance of this is.

```
Sub LoadTeachFile ()
```

Loads the G-code of the currently open teach file so it can be executed

```
Sub CloseTeachFile ()
```

Closes the currently open Teach or wizard file and stops commands being written to it.

Screen handling routines for wizards etc.

```
Sub ToggleScreens()
```

Switches between displaying the .SET and .SSET screen sets. This is employed on the standard screens to switch between the "complex" and "simple" screen sets but could be used for any purpose such a screens with and without a fourth axis or screens optimised for daytime and nighttime working.

```
Function GetPage () as Integer
```

Returns the number of the screen in the set presently being displayed. Used to remember where the user is when running a wizard.

```
Sub SetPage (page as Integer)
```

Used to display a given screen of a set, typically on return from a wizard. Equivalent to using `DoOEMButton` with the screen number.

```
Sub LoadWizard ( name as String)
```

Loads and runs the named Wizard.

```
Sub Savewizard ()
```

Saves the information in the local controls on a wizard screen in the wizardname.SET.DEFS file so that the values are on the screen when the wizard is next run.

Input/Output signals, Modbus I/O, a serial port and "foreign" ports

Signals

Scripts can access the input signals (both on parallel ports and defined virtually in response to keycodes) such as the state of home and limit switches and can control output signals.

```
Function IsActive (sigNo as Integer) as Boolean
Function IsOutputActive (sigNo as Integer) as Boolean
Sub ActivateSignal (sigNo as Integer)
Sub DeActivateSignal (sigNo as Integer)
```

`IsActive` tests input signals. `IsOutputActive` tests output signals. They will return True if the signal is active (i.e. its LED would be lit on the Diagnostics screen). In other words this test is after the application of the Active Hi/Active Lo configuration of the signal hot a test of "0 volts" or "5 volts" on the signal's pin.

`ActivateSignal` and `DeActivateSignal` similarly control the logical state of output pins. Mach3 will apply the Active Hi/Active Lo configuration to establish the electrical state required.

```
Function IsSuchSignal (sigNo as Integer) as Boolean
```

Returns TRUE if the signal is enabled. It is used to avoid things like digitising if the machine has no probe input defined.

For all these routines, the required signal is coded using the values defined on the VB Constants for Signal Names page.

```
Sub SetTriggerMacro (MacroNo as Integer)
```

Defines the number of a macro to be executed when an OEMTrigger is set (slightly unexpectedly on the Config>Set System Hotkeys dialog) to generate OEM code 301. This provides script execution without the requirement for a screen button as intermediary.

For example if `SetTriggerMacro 456` has been executed then a signal on any OEMTrigger configured to 301 will run the code in the file M456.M1S when activated.

Modbus

Modbus is a standard (details here (http://www.modbus.org/tech.php)) serial protocol which allows control of a device such as a Programmable Logic Controller (PLC) or dedicated interface (like Peter Homann's ModIO). Typically thois expands the input/output capability of Mach systems for non-time-critical functions.

Mach supports direct access to the registers of a Modbus device and will perform autopolling to map registers into ports and pins and to give simple access to the state of analog inputs and MPG counters.

Direct input routines

```
Function FillFromCoil (slave as Integer, startAddress as Integer, nBytes as Integer) as Integer
Function FillFromStatus (slave as Integer, startAddress as Integer, nBytes as Integer) as Integer
Function FillFromHolding (slave as Integer, startAddress as Integer, nBytes as Integer) as Integer
Function FillFromInput(slave as Integer, startAddress as Integer, nBytes as Integer) as Integer
```

These routines will request a memory transfer from the current ModBus device's Coil, Status, Holding and Input registers. The requested number of bytes starting from the address specified will be transfered to an internal buffer in Mach. Routine returns a 1 if successful in its transfer.

```
Function GetModWord(index as Integer) as Integer
```

This rotuine will return a "16 bit word". All data transfered from the device using the Fill routines, sends its information to a buffer. This routine will return a 16 bit word in the buffer selected by `index`. The first word is selected by `index` = 0

```
Function ModGetInputBit(nBit as Integer) as Integer
```

This routine will return the bit chosen by the index `nBit` as a one or a zero from the input buffer above. The first bit (i.e. most significant in the first word) is selected by `nBit` = 0

Example:

```
FillFromInput( 1, 102, 5 ) ' this will transfer 5 bytes to the buffer system from address 102 in the device.
i = ModGetInputBit (6) ' will get the 6th bit from that 5 byte array which you transfered
```

Polled input/output

```
Function GetInput (buffIndex as Integer) as Integer
Sub SetModOutput (buffIndex as Integer, value as Integer)
```

Reads or writes a 16 bit word from/to the buffers updated by polling the Modbus. Access to these buffers is a memory access rather than an I/O transaction implied by the raw reading routines so these calls can be made as frequently as desired.

