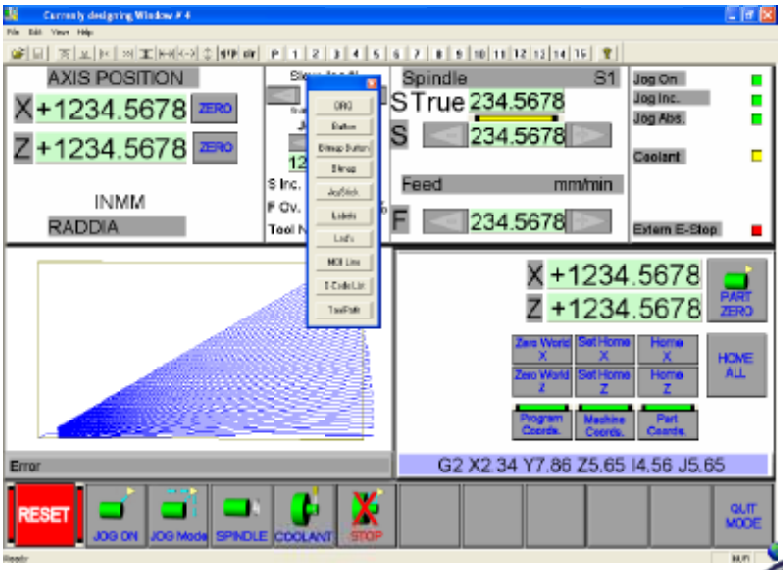
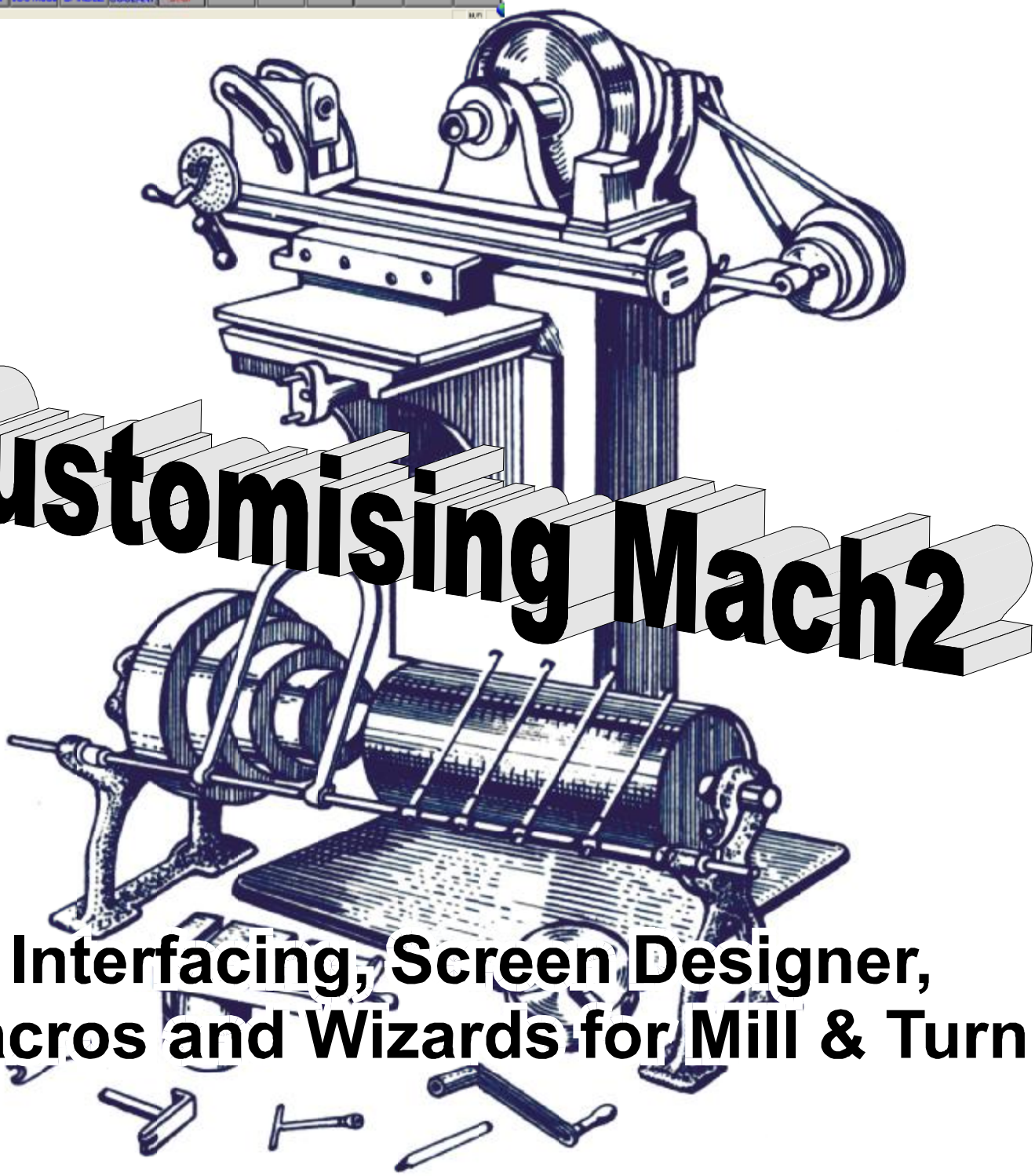


Mach2 ArtSoft



Customising Mach2

Interfacing, Screen Designer, Macros and Wizards for Mill & Turn

Mach2
Customisation Guide

All queries, comments and suggestions welcomed via support@artofcnc.ca

Mach Developers Network (MachDN) is currently hosted at:
http://groups.yahoo.com/group/mach1mach2cnc/files/

Front cover: Brown & Sharpe Universal mill 1862 (with some "artistic" liberties)
Back cover (if present): The old, gear, way of co-ordinating motion on mill table and a rotary axis

For Mach2 Release 6.11 **Manual Revision 6.11-A6**

Contents

1. Preface

Any machine tool is potentially dangerous. Computer controlled machines are potentially more dangerous than manual ones because, for example, a computer is quite prepared to rotate an 8" unbalanced cast iron four-jaw chuck at 3000 rpm, to plunge a panel-fielding router cutter deep into a piece of oak or to mill the clamps holding your work to the table!

This manual tries to give you guidance on safety precautions and techniques but because we do not know the details of your machine or local conditions we can accept no responsibility for the performance of any machine or any damage or injury caused by its use. It is your responsibility to ensure that you understand the implications of what you design and build and to comply with any legislation and codes of practice applicable to your country or state.

If you are in any doubt you must seek guidance from a professionally qualified expert rather than risk injury to yourself or to others.

This document is intended to give details about how to customise the Mach2 system. It assumes that you are familiar with the contents of *Using Mach2Mill* or *Using Mach2Turn* (still in preparation) as appropriate.

Some customisation, for example, changing keyboard shortcuts or removing unwanted controls from a screen, is very straightforward and can easily be achieved by a user familiar with typical Windows applications and the conventions for using them. Other features such as interfacing special devices like tool-changers and designing wizards for automating special tasks requires a knowledge of programming and/or hardware. This manual does not attempt to cover the basic skills but shows how to apply them when customising Mach2.

You are strongly advised to join the online discussion forum for Mach2. This is currently hosted by Yahoo! A link to join it is on the *Company* page at www.artofcnc.ca You should be aware that, while the forum has many engineers with a vast range of experience as participants, it does not constitute a substitute for a machine tool manufacturer's support network. If your application requires this level of support then you should buy the system from a local distributor or an OEM with a distributor network. In that way you will get the benefits of Mach2 with the possibility of on-site support.

Certain portions of text in this manual are printed "greyed out". They generally describe features found in machine controllers but which are not presently implemented in Mach2. The description of a greyed out feature here is not to be taken as a commitment to implement it at any given time in the future.

Thanks are due to numerous people including the original team who worked at National Institute for Standards and Testing (NIST) on the EMC project and the innumerable users of Mach2 without whose experience, materials and constructive comments this manual could not have been written. Particular thanks are due to Olivier Adler and Brian Barker for their contributions to the development and documentation of Wizards and to Les Newell for his KeyGrabber (which now grabs a lot more than key strokes!).

ArtSoft Corporation is dedicated to continual improvement of its products, so suggestions for enhancements, corrections and clarifications will be gratefully received.

Windows XP and Windows 2000 are registered trademarks of Microsoft Corporation. If other trademarks are used in this manual but not acknowledged please notify ArtSoft Corporation so this can be remedied in subsequent editions.

2. Communication routes

In this chapter we will look how the hardware of the machine tool and its user controls, the computer hardware and Mach2 communicate with each other.

You will probably find it best to skim this chapter and return to it as you look at the other individual customisations.

Before we can look at customisation in detail you will have to know how the various parts of your machine communicate with Mach2 and, indeed, how you communicate with it.

Figure 2.1 shows the important communication routes and with the following description should allow you to understand how they work. The diagram looks complex and this is a fair reflection of the fact that Mach2 is complex and can be configured to work with many types of machine. Your system will, almost certainly, not have all the features on it but, before you start customising, now is a good time to understand the options. Notice the key which explains what the different type of lines represent.

2.1 Electrical connections

If you have set up your own system you will have been through the process of connecting switches like the home switches to pins of the parallel port(s) and using the dialogs to tell Mach2 which pins are used for which function. This is shown at the bottom left of figure 2.1. Notice we have shown the limit and EStop switches directly controlling the drive electronics because this is the safest technique and recommended in this manual. Your switches might, of course, control Mach2 via the parallel port(s). This would change the arrows from the switches on the diagram to go into the port(s) instead of the motor drives box.

Mach2 makes extensive use of these signals during running. This is shown by the arrow into the "brain".

Some switches (currently 3) can be assigned to input pins as OEM Trigger signals. These can be configured to do what you wish by issuing an "OEM code" command directly to Mach2 - just like pressing a screen button does. They do not, however, need anything on a screen to do this. The OEM Button codes that are issued by each signal are defined using the Config>Set Axis Hotkeys dialog.

2.2 Keystroke connections

2.2.1 Keystrokes

A "keystroke" produces a scancode that is sent to your computer using the PS/2 or USB connectors. They can be generated in three different ways:

- By typing on the computer keyboard
- By closing/opening a switch connected to a keyboard emulator. Such devices may be connected via the Universal Serial Bus (USB) and/or be daisy-chained to the main keyboard. They were originally developed by the gaming community to allow computer games to put in and be controlled by arcade-like cabinets but are very convenient for machine control applications. It is also possible to connect an encoder (typically low resolution like those used as "digital potentiometers" in hi-fi and video equipment) and to generate a given scancode for each "click" in the clockwise direction and another scancode for each counterclockwise click.
- By pressing a button or key on a custom keyboard (perhaps like a numeric pad on a long lead, buttons on a joystick, or a "console"). These options are

really just some switches and an emulator in a standard box, probably with some neat way of labelling the switches. Windows refers to them as Human Interface Devices (HIDs) Figure 2.2 shows an example with a Mach2 function overlay sheet in place.

A keystroke when Mach2 is running can, of course, do many things: be part of a command entered in the MDI line, be part of a value entered into a DRO or be a *hotkey* shortcut. A real keyboard will be used for all these things but the other sources of keystrokes will generally only be used for shortcuts.

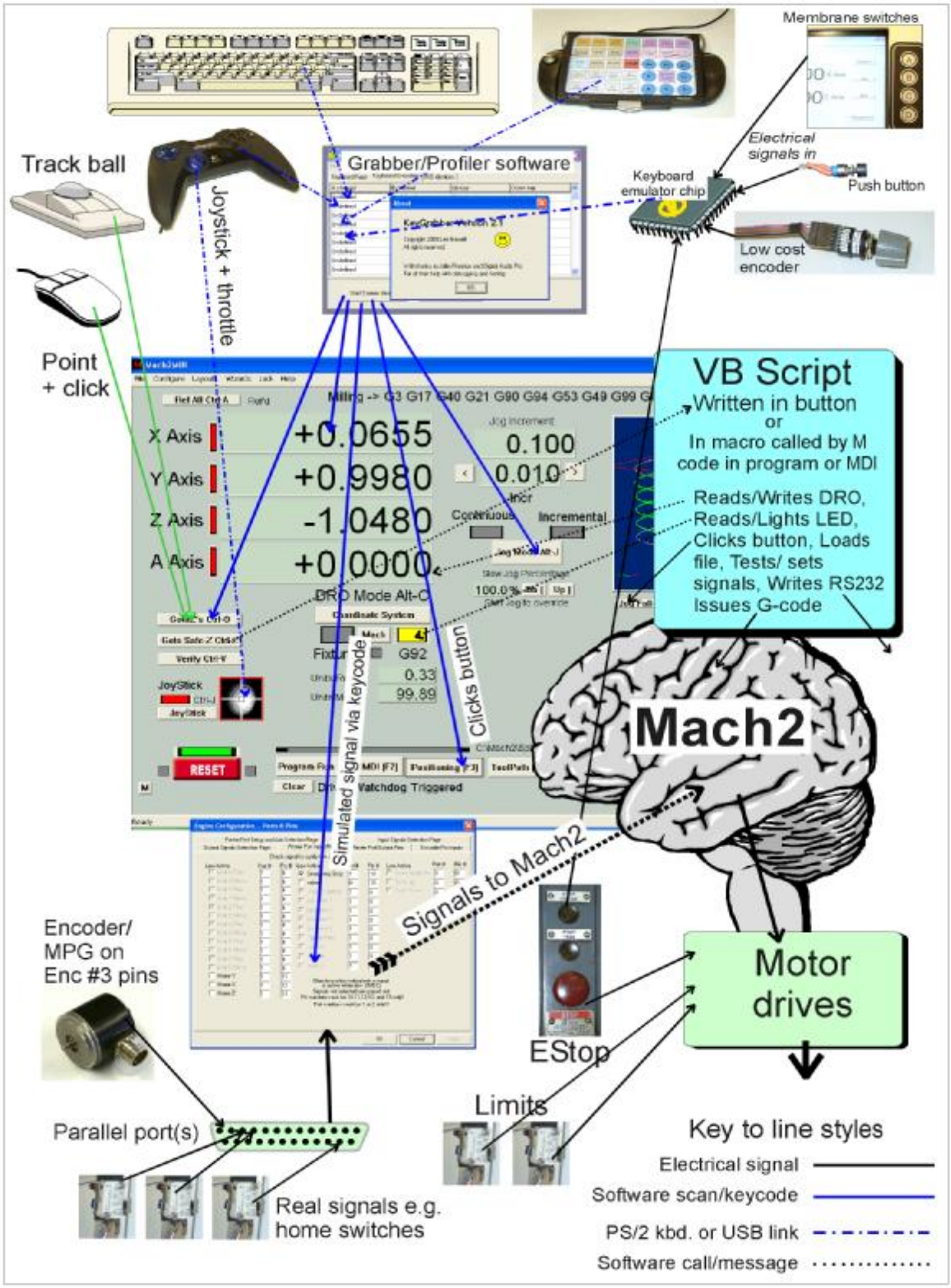


Figure 2.1 – Flow of control in a Mach2 system

2.2.2 Keystrokes and Shortcuts (Hotkeys)

A keystroke generates a Scancode which is processed by Windows and then Mach2. If it is not used directly as input to MDI or a DRO then it can perform **three** functions:

- if the code defined as a Button Shortcut (Hotkey) for a button currently displayed on screen then it is equivalent to clicking the mouse on that button
- if the code defined in the Config>Set Axis Hotkeys it will jog, load a file or select MDI or a DRO as appropriate

Figure 2.2 – A customisable keyboard

- if the code defined in the Config>Ports and Pins Input Pins dialog it will turn On (key down) or Off (key up) the signal to which it is attached. This, used with a keyboard emulator, gives a mechanism for extending the number of input pins far beyond those available on the two parallel ports. Not all signals can be made "virtual" like this and you are advised not to use the mechanism for safety critical operations.

You may consider using having real push buttons (or perhaps a membrane switch pad) beside the screen which can be labelled for different functions on each screen (making soft keys like in a bank ATM) or be used for a common function like screen changing. This is illustrated top right of figure 2.1.

A rotary encoder (MPG) connected by keyboard emulator can be used to send jog axis hotkeys to jog axes or to send the codes for hotkeys for buttons like Raise/Lower spindle speed or override feedrate.

This keystroke processing mechanism is shown by the blue lines on figure 2.1 from the "keyboard devices" to the screen buttons and the Input Pins dialog.

2.3 The KeyGrabber and profilers

As mentioned above, keystokes from the actual keyboard and keyboard emulators have their scancodes processed by Windows before they are given to Mach2. In particular, of course, Windows has to route the keycodes to the application whose window has the focus.

Custom keyboards, game pads etc. (HIDs) generally need a piece of software to define which keycodes each button is to generate. This software is often called a *profiler* and is supplied by the custom keyboard manufacturer. Profilers usually have a picture of the device and its buttons and when you click on these you are prompted to input the keycode and any <Shift>, <Ctrl> or <Alt> modifiers to be used with it.

A utility, KeyGrabber, has been developed, by Les Newell, for use in conjunction with Mach2 (and its Screen Designer) to make it easy to configure Keyboard Emulators and USB connected HIDs including custom buttons, joy sticks, throttle controls etc. It has the additional advantage that it will pass selected keycodes to Mach2 **whatever** program has the focus. This Mach2 can continue to be controlled even if you need to briefly run another application like Windows Explorer, the Calculator, indeed, you accidentally swap to a program by clicking in the task bar. Although at first sight translation of a few key codes sounds rather trivial, if you look into it we think you will discover how powerful a technique it is for making Mach2 easy to use with a variety of standard and therefore reasonably priced input devices.

Details of KeyGrabber and its installation and configuration are given in Appendix 3.

2.4 VB Script connections

A part program is written using G and M codes. It is, however, also possible to program Mach2 in another language called VB Script. This is a subset of the Microsoft Visual Basic language which is available in many applications besides Mach2, including Microsoft Office. While it would be possible to totally control a machine tool using Mach2 with a VB Script program, the facility is really intended to help customise the environment in which a G-code part program runs.

2.4.1 VB Script program

A VB Script program can be used in three ways:

- By being "attached" to a screen button and so be run when the button is clicked or its hotkey is pressed
- By being put in a macro and called from within a part program, the MDI line of the G-code attached to a button. The last of these methods being suitable if the program is too big to attach directly to the button or is required to be used as a macro from a part program as well as a button.
- By being a macro attached to an OEM input trigger by OEM code 177.

The VB Script program can:

- declare and use VB variables,
- execute conditional statements (If-Then-Else),
- perform loops, and
- call VB the functions and subroutines defined by the macro writer or provided by Mach2 as its interface to scripts.

The predefined interface functions and subroutines are defined in detail in chapter 4 below. The main ones allow the script to issue G-code commands, to read and write the values of DROs, to inspect LEDs (and for User LEDs, to set them), and to send Function and OEM codes to Mach2 thus simulating what a user does when s/he clicks a screen button. Additionally a macro can test the values of input signals, set output signals, access internal data such as motor tuning parameters and use parallel and serial ports not used by Mach2 itself.

2.4.2 Mach2 macro

As already mentioned, a macro is a piece of VB Script. Each macro has a name like M134. The M is used at the start of every macro name and the number can be any integral value up to 99999 that is not used to define a built-in M-code. These built-in numbers are listed in chapter 11. Thus for example, M12, M50, M16543 are all valid macro names; while M3, M-56, M0234, M567.4 are not valid names.

Standard macros will use the number range up to M999, Original Equipment Manufacturers (OEMs) are advised to use M1000 to M89999 and end-users can avoid naming conflicts by using M90000 to M99999.

Each macro is stored as text in its own file whose name is the macro name with file extension ".m1s". Thus one would find files: M12.m1s, M50.m1s and M16543.m1s on a system with the above macros in it. The macro files are collected in a folder called "Macros" within the Mach2 folder.

A macro is called just like an ordinary M-code command by:

- Execution of a line (block) of a part program containing its name
- Including its name on commands entered on the MDI line
- Including its name in the code to be executed on clicking a screen button.

All these features are shown by the arrows into and out of the Macro box in figure 2.1.

2.5 Windows' control

The final elements shown in figure 2.1 are the standard devices that control Windows. Common examples are the mouse, and trackball. Windows also supports a gaming joystick and throttle. This can be used for jogging and overriding jog and feed rates.

2.6 Other customisation

2.6.1 Global hotkeys

Config>Set Axis Hotkeys allows you to set up the keys which will jog the axes when the Jog Mode button is displayed on the current screen. Click the button for the axis and direction you want to set and then type the keystroke to be used. It is your responsibility to make sure they are unique and do not clash with other hotkeys you might want to use. See appendix 1 or use Screen Tweak to prepare a list of standard hotkeys.

These hotkeys can very usefully be generated by a profiler such as KeyGrabber to provide jogging support from Human Interface Devices or buttons and encoders attached to a keyboard emulator.

You can also define hotkeys which will select the MDI window, will select a DRO and will display the File>Load G-code dialog. These are useful if you are controlling Mach2 without a mouse or similar pointing device.

3. Screen Designer

His chapter describes the features of Mach2 Screen Designer. This program allows you to customise the released screens and to design your own, from scratch, both as main screens and for wizards

Mach2Mill comes with a standard set of screen layouts to suit many uses. You are strongly advised to use a screen resolution of 1024 x 768 pixels if you can, but screens can be provided for 800 x 600 and 640 x 480 pixels.

The Screen Designer lets you change the layout of any or all of the information displayed on screen by Mach2. You can, if you wish, design a complete set of custom screens but, as there are more than 800 individual objects on the standard screens, most users will probably only want to make detailed changes to the layout.

The screen to use is loaded into Mach2 using the View>Load Screen menu. Screen layouts are stored in files with the .SET extension or the .SSET extension. The Screen Editor application is used to edit layout files.

The individual screens within a layout can be moved, deleted, imported etc. using the ScreenTweak utility (q.v.) This is primarily concerned with manipulating whole screens while Screen Designer deals with individual controls, their functions and positioning.

3.1 Screen Designer basics

The Screen Designer allows you to create and edit a set of 15 screens which form the .SET file. A screen consists of a collection of *Controls*. The Controls supported are:

1) DRO (digital read-out) - to display and optionally enter numerical values
2) Button - to request an action
3) Bitmap button - action but with an image rather than text giving function
4) Bitmap – display an area of color or a picture
5) Joystick - to control manual jogging
6) Label - to identify other objects or be a placeholder for a text display
7) LED (light emitting diode) - On/Off or warning indicator
8) MDI (manual data input line) - to allow input of "G-code" line
9) G-code list - to display current part program text
10) Toolpath - to display path followed by tool in current part program.

The screens are numbered 1 to 15 and screen P which is used to place controls that will appear on every screen (P is for Persistent). This is used, for example, for the buttons used to change from screen to screen.

3.2 Try out the Designer

Close Mach2, if it is running, and use the shortcut you installed in the desktop to run the Screen Designer.

You should see a mainly blank screen with a menu bar, a Tool bar (top of screen), a Status bar (bottom of screen) and a Pallet of buttons. The pallet and toolbar can be dragged around the screen to a convenient place out of the way of controls that you are editing. The toolbar and status bar can be hidden and viewed using the View menu. You will only need to do this if you want to place controls very near or sometimes off the bottom of your finished screen.

Run Screen Designer and do not open a layout. You will get a blank screen. Click a button on the pallet and the click anywhere on the screen. You will get a default sized control of the type chosen. You need to press a pallet button and click for each control you want.

By default you will be putting controls on Screen #1. Use the numbered buttons on the toolbar to select other screens and place controls on them. Try placing some controls on the "P screen" and see that they actually appear on all screens 1 to 15.

If you click on a control you have placed you will see that it becomes selected and is drawn with the traditional sizing handles. It can be moved around the screen and resized by dragging the outline or the relevant handles. You can also move selected objects by "nudging" them with the left/right/up/down cursor keys. Each keypress moves the control by one pixel.

Multiple objects can be selected using Shift-click. One selected object in a set can be deselected by Control-click.

The selected objects can be cut or copied to a clipboard and the contents of the clipboard can be pasted onto the same or another of the fifteen screens. Notice, that although you can run more than one copy of Screen Designer at one time editing different .SET files, each has its own clipboard so you cannot move controls from one file to another using the clipboard.

When you resize a control, other than a button, with the handles then its contents are scaled to fit the new size. This is how you decide on the font size for DROs, labels etc.

When you have created a test screen, use Save As to save it with a new file name. Run Mach2 and load your screen using the View>Load Screens menu. You should never overwrite the default screens with your designs as these files are likely to be overwritten by the installer each time you upgrade Mach2 to a new version.

3.3 Making the controls work

It is now time to see how to see how controls "work". It is easiest to do this by looking at the standard Mach2 screens. Assuming you have a 1024 x 768 screen, save your trial screens (perhaps in file `Junk.set`) and open `C:Mach2/1024.set` (or perhaps safer make and use a copy of this)

You will be shown Screen #1 - the "Program Run" screen. See which screens are displayed for screens 2 to 7. **Do not save this work** especially if you are using the real screen layout!

Double-click the *Cycle Start* button on screen #1. You will be shown its properties which will look similar to figure 3.1.

3.3.1 Key scan codes

When Mach2 is running, Buttons and Bitmap buttons can of course be "pressed" by clicking the mouse. They can also have a "hotkey" assigned to them. Pressing that key on the keyboard when the button is visible is equivalent to clicking it with the mouse.

The "hotkey" is defined by its "Mach2 scancode". Screen Designer allows you to type the key you want to use and it will store and display (in decimal notation) the associated scancode.

If you need to check these codes at any time then you can calculate the value by hand as follows:

Lookup the ASCII value of the code for the key (For letters it is the uppercase version of the character for other keys it is the code of the unshifted character – i.e. 5 rather than %). Thus, for example, spacebar = 32, A = 65, / = 47 etc. If you do not have a printed table of these codes then entering "ASCII code table" into your favorite search engine should find one.

If your hotkey is to be a given key together with Shift, Ctrl or Alt depressed (or indeed a combination of these) then add, to the basic code, the following values (in decimal)

Shift	1,024
Ctrl	32,768
Alt	2,048

Thus Shift-A would be 1,089 (65 + 1,024), Alt-/ would be 2,095 (47 + 2,048) and Shift-Ctrl-Alt-spacebar would be 35,872 (32 + 1,024 + 32,768 + 2,048)

If you have worked with binary numbers then you will recognise that these rather odd looking numbers are actually powers of two.