`buffIndex` values 0 through 63 are single bits corresponding to the input and output pins of Port 0. The mapping of these to Modbus device registers is defined in the `Config>Setup Modbus Control` dialog.

`buffIndex` values 64 through 127 are 16 bit words. These are mapped in `Config>Setup Modbus Control` dialog. For full details see Modbus in Mach.

```
Sub SetModIOString (slave as Integer, xDisp as Integer, yDisp as Integer, Text as String)
```

Writes the string on the ModIO LCD screen (if present).

Notes:

(a) xDisp must be an even number, i.e. writing must start at a 16 bit word boundary in the ModIO.

(b) `SetHomannString (xDisp as Integer, yDisp as Integer, Text as String)` is a legacy routine equivalent to `SetModIOString`, with `Slave` = 1

```
 Function: GetInBit(buffIndex, Bit Number) as Integer
```

```
 Function: SetOutBit(buffIndex, Bit Number) as Integer
```

```
 Function: ResetOutBit(buffIndex, Bit Number) as Integer
```

GetInBit(BuffIndex, BitNumber) to use this one, Index number is 0-63 on channel one, and 64-127 on Channel 2, and BitNumber is 0-15 relating to the individual bit status in that word that you want, i.e. 0 or 1. See the Input section of config autopoller modbus. NOTE If you are using Bit packing special control then you can only Use the GetInBit on the 2cd channel 64-127. You have to use something like "IsActive(INPUT1)" on channel 1. If you don't use bit packing then you can use it with the 1st channel.

SetOutBit(BuffIndex, BitNumber) to use this one, Index number is 0-63 on channel one, and 64-127 on Channel 2. and BitNumber is 0-15 relating to the individual bit status in that word that you want, **It Turns the bit ON!!**. See the Output section of config autopoller modbus. NOTE If you are using Bit packing special control then you can only Use the SetOutBit on the 2cd channel 64-127. You have to use something like "ActivateSignal(OUTPUT1)" on channel 1. If you don't use bit packing then you can use it with the 1st channel.

ReSetOutBit(BuffIndex, BitNumber) is just like **SetOutBit(BuffIndex, BitNumber)** above, but it **turns the requested bit OFF.**

Serial port

You can send bytes of raw data to a serial port. The port number (i.e. n in COMn) to be used and the baud rate for transmission is set in Config>Logic. RTS/CTS hardware flow control protocol will be used to control large volumes of data but this will not be normally required. Data is transmitted 8 data bits, 1 stop bit No Parity by a call of SendSerial.

```
Sub SendSerial (chars as String)
```

Example: to write the value of X DRO to an LCD display connected to the serial (RS232) port.

```
Call SendSerial ("X-Axis = " & GetDRO (0))
```

Foreign ports

Scripts can access ports on the PC which are additional to the one (or perhaps two) parallel port(s) defined in Config>Ports and Pins. These are accessed at the basic hardware port address level and you will have to be aware of the details of the individual port addresses, allocation of data and status bits etc.

```
Function GetPortByte (pAddr as Integer) as Byte
Sub PutPortByte (pAddr as Integer, bData as Byte)
```

This feature should be used with great care as, if misused, it can interfere with any peripheral on your system, including the hard-drive.

Waiting and system features

As described above the script code and Mach3 itself run in two separate processes. You can test to see in Mach3 is busy or idle by calling:

```
Function IsMoving () as Boolean
```

This will return True if Mach3 is busy. You should call it in a loop after commanding an axis move or other function which could take a significant time and before reading DROs or LEDs that could be affected by the move.

Example:

```
Call Code ("G0X12Z100")
While IsMoving ()
WEnd
x = GetOEMDRO (802)  ' get Z value in case it has been Z inhibited
```

```
Function IsSafeZ () as Boolean
```

This will return True if Mach3 is setup to allow SafeZ moves. This is done under Config->Safe_Z.

```
Function IsStopped () as Boolean
```

Returns True if Mach3 is not in the process of executing a part-program. ??? Can anyone clarify what happens when in Feedhold, SingleStep and waiting for a Shuttle Jog pulse between blocks of code?

```
Function IsLoading () as Boolean
```

Returns true if the part program is loading rather than being actually run (e.g. so the toolpath is being generated or regenerated). This can be used to inhibit script actions like Question() in this situation.

```
Sub SystemWaitFor (sigNo as Integer)
```

Waits for the given signal to become active. This allows interfacing with physical controls on the machine.

```
Sub WaitForPoll ()
```

Waits until a Modbus Poll has taken place. Not sure if this is safe if Modbus is not turned on???. Use this in a loop, e.g. in macropump macro waiting for interesting things to come from the Modbus device.

```
Function IsFirst () as Boolean
```

Returns True if this is the first call of the function after Mach3 has exited from the EStop state. This can be used to re-initialse data that would be lost at a n EStop.

```
Sub LoadLinTable()
```

Reloads the table giving linearisation data for Mach3 spindle speed.

```
Sub SwapAxis(n,n)
```

This command swaps the step and direction pinout of two axes of your choice. X=0, Y=1, Z=2, A=3, B=4, C=5 and Spindle=6. For example: *SwapAxis(0,1)* will swap the X and Y axis. From here on any programed move or jog of the X-axis will actually move the Y axis. Please note that it is ONLY the pinout that is swapped! The effect is just as if you opend Config->Ports&Pins and changed the pinout there or swapped the wires at your break-out-board or drives. The motortuning values does not "follow" the axis. Be careful with this, if you exit Mach3 after issuing a SwapAxis() the swapped pinout will be saved in the .xml and will be in effect the next time you start Mach3. This command can ONLY BE USED ONCE - you are not allowed to first swap X with Y and then later Y with B for example. To reset the swapped axes you must use the command *ResetAxisSwap()*. After the swapped axes has been reset you are free to call SwapAxis() again.

```
Sub ResetAxisSwap()
```

This command resets a previous AxisSwap() call. It is the correct way to reset any swaped axes. Calling SwapAxis() a second time without calling ResetAxisSwap() is not allowed.

Legacy and special system VB Script calls

The following functions are still available to writers of macro scripts at Release 1 of Mach3. Their general use is, however, deprecated as better and more general ways are available (usually by accessing a DRO or LED with the Get/Set routines) or they are aimed at internal systems use. They may be withdrawn or changed in subsequent revisions of the Mach software.

```
Sub CloseDigFile()
```

Close the digitize point file.

```
Function CommandedFeed() as Double
```

This will return the currently applicable feedrate (including any override.

```
Sub DisablePWM ()
```

Inhibit output of PWM spindle signal for Digispeed.

```
Sub DisableSignal (signal as Integer)
```

Disables operation of given signal.

```
Sub DoSpinCCW ()
```

Starts the spindle in a counterclockwise direction.

```
Sub DoSpinCW ()
```

Starts the spindle in a clockwise direction.

```
Sub DoSpinStop ()
```

Stops the spindle.

```
Sub EnablePWM ()
```

Enable output of PWM spindle signal for Digispeed.

```
Sub EnableSignal (signal as Integer)
```

Enables operation of given signal.

```
Function FeedRate() as Double
```

??? Anyone able to clarify which rate this is (blended, commanded, actual etc.)

```
Function GetABSPosition(axis as Byte) as Double
```

This will return the absolute machine coordinate of the given axis.

```
Function GetCurrentTool() as Integer
```

Returns the number of the currently active tool (i.e. after G43 executed)

```
Function GetIJMode() as Integer
```

Returns 0 for Absolute mode, 1 for Incremental mode.

```
Function GetRPM() as Double
```

This will return the actual speed of the spindle as measured by the Index sensor (if fitted).

```
Function GetSafeZ() as Double
```

This will return the current Safe_z in Machine coordinates to the VB routine.

```
Function GetScale(Axis as Integer) as Double
```

Returns the scale factor for the given axis.

```
Function GetSelectedTool() as Byte
```

Will return tool selected but not yet activated.

```
Function GetToolChangeStart(Axis as Byte) as Double
```

Will return the position of an axis when a toolchange was called for.

```
Sub GotoSafeZ ()
```