3.3.2 Defining function by G-code or VB Script

This is the most direct way of making a new button work. You can provide some G-code or VB Script to be executed when it is clicked. You need to check the appropriate radio button to say what type of code you have written in the edit box. As an example look at figure 8.3 and the G50 button on the MDI screen of 1024.set. You will see that the latter (not unexpectedly!) issues the command "G50".

You can define buttons using G-code to do whatever you want that can be done by a single MDI line. The VB Script program, indeed almost always will, have multiple lines and can be up to 64k characters long!

Beware: If you use G-code like this, then any scaling which is active on axes at the time you click the button will be applied to the coordinates built into your button. The problem does not, of course, apply to zero values, which is a common case, whatever scaling is applied.

Hint: Remember that a button will only be operated by its hotkey if it is on the persistent or

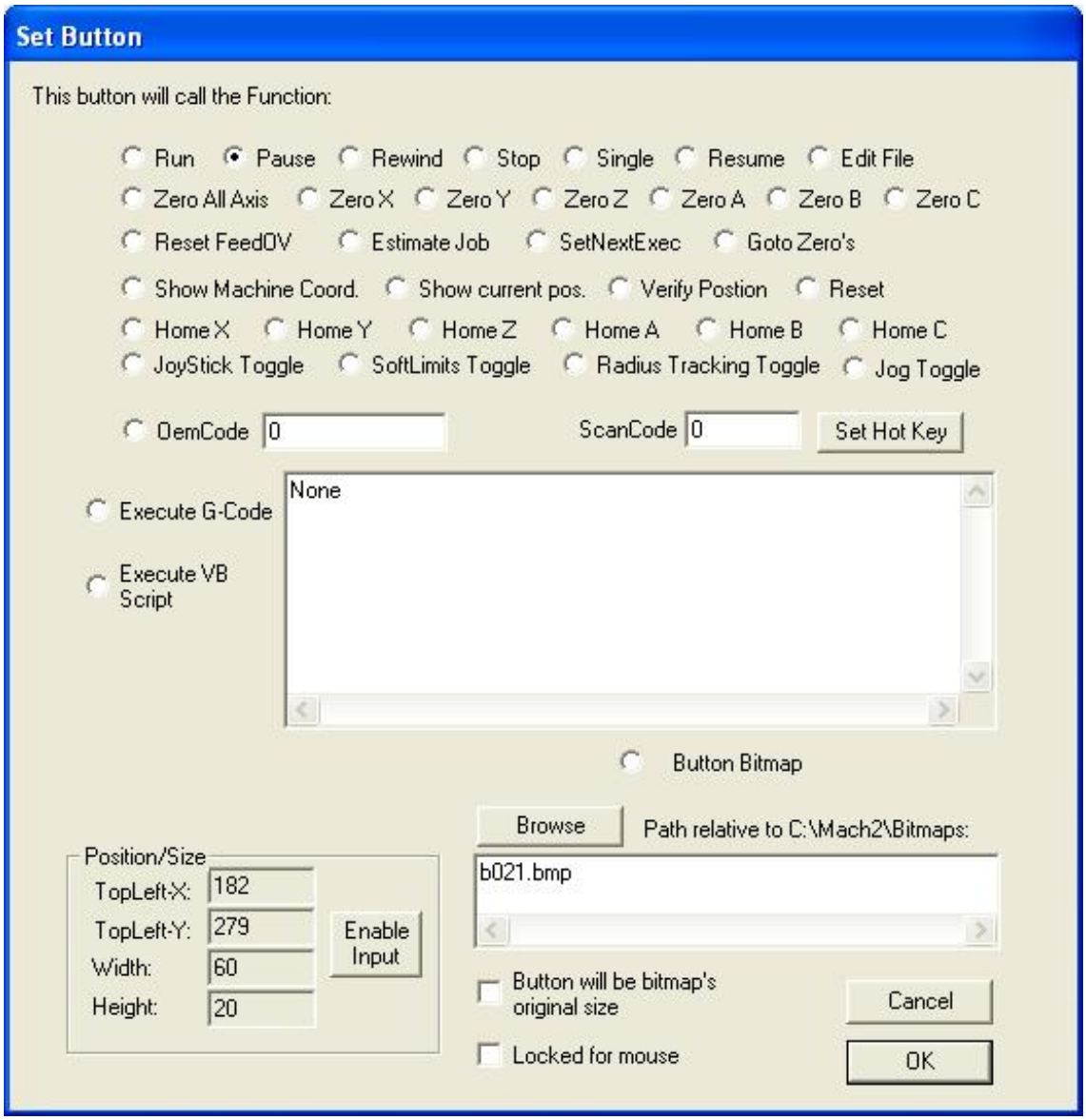


Figure 3.1 - Properties of standard *Cycle Start* button

currently displayed screen. A useful trick for VB Script or G-code buttons, that you want to work at all times, is to draw them on the persistent screen but off the bottom of the viewable area. You can "design" here by hiding the toolbar and status bar using the View menu or by designing on a higher resolution display than you are going to use with Mach2.

If you are only interested in adding hotkeys, G-code and VB Script then you probably need to skip to the section on "Properties of other controls" and then read no further about Screen Designer. Exit **not saving** any changes to the 1024.SET.

3.3.3 Defining function by name

Notice, after double clicking a button, that there is a long list of radio buttons for functions that a button can call. *Cycle Start* uses the one called "Run". One possibility is "OEM code" which we will look at later. The button has a title given in its "Button Text" box and can be activated by a "Hot key" (Alt-R in fact). The user is reminded of this in the title - some buttons are very small so the shortcut is given in a separate label. Mach2 identifies keys by their scancode. Alt-R is code 2162. If you click *Set Hot Key* and type any other key (or Shift/Ctrl/Alt key combination) this will set a new hot key for *Cycle Start*. You will see its scan code. For safety set it back to Alt-R.

The list of shortcuts used on the standard screens is given in Appendix 1.

Close the dialog by OK

3.3.4 Defining buttons by OEM code

Now look at the properties of the *Flood* button by double clicking it. You will notice that its function is defined as OEM code 113 in the list of radio buttons. OEM codes are a way of extending the range of buttons for more esoteric purposes without the list of names becoming much too long. The allocation of named functions and OEM codes to buttons on the standard screens is given online in Appendix 2.

If you look at the properties on the *MDI* button you will see that its OEM code is 2. Codes 1 to 15 are used to select the fifteen possible screens. If you look at the codes in all the screen selection buttons you will see which screen # is allocated to each and this will tie up with the rather odd order you will have discovered at the beginning of this section.

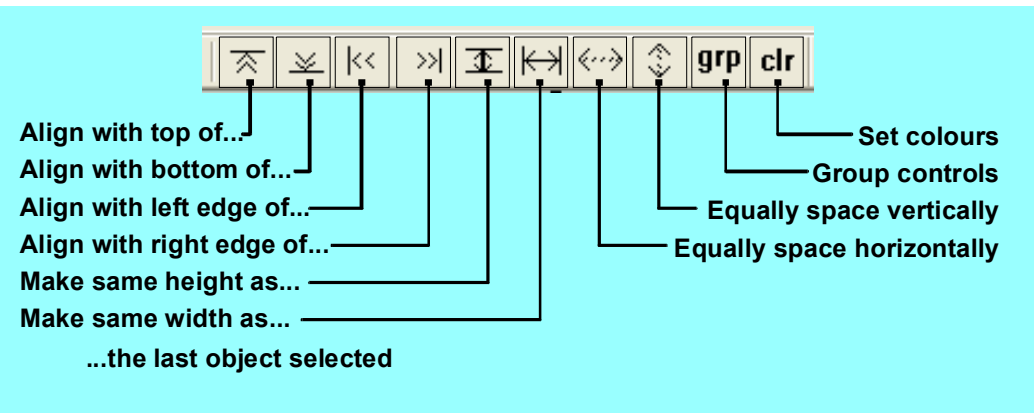


Figure 3.2 - Screen Designer alignment tools

3.4 Getting a tidy visual effect

The Screen Editor has several features to ensure a neat looking screen design. These are accessed from the icons at the right hand end of the toolbar. (Figure 3.2)

With the exception of the "Clr" - for Color - button they all act on a multiple selection of controls. These will often be of the same type (e.g. six DROs for the six axes) but do not need to be.

The **last** control selected in the multiple selection is special as it is the *master control* of the selection and other controls will be moved, sized etc. in relation to it.

3.4.1 Alignment icons

The first four icons on the bar align the appropriate edge of selected controls with that same edge of the master control. For example "Align with left edge" would move all the controls selected horizontally so all the left hand sides were in a vertical line with the left edge of the master one. Figure 3.3 shows the DROs just after drawing. They are then selected with the top one chosen last and Align Left Edge clicked. Figure 3.4 shows the result.

You need to be slightly careful as if you align left and, say, top then all the controls will be on top of each other. If this happens by accident then you need to deselect them by clicking on a blank bit of screen, select them one by one and drag them apart.

3.4.2 Sizing icons

In the example the controls are different sizes. This would not happen if you had used the clipboard to duplicate them but it is easy to make them all the same width and or height as the master of the selection. Figures 3.5 and 3.6 show this done in two stages with the two "Make same.." icons.

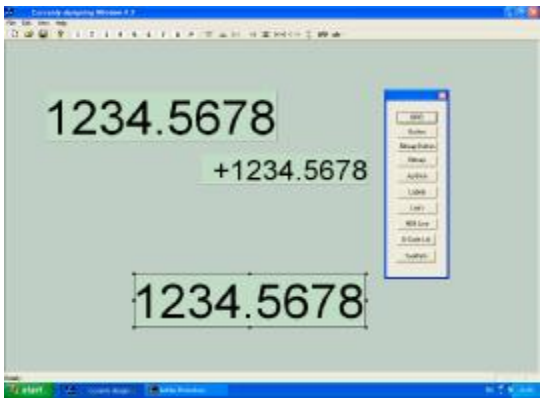


Figure 3.3 - Ragged DROs as drawn

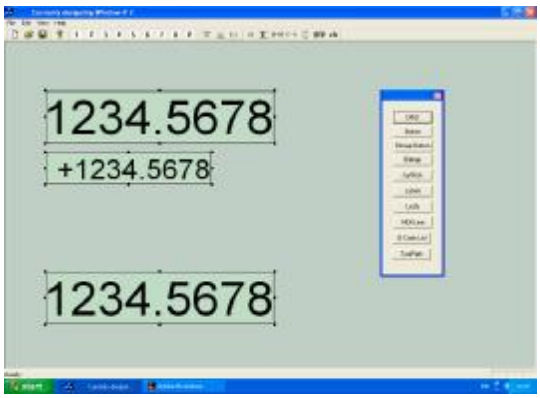


Figure 3.4- Justified DROs

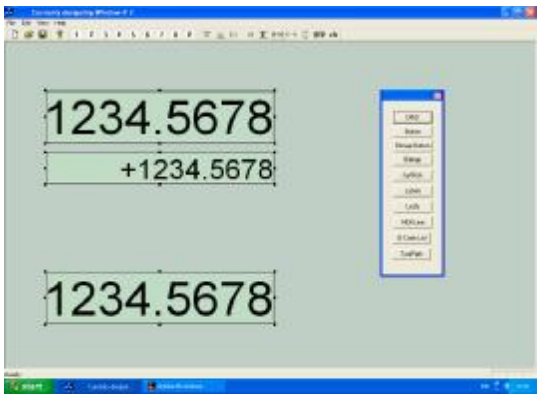


Figure 3.5 - Same width DROs

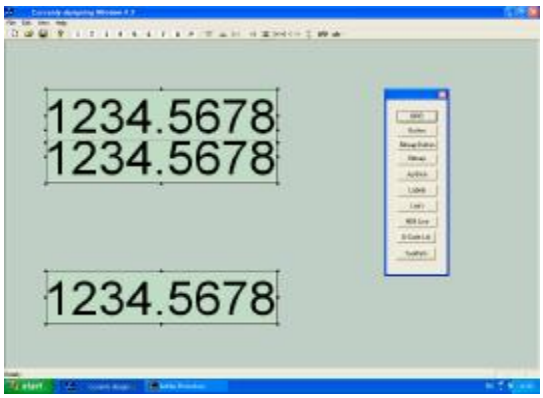


Figure 3.6 - Same size DROs

The size of a control, other than a button, will determine the size of any text in it.

3.4.3 Spacing controls uniformly

Finally the "Equally space …" icons can be used to space the selected controls equally. The result of this is shown in figure 3.7.

It is worth spending a bit of time experimenting with these icons on different types of control to get a feel for what can be done to make a neat layout with very few clicks of the mouse.

You should get into the habit of saving your layout frequently as the current version of the Screen Designer has no Edit>Undo facility.

You are strongly advised to save any changes in a layout with a name of your own choosing. If you just change 1024.set then, as this is replaced at each Mach2 upgrade, you will lose your changes.

3.5 Properties of other types of control

Now work your way through the different types of control and by inspecting the properties of examples of each type you will be able to see how Mach2 standard screens are built.

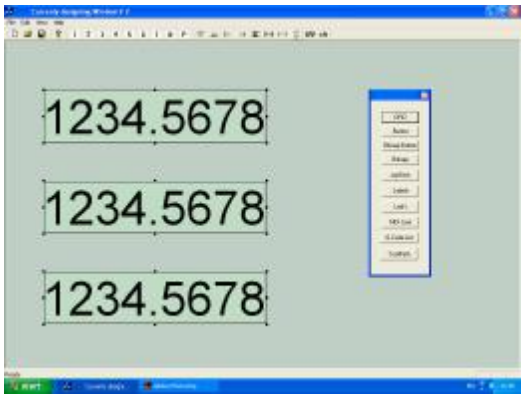


Figure 3.7 - Equally spaced controls

3.5.1 User LEDs and DROs

User defined LEDs and DROs are setup with values 1000 to 1254 in the OEM code field.

You can, for example, set the number of lines to be displayed in the G-code window and that, together with the size of the window, will define the text size.

Notice that DROs have a hotkey associated with them. When this key is pressed when Mach2 is running the appropriate DRO will be selected for data input. This is very useful in systems that are run without a mouse.

Some controls are very simple (e.g. the joystick ball) while others are complex (LEDs are different colors, can flash etc.).

3.5.2 Properties of Intelligent Labels

If a label has some reserved text in it then this text is replaced when Mach2 runs by information about what is happening (e.g. the name of the part program which is loaded).

The following intelligent labels are defined in the current version:

3.5.2.1 System labels

File (the part program), **Error** (the last Mach2 error message), **Mode** (the current modes of the system), **Profile** (the name of the current profile file)

3.5.2.2 User Labels/Tickers

UserLabel1, **UserLabel10**, up to **UserLabel255**

Ticker1 up to **Ticker255**

These display the text set be VB Script calls to SetTicker and SetUserLabel. The difference is that a Ticker scrolls the text so that a small control can display a long message.

Note: The intelligent label name text is case sensitive. For example "FILE" displays the word FILE but "File" displays the current part program's file name.

Figure 3.8 - Bitmap buttons

3.5.3 DRO groups

Mach2 is designed so it can be operated without a mouse or other pointing device although if one is provided it will probably be best to use it.

DROs can be selected by a hot key and the cursor keys can be used to move around them. To make this as convenient as possible they are combined into groups. Left and right keys go from group to group and up and down keys move within a group. The ***Grp*** icon in Screen Designer allows you to allocate a group number to a multiply selected group of DROs.

3.5.4 Use of Bitmaps

The location of the file containing a Windows bitmap is given in the setup dialog for bitmap buttons and for background bitmaps. All files must be in the Mach2 Bitmaps folder or in folders below it in the tree. It is convenient to group all the bitmaps for a custom layout in a folder within Mach2\Bitmaps to make them easy to maintain and to distribute to other users.

Screen Designer will always store the path relative to Mach2\Bitmaps in the layout file and it is this path that is displayed in the dialog after you have used Browse to locate a bitmap file. This can be seen in figure 3.1.

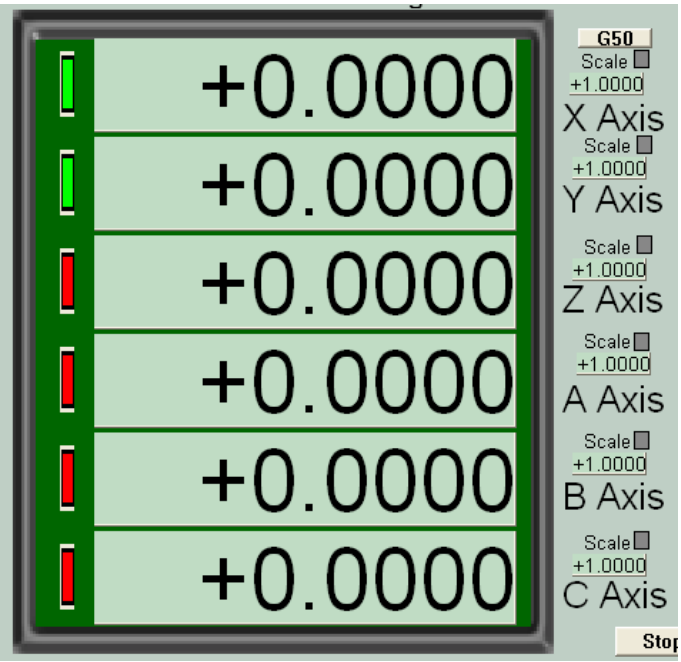


Figure 3.9 - Bitmap frame for DROs

3.5.4.1 Bitmap buttons

A bitmap button can be used in place of any normal button to give emphasis to the function of the button. For an example see figure 3.8

Bitmap buttons when created are empty and transparent. In Screen Designer a black border is drawn but such a button will be invisible on the screen when Mach2 is running. Such a button can be though of as a "hotspot" on the screen with its function being indicated on a background bitmap below it. An example is given in the next section.

3.5.4.2 Visual grouping with bitmaps

A set of related DROs buttons LEDs etc. can be placed "in" a frame or bezel by placing a suitable bitmap "underneath". A example is shown in figure 3.9. You should complete the placing of the controls before drawing the bitmap as it is more difficult to select the handles to resize the controls when the bitmap is present.

Screen designer provides two options on the Edit menu to assist with bitmaps as backgrounds. Edit>Paste Exact pastes an object in exactly the same place on a screen as the place from which it was Cut or Copied. If you wish to change the size or position of controls on a bitmap then you can cut it and paste it temporarily on an unused screen. Now move the controls and when finished you can Cut from the temporary screen and Paste Exact to restore the position of the bitmap.

If you want to make big changes to a screen then you might cut and paste all the bitmaps to the temporary screen at one time.

After re-pasting bitmaps back to their proper place they will of course be on top of the layers of objects. Edit>Bitmaps to Back allows you to put all the bitmaps on the current screen behind the other controls.

You can use one bitmap file for many backgrounds or bezels. A basic size of 100 pixels square works well for coloured backgrounds.

Stretching a square bezel to become very long and thin will distort the width of the border so you might need to have more than one basic shape. Try to minimise the different styles and colour schemes or your screens will look cluttered and be more difficult to use.

3.5.4.3 Identifying controls by the background bitmap

Although the controls in Screen Designer are quite flexible, you can achieve an even more interesting presentation by putting all the graphic detail in a large background bitmap.

Figure 3.10 illustrates a prototype of such a screen (courtesy of Ken Bell). In such a design the DROs, labels, LEDs etc. are real Mach2 controls. The buttons are in the bitmap but have empty bitmaps drawn over them to define the "hotspots".

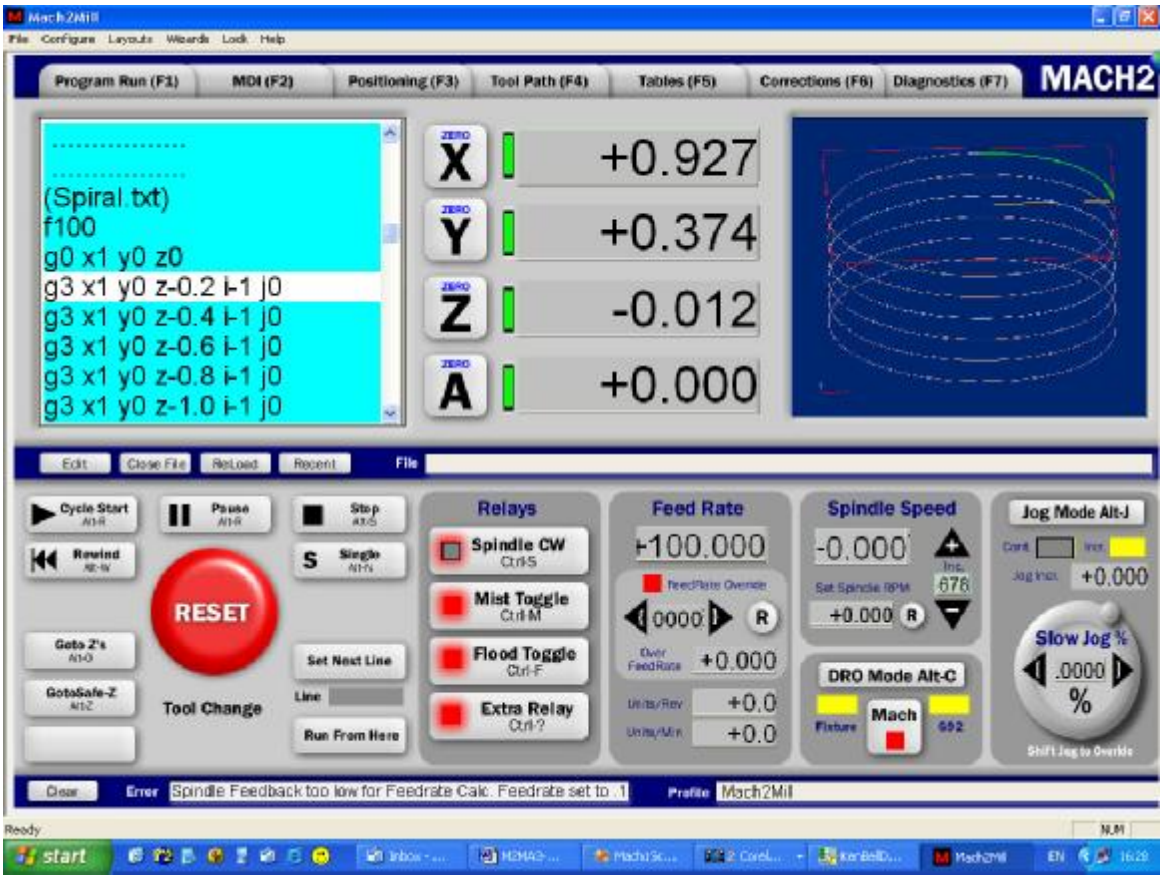


Figure 3.10 – Screen with all graphics on bitmap and transparent buttons

Another advantage of this type of screen design is that it is fully scalable to different screen resolutions using ScreenTweak as it does not have the problem of the fixed size text of conventional buttons.

3.5.4.4 Dynamic changes with bitmaps

The Screen Designer does not copy the bitmaps when you place them on the screen at design time; it just sets up a link to the image file path relative to the Mach2\Bitmaps folder. The bitmap is loaded when Mach2 itself is run. This means that you can replace the bitmaps with ones of your own design and by giving them the names used by the original designer of the screen you can customise the appearance of your system without having to run Screen Designer or having to place or size the bitmaps. Figure 3.11 shows this done to the set of DROs shown in figure 3.9

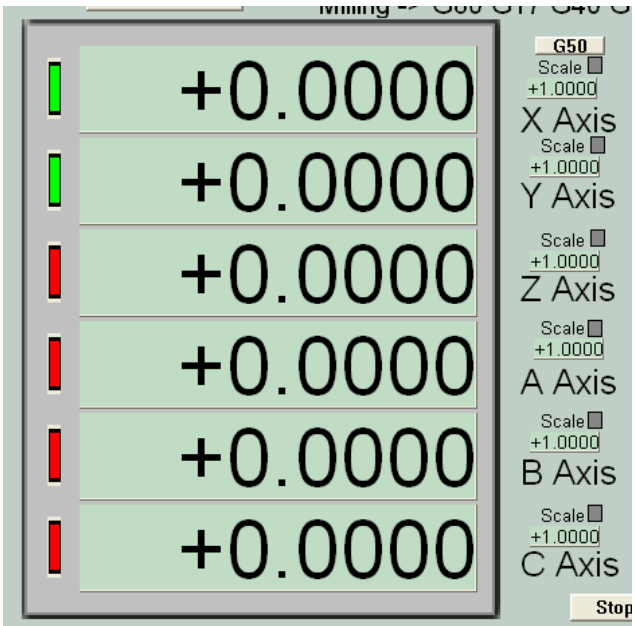


Figure 3.11 - An alternative frame link in dynamically

3.6 Advanced features for setting up controls

Figure 3.1 shows several other features of the dialogs for setting

up buttons. Similar features exist on the dialogs for other controls.

Positioning: The position of the top corner of the control and its size is given in the dialog. If you click the *Enable Input* button then this data can be altered and the new size and position will be applied to the control when OK is used to close the dialog.

Original size: The *Button will be bitmap's original size* checkbox on the dialog for bitmap buttons and a similar one for bitmaps allows the size of the control to be adjusted t the number of pixels in your bitmap as it was originally created. This option can be used with the option to specify a new position. If this is not used then the top lefthand corner is not altered.

As you can only change the size and position of a control that is on the screen's visible area, you need to be careful with the values that you enter. In an emergency you can move any controls that lie partly off a 1024 x 768 screen to a visible position using Edit>Retrieve Off-screens. This will also reposition controls intentionally placed just off the page such as labels that start with § used to identify screens to ScreenTweak.

Locking: The *Locked for Mouse* checkbox allows you to lock the control so that it cannot be moved or resized by the mouse or be nudged. Double-click will still be available to reset its properties.

You can also lock all the selected items on a given screen using Edit>Mouse-lock all selected. You can unlock all the locked controls on a screen using Edit>Clear Mouse-locks.

3.7 Colors

The Screen Designer ***Clr*** icon lets you define the colour scheme for your screens and controls. This is a global setting; it does not relate to the selected items.