Will move to the absolute Z coordinate specified in the Safe_Z DRO.

```
Function MaxX() as Double
Function MaxY() as Double
Function MinX() as Double
Function MinY() as Double
```

??? Anyone able to document these please

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

Writes the string, which should have only one line, in the Error intelligent label and history. Equivalent to executingbCode "MSG," & text

```
Sub OpenDigFile()
```

Open a digitize point cloud file. User is prompted for filename.

```
Function QueueDepth() as Byte
```

Depth of planner queue is returned.

```
Function Random() as Double
```

Returns a pseudo random number.

```
Sub RefCombination(Axes as Integer)
```

Performs simultaneous referencing on several axes. They are coded by ORing or addition of the following codes: X = 1, Y= 2, Z = 4, A = 8, B = 16 and C = 32.

```
Sub ResetTHC()
```

Resets the Torch Height Control code

```
Function RetractMode() as Integer
```

??? Anyone able to define this function please

```
Function Round() as Double
Function roun() as Double
```

??? Anyone able to detail these functions please

```
Sub RunFile ()
```

Executes the currently loaded G-code file.

```
Sub SetCurrentTool(Tool as Byte)
```

Will return currently selected tool

```
Sub SetFeedRate(Rate as Double)
```

Sets current FeedRate

```
Sub SetSpinSpeed(SWord as Double)
```

Sets current speed as by using the S word

```
Sub SetMachZero(Axis as Integer)
```

Defines the current position of the specified axis to be machine zero.

```
Sub SetIJAbs()
Sub SetIJInc()
```

These will set the IJ mode to absolute and incremental respectively

```
Sub SetIJMode(mode as Integer)
```

This will set the IJ mode to Absolute if mode = 0 and Incremental if mode = 1

```
Sub SetOutput (val as Integer)
```

??? Can anyone define function of this routine please

```
 Sub SetPulley( PulleyNo as integer)
```

Sets a new value for the current pulley. PulleyNo should be in range 1 through 4

```
Sub SetSafeZ(SafeZ as Double)
```

This will set the Safe_Z which should be in Machine coordinates

```
Sub SetScale(Axis as Integer, Scale as Double)
```

Sets the given scale factor for the given axis.

```
Sub SingleVerify(Axis as Integer)
```

Do a "silent" verification run on one axis not reporting the outcome, just correcting the axis position.

```
Sub SingleVerifyReport(Axis as Integer)
```

Do a normal verification run on one axis reporting any discrepancy.

```
Sub StraightFeed(x as Double, y as Double, z as Double ,
                    a as Double, b as Double, c as Double)
```

This will perform a feedrate move to X1,Y2,Z3…etc.

```
Sub StraightTraverse(x as Double, y as Double, z as Double ,
                              a as Double, b as Double, c as Double)
```

Performs a rapid move to x, y, z etc.

```
Sub THCOn()
```

Turn on THC control

```
Sub THCOff()
```

Turn off THC control.

```
Function ToolLengthOffset() as Double
```

Gets the tool offset length currently in effect if any.

```
Function tXStart() as Double
Function tZStart() as Double
Function tEndX() as Double
Function tEndZ() as Double
Function tFirstPass() as Double
Function InFeeds() as Double
Function ThreadDepth() as Double
Function GetMinPass() as Double
Function tMinDepth() as Double
Function tGetCutType() as Double
Function tCutDepth() as Double
```

```
Function tGetInfeedType() as Double
```

```
Function tClearX() as Double
Function tZClear() as Double
```

```
Function tLead() as Double
Function tSpring() as Byte
Function tPasses() as Byte
Function tChamfer() as Double
Function tTaper() as Double
Function tTapers() as Double
Function tInFeed() as Double
Function tDepthLastPass() as Double
```

```
Sub tSetCutType (type as Integer)
sub tSetInFeedType (type as Integer)
```

Gets/Sets parameters defined in a G76 threading cycle call for use by the canned cycle Script.

```
Sub VerifyAxis(Silent as Boolean)
```

Do a verification run. If silent is true, do not report the outcome, just correct the axis position.

```
Sub ZeroTHC ()
```

ZeroTHC clears the THC correction value so new correction may be done within THC's limits.

Retrieved from "http://www.machsupport.com/MachCustomizeWiki/index.php?title=Mach_specific_Subroutines/Functions_grouped_by_purpose"