3.8 Implementing two levels of screen complexity

It is clear that some users will require a full range of controls on a screen while others will find a machine easier to use with only the bare minimum of controls displayed. This requirement can easily be met by designing two (or even more) screen sets and setting up a profile to load each from its own shortcut.

On occasions, however, it is convenient to be able to switch the level of complexity dynamically. This is done by having the two screen sets in the same folder and with the same name but distinguished by the extensions .SET and .SSET The profile should initially load the .SET file.

The switch between sets is achieved by a button on the screen(s) which calls the VBScript routine `ToggleScreens()`.

You can see how this works on the standard mill screens by running Screen Designer of `1024.set` and `1024.sset`.

4. Coding VB Script programs

This chapter aims to help you write button or macro code of your own which can range from performing simple tasks (e.g. implementing non-standard M-codes from other controllers) to performing complex calculations on data which will be used in the part program. This might simulate, for example, engine turning as geometric pattern to decorate an object.

Warning: It is not advisable to write macros to call in part programs if you want them to be portable to other CNC controllers.

If you are considering writing VB Script then you will need to have some experience of simple computer programming (e.g. in Visual Basic, C or even BASIC). This chapter does not attempt to teach programming.

All the usual VB Script data and control structures can be used in the program. For details consult *Windows Script 5.6 Documentation* available for download by following some obvious links from:

http://msdn.microsoft.com/scripting/

The Microsoft presentation is rather oriented to scripts of web pages but the documentation that you can download summarises the data types, control structures, operators etc. very clearly. You will not need to use advanced features like Classes. Regrettably this information is not presented in a form which can conveniently be printed out so you might consider purchase of one on the many books on VB Script.

You will find that there are many operators for manipulating strings, doing trigonometry etc. VB Script is a complete and very capable language. The only rstrictions within Mach2 scripts are on the use of input/output functions and subroutines.

4.1 A simple button script

Many machine tools allow you to control the feedrate, spindle speed in a geometric progression (i.e. each step is so-much times the previous one).

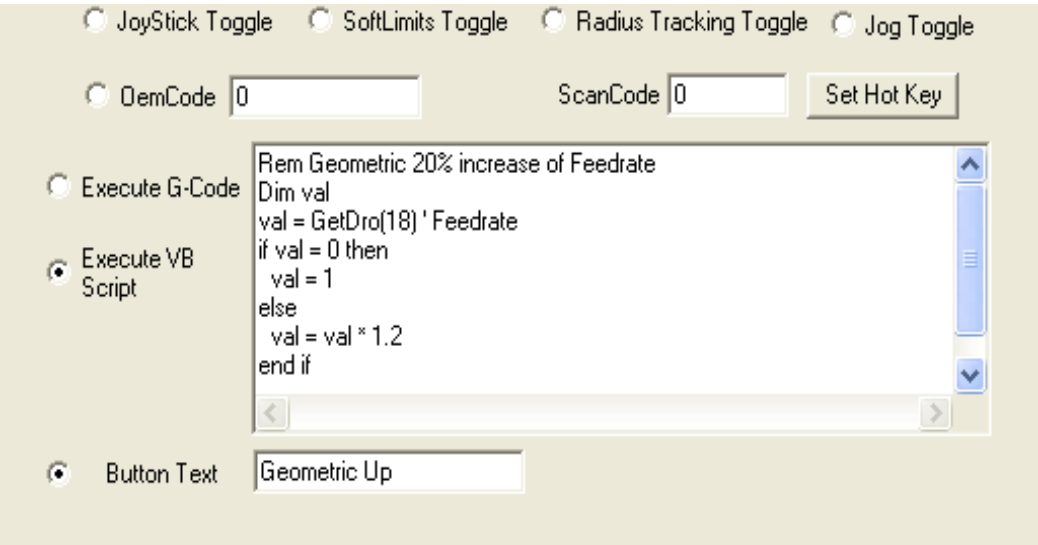


Figure 4.1 – Setting up the geometric feedrate button

Figure 4.1 shows the dialog box setting up a button that will increment the feedrate in this way. This operation is performed using the Screen Designer program. Full details of this are given below. For the present we concentrate on the VB Script code. Most of it is shown in the edit box of the dialog. The complete code is as follows:

```
Rem Geometric 20% increase of Feedrate
Dim val
val = GetDro(18) ' Feedrate
If val = 0 then
  val = 1
Else
  val = val * 1.2
End If
Call SetDro(18,val)
```

The first line is a Remark, ignored by Mach2 but useful to remind one what the button does. The Dim statement declares a variable to use again not essential. GetDRO (18) reads DRO function code 18 (which is the commanded feedrate) and assigns it to the variable *val*. The quote ' character introduces a comment on the end of a line of code. If *val* is zero then it is set to 1 otherwise it is incremented by 20%. Finally the new *val* is stored in the Feedrate DRO so setting the required rate.

As you can see from the dialog the caption on the button will be "Geometric up". A similar button might be provided with code for Geometric Down.

4.2 Sample macros

4.2.1 A simple macro

The following is code to form a very simple Mach2 macro. Suppose it is stored in a file M99990.m1s (the extension is letter-m, figure-one, letter-s). The call M99991 could be included in the file at the beginning of each part program you want to run. If you set a feedrate by the F-word before running the program then it will use that rate otherwise the macro will ask the operator for a value.

```
Rem Default feedrate setting macro
Dim NewRate, FRateFunCode
FRateFunCode = 18    ' easier to read program if you use a variable
If GetDRO (FRateFunCode) = 0 Then
     Newrate = Question ("What should the feedrate be for this job")
     Call SetDRO (FRateFunCode, Newrate)
End If
```

4.2.2 More complex macro

The following is more complicated example of a Mach2 macro. Suppose it is stored in the file M99992.m1s It might be useful to edit M99992 into the start of a part program if the post-processor that created the program has not taken care to set all the modal values at the start of the program. It would also be handy macro to call by `M99992 Q100 S2000` from manual data input (MDI) to set up a feed and speed and the modes before running a program.

```
Rem This macro sets modals to defined values. Useful at start of
'         a part program if you are not certain about the default
'         state of your controller
Rem The P-word can be used to choose options:
'         Px1  gives inch units (Px0 gives mm)
'         P1x gives constant velocity (P0x gives exact stop)
'            e.g. P10 gives CV and mm
Rem The Q word can set a feed rate. The S word can set a
'          spindle speed
Rem The code is written to illustrate some features of VB Scripting
'         so includes declarations that are not strictly needed etc.
'         The logic can be done in other ways. You might like to
'         try some!

Option Explicit ' declarations are mandatory

Dim TxtChoice, PValue, QValue, SValue
PValue = Param1()
  Select Case PValue
    Case -1
      TxtChoice = "G21G61"
    Case 1
```

```
    TxtChoice = "G20G61"
  Case 10
    TxtChoice = "G21G64"
  Case 11
    TxtChoice = "G20G64"
  Case Else
    Code "(MSG,P word is not valid - defaults are set)"
End Select

  Call Code (TxtChoice & "G17G40G49G50G90G94")  ' set the modes


  QValue = Param2() : SValue = Param3()  ' get values

  If QValue >0 Then           ' a value has been given
      SetFeedRate (QValue)    ' using a Mach2 function
  End If
  If SValue > 0 Then
        code "S" & SVal    ' using G-code where there is no function
  End If
```

Commentary on program

As mentioned before the comments, lines starting with Rem or the text following an apostrophe ' are not interpreted by the system.

Unlike the C and C++ languages, VB Script is not case sensitive so `NEWRATE`, `NeWRate`, `newrate` are all the same variable. You can, however, use case to make your programs easier to read.

Variables can be used in the VB Script code to store numbers or character strings. They do not have to be declared but using a `Dim` declaration helps other people to follow your program.

The program next accesses the value of the P word in the macro call using the Mach2 standard function `Param1()`. The brackets show that a function is being called. The value, in this case a number, is assigned to the variable `PValue`.

Next the `Select Case` control structure is used to assign a different string to `TxtChoice` depending on the value of the P word. Other control structures available include `If Then Else End If` and various forms of loop.

The VB Script program can interact with the operator by asking for a value by using the Mach2 function Question. In the example above no value is used. If a value is required then the form `x = Question ("What is the new X value")` is used.

Next the program uses the `&` operator to concatenate (join) the `TxtChoice` string which is already set up with the standard initial G values and uses the Mach2 function `Code` to send the string to Mach2 to be interpreted.

Finally the R and S words are accessed by `Param2()` and `Param3()` and, if used, are sent to Mach2 by the function `SetFeedRate` and another use of `Code`.

4.3 A common confusion with VB Script and a hint

You may have noticed two different ways of calling the Code subroutine in the second example.

```
Call Code (TxtChoice & "G17G40G49G50G90G94")  and
Code "S" & SVal
```

Could have been written

```
Code TxtChoice & "G17G40G49G50G90G94"  and
Call Code ("S" & SVal)
```

Either is correct and both do the same thing. If you like to group the arguments with brackets then you need to use the keyword `Call` as well.

A function can be called and have its result thrown away as in this example:

```
Question ("Did you know P word was not valid - defaults are set")
```

Using brackets with no `Call` show that it is a Function which is expected.

Sometimes you can get away with breaking these rules but you will find that you get very hard to find bugs!

If you include the statement `Option Explicit` at the start of your code the VB Script will require you to declare all variables (e.g. with `Dim` statements). This is very helpful in trapping bugs caused by misspelled variable names.

4.4 The Mach2 VB Script functions and subroutines

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

4.4.1 To execute G or M-codes from a script

The simplest and most powerful routine is:

```
Subroutine 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 Mach2 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"
```

4.4.2 For accessing the screen controls

As you have seen in the earlier examples, 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 Mach2 and its Screen Designer for the DRO, LED or button operation you want to use.

There are, for historical reasons, two different code lists for each type of control. The original built in controls are described by their Function Code. Later features have controls described by so called "OEM" Codes. You will have seen the reason for this if you have already tried Screen Designer. If you have not used Screen Designer have a look at the figure showing the Properties of the Standard Cycle Start button in this chapter. The buttons that are defined by checking one of the Radio Buttons in the dialog are referred to by Function Codes. Buttons defined by checking OEM Code and entering a Code value in the box are referred to by OEM Codes.

DROs and LEDs can be defined that have no meaning to Mach2 being solely for you use. There are 255 of each denoted by "OEM" 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.

The Function code numbers start at zero in the top-left and count up across the screen. Thus in the Set Button screen, Run (i.e. the Cycle Start button) is Function = 0 and Zero Y is Function = 9. Rather than looking up codes using Screen Designer, you may find it more convenient to refer to the complete table of codes in the appendix.

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:

```
JoyStickLEDFn = 14
:
:
bJoy = GetLed (JoyStickLEDFn)
```

LEDs

Function GetLED (ledFun **as** Integer) **as** Boolean

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

Function GetUserLED (ledUserCode **as** Integer) **as** Boolean

Choose the appropriate routine depending on whether you want to access a built-in, OEM or User LED. ledUserCode must be in the range 1000 to 1244. 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 = GetLed (14)          ' set variable bJoy if Joystick is enabled
If GetOEMLed (29) Then ….. ' see if a Fixture is in use
SetUserLED (1002, 1)              ' turn on user LED
```

DROs

Function GetDRO (droFun **as** Integer) **as** Double

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

Function GetUserDRO (droUserCode **as** Integer) **as** Double

Choose the appropriate routine depending on whether you want to access a built-in, OEM or user DRO. droUserCode will be in the range 1000 to 1254. The result is the current value displayed by the DRO.

Sub SetDRO (droFun **as** Integer, newValue **as** Double)

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

Sub SetUserDRO (droUserCode **as** Integer, newValue **as** Double)

Choose the appropriate routine depending on whether you want to access a built-in, OEM or user DRO. droUserCode will be in the range 1000 to 1254. 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 = function 6) then you cannot set it in a script.

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

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 SetDRO (18, GetDRO (18) * 1.1)     ' increase feedrate by 10%
```

Button Commands

Sub DoButton (buttFun **as** Integer)

Sub DoOEMButton (buttOEMCode **as** Integer)

Choose the appropriate routine depending on whether you want to use a built-in or OEM command. Mach2 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
```

4.4.3 Interrogating Mach2 internal variable

The current value of Mach2 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)
```

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.

4.4.4 Access to the machine G-code parameter block

Mach2 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 in chapter 11.

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))
```

4.4.5 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
```

4.4.6 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.

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.

```
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).

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

A development feature for speaking a text string. Requires detailed setup for which some information is available in the MACHDN online archive.

User defined DROs

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 DROs which are not used by Mach2 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 Mach2 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 macros 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 Mach2 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.

e.g. `SetUserLabel 12, "You must enter a whole number of holes"`

```
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. not very seriously:

```
SetTicker 205, "This is a very long error message because
you seem to have done something very silly"
```

4.4.7 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 LoadRun (pathname as String)
```

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

```
Sub OpenTeachFile (pathname as String)
```

Opens the given file for G-code and starts writing of commands executed (e.g. by MDI) to it.

```
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.

4.4.8 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 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.

4.4.9 Input/Output signals, a serial port and "foreign" ports

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

Sub ActivateSignal (sigNo **as** Integer)

Sub DeActivateSignal (sigNo **as** Integer)

IsActive tests input signals. It 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. Mach2 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 in the appendix.

Sub SetTriggerMacro (script **as** String)

Defines the number of a macro to be executed when an OEMTrigger is set (slightly unexpectedly on the Config>SetHotkeys dialog) to generate OEM code 277. 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 277 will run the code in the file M543.M1S when activated.

4.4.10 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 Configure>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))
```

4.4.11 Foreign ports

Scripts can access ports on the PC which are additional to the one (or perhaps two) parallel port(s) defined in Configure>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.

4.4.12 Waiting and system features

As described above the script code and Mach2 itself run in two separate processes. You can test to see in Mach2 is busy or idle by calling:

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

This will return True if Mach2 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 = GetDRO (2)  ' get Z value in case it has been Z inhibited
```

```
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). This can be used to inhibit script actions like `Question()`.

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

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

```
Function IsFirst () as Boolen
```

Returns True if this is the first call of the function after Mach2 has exited from the EStop state. This can be used to re-initialse data that would be lost at a n EStop.

4.4.13 A more complicated macro example

The next example is used to show the features available in more detail and to point out some of the difficulties in using macros. The macro is designed to perform a number of roughing cuts to reduce the size of a piece of bar. Figure 4.2 shows the operation partially completed. The macro asks the operator for the X values of the ends of the piece to be cut, the initial Z position of the top and the final Z coordinate of the top when the roughing has been done.

Figure 4.2 – Roughing a bar

For the sake of illustration we suppose it is to be called by M62 and so will be stored in the file M62.m1s

```
Rem This macro M62 makes a series of roughing cuts to reduce a
'       non-standard piece of material to a desired initial size.
Rem The P-word imust be provided to give depth of cut to be used
Rem The tool diameter and area to be machined are input in response to
'       prompts from the macro. The units are the currently selected ones.
'       The macro leaves the machine in Absolute distance mode
Rem The code is written to illustrate some features of VB Scripting so
'       includes declarations that are not strictly needed etc. The logic
'       can be done in other ways. You might like to try some!

Option Explicit

Dim TxtChoice, CutDepth, StartX, EndX, StartZ, EndZ

    CutDepth = Param1()
    While CutDepth <= 0         ' the P word is not given
         CutDepth = Question ("What depth of cut do you want each pass?")
    Wend

       StartX = Question ("What is X of left end?")
       EndX = Question ("What is X of right end?")
       While EndX <= StartX
           EndX = Question ("X right must be > left. What is it?")
       Wend


       StartZ = Question ("What is Z of top of material?")
       EndZ = Question ("What is desired final Z?")
       While EndZ >= StartZ
           EndZ = Question ("Final Z must be below initial Z. What is it?")
       Wend
       code "G0Z" & StartZ
       CurrZ = StartZ       ' top of work
      code "G0X" & EndX    ' cutter to  right
       code "G1Z" & CurrZ   'feed down for first cut

       While CurrZ > EndZ              ' loop for the z planes cut
           CurrZ = CurrZ - CutDepth    ' Update Z
           If CurrZ < EndZ Then
               CurrZ = EndZ            ' last cut dead to size
           End If
           code "G1Z" & CurrZ          ' feed down
           code "G1X" & StartX         ' cut pass
           code "G0Z" & (CurrZ + 0.1)  ' avoid scratching surface
           code "G0X" & EndX           ' back to right end
```

```
    Wend
    code "G0Z" & StartZ
    code "G0X" & StartX

Rem End of M62
```

In this macro, which will probably be used from MDI the P word is used to define the depth of cut in each pass. So a typical call using inch units might be `M62 P0.05`

The macro then asks a series of questions to establish the X coordinates of the ends of the bar and the original and desired Z values. These values are validated and re-prompted for if they are invalid.

A `While` loop then performs cutting moves with ever decreasing Z until the desired size is reached.

4.5 Script Snags and Hints

4.5.1 What Windows/Mach2 does with your macro

In order to understand some of the limitations of macros and to help you debug them, you need to understand what Windows and Mach2 do when you run a script from a button or a macro from a part program or MDI.

There are three stages in running a script. The last two proceed in parallel:

Analysis: The lines of text in your script or macro file are read by the Windows VB Script engine and converted into a more compact internal symbolic form.

Macro run-time: The VB Script program is run. At this stage the user will be asked and will answer questions. Calls to the Mach2 functions will be made. These will return values or state of an input pin (`IsActive()`) and write requests for G-code to be executed (`Code` etc.).

The function `ismoving()` can be called to see if the G-code runtime thread (see below) is executing commands. It returns a non-zero value if commands are being executed or moves are buffered.

G-code run-time: Mach2 which is running as a separate "thread" executes the G-code.

This scheme has several implications for you which are described below.

4.5.2 Script error reporting

Errors discovered in the analysis stage (e.g. mis-spelling `while` as `whyle`) are reported with the offending line displayed. This will make them fairly easy to correct.

Unless you have a Microsoft debugger from Visual Studio installed on your computer, errors at macro run-time do not give any useful diagnostics. Errors can range from the wrong number of arguments passed to a function, division by zero etc. You may be able to work out where they are from the Questions asked, by putting in extra "dummy" Questions or by the use of `On Error Resume Next` followed by a call to the Mach2 `Question` function to display the error message stored in the global variable `err.description`. Note that, as any G-code issued is not running in synchronism with the script, you cannot tell where you are by seeing the axes move or the DROs change.

Errors when the G-code runs can be very difficult to trace as Mach2 tends to ignore requests it does not understand rather than flag them up to you. You will just have to "dry run" the macro code to see exactly what is being requested. It is sometimes helpful to store a line of G-code that you have constructed in a variable so that you can print it in a "debugging" Question or Message to check that it looks right.

4.5.3 Stuck in a rut?

As you have some very powerful control structures it is quite easy to get a script stuck in a loop. For example in this code fragment `EndX` is mistyped `EndZ`

```
While EndX <= StartX
     EndZ = Question ("X right must be > left. What is it?")
Wend
```

There is nothing that the operator can type that will correct the invalid EndX value for the code loops for ever. You can only get out of this by using Control-Alt-Delete and getting Windows to end the Mach2 program. In this case you must run OCXTest to reset the driver or re-boot Windows.

If the line in grey in this code

```
While CurrZ > EndZ               ' loop for the z planes cut
    CurrZ = CurrZ - CutDepth     ' Update Z
    If CurrZ < EndZ Then
        CurrZ = EndZ             ' last cut dead to size
    End If
    code "G1Z" & CurrZ           ' feed down
    code "G1X" & StartX          ' cut pass
    code "G0Z" & (CurrZ + 0.1)   ' avoid scratching surface
    code "G0X" & EndX            ' back to right end
Wend
```

is omitted or if `CutDepth` could be zero or negative then the `While` will run for ever. This is worse than the loop above because each time round it requests the execution of some G-code. Eventually the buffer will overflow and may well crash Mach2 in totally unpredictable ways.

4.5.4 Reporting errors to users

If you accept data from users by Question() or by user DROs in a conversational programming screen then you need to validate this data with logic in the VB Script. For example, few mills will drill holes with negative depth successfully!

You can write messages to the error line by exploiting the Message script call

```
Message "P word is not valid - defaults are set"
```

After a user error, you may not be able to take a default action and so may wish to exit from your script. The VB Script language does not have a suitable `Exit` command that can be used in the main program (which is what most scripts will be) or a GOTO so you may end up with very deeply nested `if` commands.

A way of avoiding this is to make the code of your script be a subroutine called, say, `MainProg`. The actual main program just calls this. Now, because Exit Sub is valid in a subroutine and immediately exits from it you can easily abort the run of your script. A skeleton of the whole cone might look like:

```
Sub MainProg
' get data…from DROs..
'
Rem now check the user data
If UserFooVal > MaxVal Then
      Message "You are not allowed values of " _
      & "Foo greater than" & MaxVal
      Exit Sub ' bail out because of error
End If
Rem script continues to use good value…..
'
'
End Sub    ' MainProg
MainProg   ' call the actual code when button/macro used
```

4.6 Legacy/System VB Script Functions

The following functions are still available to writers of macro scripts at Release 2.7 of Mach2. Their general use is, however, deprecated as better and more general ways are available 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 GetABSPosition(axis as Byte) as Double
```

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

```
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 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 OpenDigFile()
```

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

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

Depth of planner queue is returned.

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

Sub SetCurrentTool(Tool **as** Byte)

Will return currently 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 RunFile ()

Executes the currently loaded G-code file.

Sub SetIJAbs()

This will set the IJ mode to absolute

Sub SetIJInc()

This will set the IJ mode to incremental

Sub SetIJMode(mode **as** Integer)

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

Sub SetSafeZ(SafeZ **as** Double)

This will set the Safe_Z

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)

Rapid move.

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

Gets 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.

5. Designing wizards

The techniques of VB Script and custom screens can be combined to implement "conversational programming" of the machining of a design using Mach2. Examples of this feature have been shown earlier in the manual. This section gives details of how you can write your own wizards.

5.1 What is a wizard?

A Mach2 wizard is feature which allows you to create a G-code part program by filling in some simple information on a special screen or screens. Wizards provide open-ended capability to Mach2 as they can be written and shared by users and are trivial to install on a system.

Examples of wizards supplied with the standard release of Mach2 allow cutting circular and rectangular pockets, digitising a model and engraving text.

Wizard functions can replace the need to use a full CAD/CAM software system for some prototype work. You do not need to be able to write or understand G-code to use a wizard although the code generated can be a useful learning resource. You can, for example, see the steps required to cut a pocket by looking at the code produced by running the wizard.

5.2 A wizard's working in a nutshell

The basic operation of a wizard is very straightforward:

- The user chooses a wizard from the table of those installed on the system
- Mach2 sets aside the standard screens and displays the wizard screen (or possibly the first of a set)
- The user provides information to the wizard by entering values in DROs and by using buttons to switch LEDs On and Off. For example the diameter of a pocket, the depth of each cut and the required spindle rotation could be specified.
- The wizard checks that the data that has been provided describes a possible operation. It is, for example, very difficult to cut a 1" diameter pocket with a 30mm diameter cutter!
- The wizard then writes a file (like a Teach file) containing the G-code to perform the required task and loads this into Mach2
- Finally the user exits from the wizard and returns to the normal Mach2 screens.

Before attempting to design your own wizard you need to be familiar with using Screen Designer and writing your own VB Script to attach to buttons or to use in macros. In addition, you should experiment with the standard wizards. You will see that they have slightly different user interfaces because they are written for different purposes by different people.

The following sections give a step-by-step tutorial for how the Digitize wizard works and some general advice on creating easy to use and reliable wizards.

We will assume, in this chapter, that Mach2 is set up in metric units although, of course, you can use inch units just as well.

Creating a wizard that works for **you** is quite straightforward; it is much more difficult to include all the checks to guide a user who does not really understand machining.

5.3 Worked example – the Digitize wizard explained

This tutorial will guide you through the development of the Digitize wizard currently found in the released Mach2 add-ons folders. Its focus will not be primarily on the aesthetic design of wizards but on the technical aspects involved in the design. Our discussion will

detail the use of user DRO's, teaching files, and restoring the system screens after the wizard has run.

The wizard creates a part program which moves over a rectangulare area in the X and Y plane plunging a probe in the tool holder to discover the Z height of a model object on the table. If you have not used the Digitize wizard, now would be a good time to try it. It does not matter if you have not got a probe – just stop before actually running the part program.

5.3.1 The first step

To start writing a wizard you create a new folder in the Addons directory which can be found in the main Mach2 system directory normally located at C:/Mach2. (see figure 5.1)

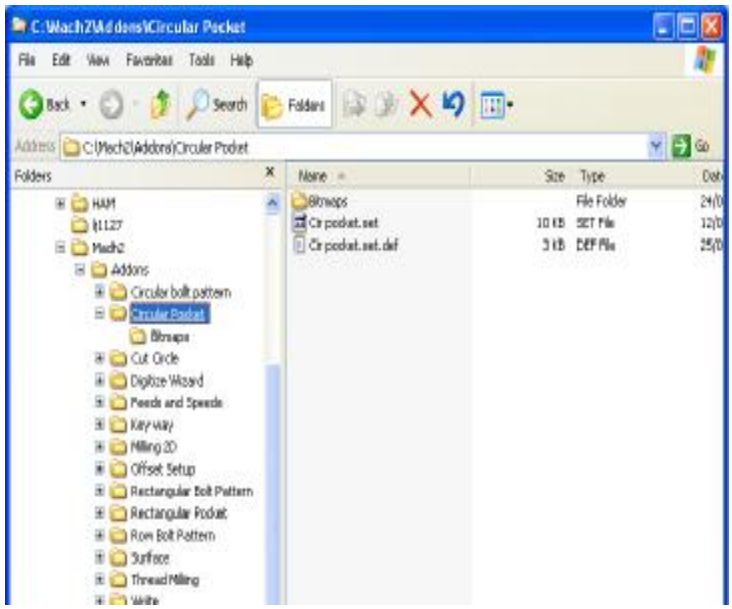


Figure 5.1 – The Addons folder in C:/Mach2

Since this is a wizard for digitizing we will name our folder **Digitize**.

When Mach2 is started it searches the Addons folder for the names of all subdirectories it can see. These names are used to build up a table of names to be displayed to the user to select from when using the Wizards menu

Inside the Digitize folder we will put a further folder called Bitmaps. It is best that you keep bitmaps which are used for a particular wizard's screen(s) in its own bitmaps directory. This helps to keep the clutter down on your system. Now that we have created the folders necessary to hold the wizard, we can begin to create our wizard.

A wizard is simply a screen which is used for data entry by the user and has some buttons to start using the data when it is complete. Most wizards will generate G-code to be run after leaving the wizard screen. While it is possible for wizard to actually run G-code, this is an advanced use of the wizard and it is not recommended for inexperienced wizard writers.

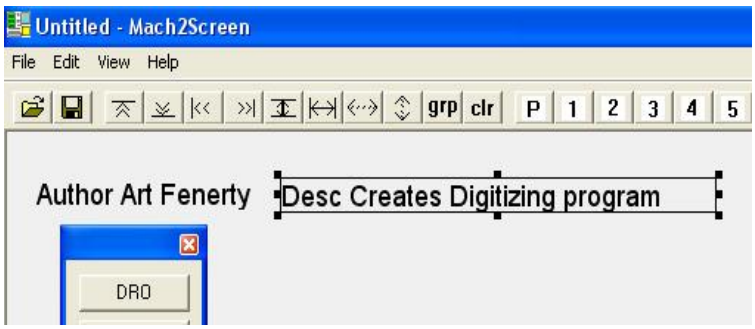


Figure 5.2 – The smart labels identifying the wizard

The Digitize wizard will have a single screen that you create by running the Screen Designer program. If you have not got a shortcut set up for it then run C:/Mach2/Mach1Screen.exe When this loads you will have a blank set of screens with screen #1 being displayed

The first thing to do is tell the program who wrote this wizard. See figure 5.2. To do this we create a label which says the word *Author* followed by the author's name. We also want to be able to describe to the user what this wizard will do so we create a label which begins with *Desc* which tells the program that the following text is the description to be displayed in the wizard selector. These two "smart labels" will not appear on screen when the wizard is run; they are only used for the wizard selector dialog.

Next we add the rest of the controls required for the user to define what the wizard is to do. We can label and tidy them up them later. You should save your layout in the folder you created for your wizard. The screen will look like figure 5.3

5.3.2 Making the wizard work

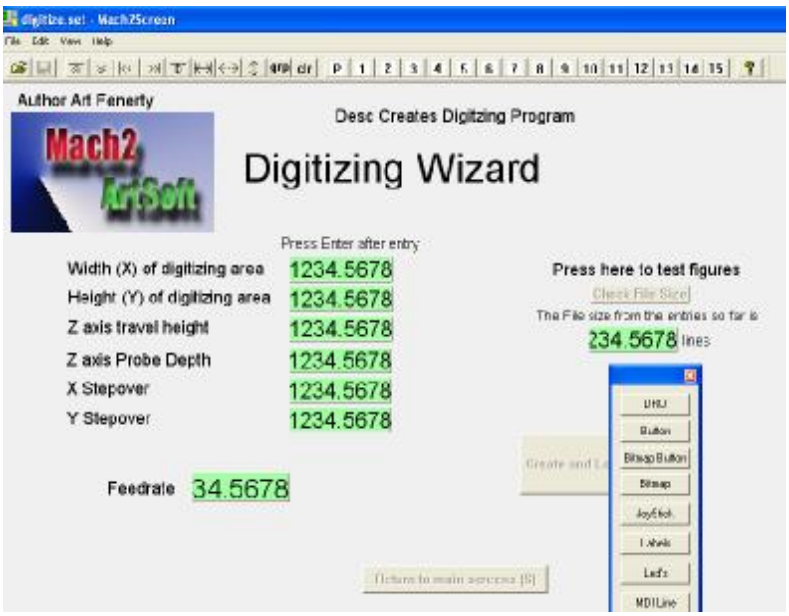


Figure 5.3 – Controls in place on screen

Although this screen looks complete it does not yet do anything. The next job is to define the functions of each control. Consider the *Width* DRO.

As you can see, in figure 5.4, it is given an OEM code of 1001. Any DRO which has a higher value than 1000 is called a User DRO. These User DROs are places to input, display and remember numbers. They can be used to save settings when a user wishes those settings to be persistent from one the wizard to another. So we continue by setting up the nine DROs which we intend to use to provide the data for our program. Each is given its own DRO number.

Figure 5.4 – The values for Width DRO

You can see that the wizard has a button to test the file size that will be created by running the finished wizard. This is important, as the user may not realise the implications of probing at locations that are too close together! Let us look at how this button can be made to work. Figure 5.5 shows what double-clicking the finished button gives.

Its function is chosen by the Radio button *Execute VB Script*. When this button is pressed the VB Script inside the button will be run.

It begins by getting the values of all the User DROs on the screen and assigning their value to variables. This makes the code doing the calculation much easier to read than if it just has OEM button numbers. We then do a simple calculation of the Width divided by the Step times the Height divided by its Step. This is roughly the number of probing positions.

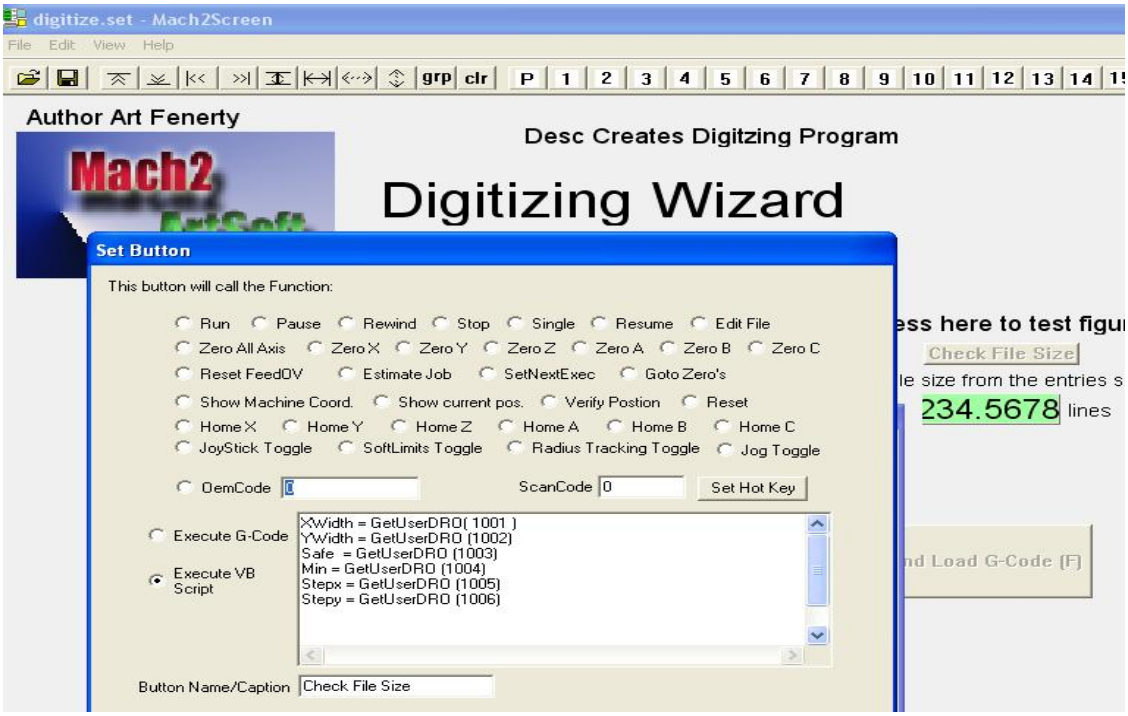


Figure 5.5 – The *Check File Size* button

We then send the result of the calculation to User DRO 1007.

There is very little to worry about in terms of what Mach2 is doing when running a simple script which does calculations. Some care is, however, needed when running a script which actually writes G-code data that you will later run.

5.3.3 Making the wizard write a part program

To understand generating code let us look at the Create and Load G-code button and analyze what it does

Here is the script which is stored in the create code button.

```
XWidth = GetUserDRO(1001)
YWidth = GetUserDRO (1002)
Safe  = GetUserDRO (1003)
Min = GetUserDRO (1004)
Stepx = GetUserDRO (1005)
Stepy = GetUserDRO (1006)
Feed =  GetUserDRO (1008)
```

It starts by assigning values to the variables for easy of reading the code

```
Rem now open a digitize.tap file
Rem all code commands then go to file.
OpenTeachFile "Digitize.tap"
```

The above line tells the system to open to teach file named digitize.tap. This is very important because without opening this file system will attempt to run the lines of code as it creates them. We just want them saving to run when we return to the main Mach2 screens..

```
code "(Digitize File)"
```

This puts out a G-code comment to the file so you'll know what it is when it is in the G-code window

```
Rem set the current location to 0,0,0 and open the file
code "M40"
code "G92X0Y0Z0"
code "F" & Feed
```

The above lines start the preamble with a G92 offset and set the feed rate according to what has been entered in the DRO.

```
direction = 0
' first iterative loop
for y = 0 to YWidth step Stepy
    if direction = 1 then
       direction = 0
    else
       direction = 1
    end if
    for x = 0 to XWidth step Stepx
       if direction = 1 then
           code "G0X" & x & "Y" & y & "Z" & Safe
       else
           code "G0X" & XWidth - x & "Y" & y & "Z" & Safe
       end if
       code " G31 Z" & Min
       code " G0 Z" & Safe
    Next x
Next y

if direction = 1 then
    code "G0X" & XWidth & "Y" & YWidth & "Z" & Safe
else
    code "G0X" & 0 & "Y" & YWidth & "Z" & Safe
end if
```

All of the above lines to the calculations and write the G31 commands which will actually do the probing in the program that you creating.

```
code "G0X0Y0Z" & Safe
code "G0X0Y0Z0 "
code "M30"
```

And finally wecomplete the program with an M30 to do a rewind.

```
CloseTeachFile
```

The above line closes the digitize teach file.

```
call LoadStandardLayout()
call LoadTeachFile()
```

In the above to lines tell the program to remove the wizard from the screen and load the screen layout that Mach2 normally runs with, and then load the Teach file into the system ready for cutting.

That would be the complete job but a user might want to create several digitizing jobs, and would not want to re-enter the data each time. The wizard can be provided with a button on its screen which tells the system to save the current settings when the wizard exits so that next time and is called the same settings will be in all the DRO's.

To do this we simply create a button, make it run a VB Script routine `Savewizard()`. This command tells the system to save all the DRO's that have been used into a file in the wizard's folder with a `.DEF` extension.

This saving works as follows. When the wizard is loaded, the first 200 OEM DRO's, ticker labels and LEDs are saved so that the wizard will not overwrite them. The next 55 DRO's labels and LEDs if they are changed by the wizard will stay changed when the wizard exits and control is passed back the Mavh2. If the `Savewizard()` script call is made then all 255 DROs, LEDs and ticker labels are stored in the `.DEFS` (for Defaults) file for loading next time the wizard is called.

5.3.4 A wizard that runs its own code

Now let us take a look at a bit of script which is run **in** the wizard itself. This will actually move the machine when the script is activated. The digitize wizard does not use any such code but as there are a couple things to worry about when designing that type of wizard and we need to look at them.

Here is an example script which makes a tool move down to table, hit a switch, move back up, so that we know the length of the tool.

```
Code "G28.1Z0"

While IsMoving()
Wend
```

The first line initiates the move and the next two lines cause the system to wait until the movement previously command it has completed. This is very important. Because the script runs in its own thread, synchronizing movements can be very complex. These two lines allow you to synchronize the movement you have asked for to ensure further commands do not happen too early. Always use the `While IsMoving()/Wend` loop when in any doubt.

As another more complete example:

```
Code "G31Z-100"
While IsMoving()
Wend
```

In the above lines we command a probe movement of the tool downwards from the home switch. It will stop when the tool hits a switch. Now that it has stopped, we can set the Z axis DRO to be zero with

```
SetDro(2,0)
```

The next two lines to move up 3 mm, which we would have previously calculated as the deceleration distance all the tools at that speed.

```
Code "G91"
Code "G0Z-3"
```

And finally, reset the Z. DRO to zero. Our tool is now zeroed.

```
Code "SetDRO(2,0)"
Code "G90"
```

5.3.5 Other precautions

You must take particular care when writing a macro to try to not call other macros. Unknown effects can be seen when one attempts to call a macro from a macro. The macro interpreter also can only hold so many lines at a time, is therefore good practice to issue a While /Wend loop every few lines of code if they command movement.

All macros in Mach2 run in their own thread. This can create problems in certain circumstances so a designer is advised to take flow into account. This is very important when G-code statements that modify the state of Mach2 are used. Things like feedrate changes, spindle speed, signal changing, condition testing need to be thought out to avoid problems if they reported too early. Take the following situation:

```
Code "G31Z-10″
Code "G92Z0
Code "G1Z5″
```

In these three lines, one would expect that the Z axis would probe downwards to a maximum depth of -10 mm, then zero the DRO and move to +5mm. What will happen instead though is that the Z will begin probing downwards, the G92 statement will zero the DRO while the Z moves down, the Z will finish at an unknown negative coordinate. Then it will lift way up to +5. This is because of the asynchronous nature of the macro interpreter. To do this properly, you need to have a

```
While IsMoving()
Wend
```

after the code `G31Z-10` statement. This will make your macro wait until the probe is done before actually commanding a G92Z0. Subsequent versions of the Mach system will reduce the need for this sequence but it will never do any harm to include it.

Armed with this basic introduction you can now study other Wizards. The facility is very powerful but you do need to be very careful in testing your code. Some hints are given in the next section.

5.4 Wizard design hints

5.4.1 Function

The function of the wizard can range from simple tasks, such as the setting up of work offsets, to complex routines, such as automated circular pocketing or engraving. In any case, it is very important that the function is laid out in well designed, logical steps. For example, to make a cut, you would first have to tell the tool to rapid to the starting point, then feed to depth, then feed to the end point, then retract the tool. The maths to calculate the tool paths is best done initially by hand, in order to ensure that you are going to get the results that you are looking for. If possible, try to make a list of what inputs you are going to need so that you can easily make DROs and/or LEDs for them when you design your screen.

5.4.2 Screen Design

A major component of any good wizard is the operator interface. So, the first step is to get out a piece of paper and sketch a design of the screen layout. The screen should have a nice work flow - uncluttered and intuitive. Inputs and DROs should be appropriately labelled, and functional controls should be grouped if possible (i.e. spindle controls, coolant, units, etc). To help the user to enter data is entered into the correct DROs, you can use bitmaps as diagrams. The circular pocket wizard shows good examples of this technique with both line and photographic illustrations.

5.4.3 Writing the Code

The VB Script code on buttons is the core of the wizard screens and can be made to do many different tasks. There is a minimum of three buttons that need to appear in every wizard – Exit, Post G Code, and Save Settings.

Figure 5.6 – Wizard interface buttons

The code for Exit is:

```
CloseTeachFile
call LoadStandardLayout()'reload the main screenset of Mach2
```

The code for Save Settings:

```
Savewizard()
```

Post Code is responsible for outputting the actual G-code to implement your wizard.

While the VB Script can be input directly into the button using its edit box in Screen Designer, it is better done in Notepad or in a code editor like ConText. After initially writing out the code the text editor, it can be put into the button by using copy (Ctrl-C) and paste (Ctrl-V). If you use a good editor you will get the benefits of syntax highlighting which will make your code easier to check.

We recommend using the line `Options Explicit` at the top of each piece of VB Script and declaring all the variables with Dim statements. This will avoid strange bugs caused by misspelled variable names.

5.4.4 Error checking

The example below shows a simple wizard that checks its data before using it.

Step One: Get the information from the wizard screen

```
PointX_1 = GetOEMDRO (1010) 'Get value from DRO# 1010
PointY_1 = GetOEMDRO (1011) 'Get value from DRO# 1011
PointX_2 = GetOEMDRO (1012) 'Get value from DRO# 1012
PointY_2 = GetOEMDRO (1013) 'Get value from DRO# 1013
Depth = ABS (GetOEMDRO (1014)) * -1
Rem Get value from DRO# 1014 and make it negative
Feed = GetOEMDRO (1015) 'Get value from DRO# 1015
```

Step Two: Error checking, Is the Info Good????

```
While Depth = 0
   Depth = Question ("Depth can't be 0," & _
            "what would you like for a new Depth:")
   Call SetUserDRO(1014, Feed) 'Set new value for DRO# 1014
Wend
While Feed <= 0
   Feed = Question ("Feed set wrong, " & _
            "what would you like for a Feed:")
   Call SetUserDRO(1015, Feed) 'Set new value for DRO# 1015
Wend
```

Step Three: Make a Program!

```
OpenTeachFile "Cutline.tap" 'Opens the Teach file
Code "G0 G49 G40 G17 G80 G50 G90 "
     'First line of code known as a safe start up block
Code "G00 X" & PointX_1 & " Y" & PointY_1
     'Move to Point 1 at rapid
Code "G01 Z" & Depth & " F" & (Feed / 2)
     'Plunge tool to depth at half Feed
Code "G01 X" & PointX_2 &" Y" & PointY_2 & "  F" & Feed
     'move to Point 2 at Feed
Code "G54 G00 Z0.000" 'Rapid to Machine Z axis zero
Code "M30" 'End the Program
```

Step Four: Exit and load

```
CloseTeachFile 'Close the Teach File you have open
Call LoadTeachFile()
     'Load the file (Cutline.tap) that was just made into Mach2
```

Suppose the user enters data:

PointX_1 = 0.000
PointY_1 = 2.000
PointX_2 = 4.000
PointY_2 = 6.000
Depth = -0.750
Feed = 10.0

The G-code in the teach file will like the following:

```
G0 G49 G40 G17 G80 G50 G90
G00 X0.000 Y2.000
G01 Z-.750 F5.0
G01 X4.000 Y6.000 F10.0
G54 G00 Z0.000
M30
```

5.4.5 Documenting the wizard

There are 3 major pieces required to properly document the wizard – the folder name where the wizard is stored, the description, and the author. All three pieces of information are displayed on the "Pick wizard" dialog. The column on the left most side is the name of the directory in which the wizard screen file resides. This directory is located in the `<Mach2>/Addons` folder. It is important to name the wizard appropriately, i.e. "Circular Pocket" as opposed to "wizard_01". The center column holds a description of the wizard's function. The description is added by placing a label control on your wizard screen design and setting the text to be `Desc` *this would be the description*. The `Desc` is how Mach2 knows that it is a description and all of the text after the `Desc` is what is displayed. This label is hidden from view when actually running the wizard screen. The right side column displays the author's name. Setting this is exactly like setting up the description – place a label control on your wizard screen design and set the text to be `Author` *your name here*. This label is also hidden from view on the running wizard screen. You should always include an author label on your wizards so that you can receive proper credit for all of your hard work.

5.4.6 Troubleshooting

There are a few things that can be done to help debug the code.

The most important is to have descriptive variable names. This will help in making the code more readable. For example, instead of having an input for the Rapid Plane and calling it "x", name the variable `Rapid_Plane` – not only does it save you the trouble of having to jot down a note to tell you that "x" is the Rapid Plane value, but it's a lot easier to follow that value through any equations you do with it, to pick out any possible mistakes.

Another good idea is to add a toolpath window and a code window to the wizard screen. These allow you to view the code and toolpath, and this helps in testing because you can change the DRO values and check for errors.

Also, be careful when using loops - they are great for doing repetitive tasks, but have the potential to lockup the computer (endless loops)! To keep this from happening, you can add an `If` statement to see if the loop is run more than an arbitrary number of times, and then break out of the loop if it exceeds that number.

6. Appendix 1 – Reference tables for Codes

6.1 Keyboard shortcut codes

	HotKey	DRO/Button	ScanCode	Fn/OEM Code
	- <minus>	Decrease spindle speed	109	32/164
	' <sQuote>	Red reset button	192	21/0
	/	Reset feedrate override	191	14/0
	[	Reduce slow jog speed	219	32/112
	]	Increase slow jog speed	221	32/111
	+	Increase spindle speed	107	32/163
	<Alt>1	Choose Program Run	2097	32/1
	<Alt>2	Choose MDI	2098	32/2
	<Alt>3	Choose Manual	2099	32/4
	<Alt>4	Choose Toolpath	2100	32/3
	<Alt>5	Choose Offsets	2101	32/7
	<Alt>6	Choose Settings	2102	32/6
	<Alt>7	Choose Diagnostics	2103	32/5
	<Alt>A	Cycle MPG jog axes	2113	32/175
	<Alt>H	Go to Home	2120	32/138
	<Alt>i	Reset interpreter	2121	32/102
	<Alt>J	Cycle jog step size value	2122	32/171
	<Alt>N	Toggle single block execution mode	2126	4/0
	<Alt>R	Cycle start	2130	0/0
	<Alt>S	Stop cycle	2131	3/0
	<Alt>U	Toggle inch/mm units	2165	32/106
	<Ctrl><Alt>J	Jog On/Off toggle	34890	32/103
	<Ctrl>F	Toggle flood coolant	32838	32/113
	<Ctrl>J	Cycle Cont/Step/MPG jog modes	32842	32/245
	<Ctrl>M	Toggle mist coolant	32845	32/114
	<Ctrl>O	Goto zero position	32847	17/0
	<Ctrl>S	Toggle joystick enable	32851	28/0
	<Ctrl>V	Verify axis referencing	32854	20/0
	<Ctrl>W	Rewind part program	32855	2/0
	<Ctrl>X	Select X axis DRO	32856	0/0
	<Ctrl>Y	Select Y axis DRO	32857	1/0
	<Ctrl>Z	Goto safe Z position	32858	32/104
	<space>	Pause	32	1/0
	Delete	Delete block toggle	46	32/176
	End	Optional stop toggle	35	32/177
	F10	Reduce feedrate	121	32/109
	F10	Zero encoder Y DRO	121	32/134
	F11	Increase feedrate	122	32/108
	F11	Zero encoder Z DRO	122	32/135
	F5	Toggle spindle CW	116	32/110
	F6	Goto tool change position	117	34/0
	F9	Zero encoder X DROs	120	32/133

Note: The shortcuts for any given special layout can be found in the CSV file which can be exported using Mach2ScreenTweak

6.2 Button, LED and DRO codes

Type	Function	FCode	IsOEM	OEMCode
1	X DRO	0	No	
1	Y DRO	1	No	
1	Z DRO	2	No	
1	A DRO	3	No	
1	B DRO	4	No	
1	C DRO	5	No	
1	X Vel DRO	6	No	
1	Y Vel DRO	7	No	
1	Z Vel DRO	8	No	
1	A Vel DRO	9	No	
1	B Vel DRO	10	No	
1	C Vel DRO	11	No	
1	Jog Inc Inc DRO	12	Yes	1
1	Pulse Freq DRO	12	Yes	2
1	Slow Jog % DRO	12	Yes	3
1	X min DRO	12	Yes	4
1	Y min DRO	12	Yes	5
1	Z min DRO	12	Yes	6
1	A min DRO	12	Yes	7
1	B min DRO	12	Yes	8
1	C min DRO	12	Yes	9
1	X max DRO	12	Yes	10
1	Y max DRO	12	Yes	11
1	Z max DRO	12	Yes	12
1	A max DRO	12	Yes	13
1	B max DRO	12	Yes	14
1	C max DRO	12	Yes	15
1	X G92 Axis Off DRO	12	Yes	16
1	Y G92 Axis Off DRO	12	Yes	17
1	Z G92 Axis Off DRO	12	Yes	18
1	A G92 Axis Off DRO	12	Yes	19
1	B G92 Axis Off DRO	12	Yes	20
1	C G92 Axis Off DRO	12	Yes	21
1	Queue Depth DRO	12	Yes	22
1	Time Scale DRO	12	Yes	23
1	PWM Base DRO	12	Yes	24
1	Torch Corr Sp DRO	12	Yes	25
1	Torch Height Corr DRO	12	Yes	26
1	Torch Height Max DRO	12	Yes	27
1	CPU Load DRO	12	Yes	28
1	Enc X DRO	12	Yes	29
1	Enc Y DRO	12	Yes	30
1	Enc Z DRO	12	Yes	31
1	X axis Ref Sw DRO	12	Yes	33
1	Y axis Ref Sw DRO	12	Yes	34

Type	Function	FCode	IsOEM	OEMCode
1	Z axis Ref Sw DRO	12	Yes	35
1	A axis Ref Sw DRO	12	Yes	36
1	B axis Ref Sw DRO	12	Yes	37
1	C axis Ref Sw DRO	12	Yes	38
1	True spindle DRO	12	Yes	39
1	Worst Case DRO	12	Yes	40
1	Tool X Offset DRO	12	Yes	41
1	Tool Z Offset DRO	12	Yes	42
1	Tool Dia DRO	12	Yes	43
1	Tool Tip Rad DRO	12	Yes	44
1	Touch Corr DRO	12	Yes	45
1	Fixture # DRO	12	Yes	46
1	Part X Offset DRO	12	Yes	47
1	Part Y Offset DRO	12	Yes	48
1	Part Z Offset DRO	12	Yes	49
1	Part A Offset DRO	12	Yes	50
1	Part B Offset DRO	12	Yes	51
1	Part C Offset DRO	12	Yes	52
1	CPU Spd DRO	12	Yes	53
1	Safe Z DRO	12	Yes	54
1	Overidden Feed Rate	12	Yes	55
1	Pulley DRO	12	Yes	56
1	Max Speed DRO	12	Yes	57
1	Velocity per Rev DRO	12	Yes	58
1	X Scale DRO	12	Yes	59
1	Y Scale DRO	12	Yes	60
1	Z Scale DRO	12	Yes	61
1	A Scale DRO	12	Yes	62
1	B Scale DRO	12	Yes	63
1	C Scale DRO	12	Yes	64
1	Lowest Torch Correction DRO	12	Yes	65
1	Threading Entrance Angle DRO	12	Yes	66
1	Max Entrance Points DRO	12	Yes	67
1	Rotational Time Error DRO	12	Yes	68
1	Entrance Trigger DRO	12	Yes	69
1	Time Correction Derivative DRO	12	Yes	70
1	Normal Spin Counts DRO	12	Yes	71
1	Current Spin Counts DRO	12	Yes	72
1	Spin Adder DRO	12	Yes	73
1	Spin up/down incr.	12	Yes	74
1	Stock Size DRO	12	Yes	75
1	Laser X Grid	12	Yes	76
1	Laser Y Grid	12	Yes	77
1	Repetitions DRO	12	Yes	78
1	Lower Z-Inhibit By DRO	12	Yes	79
1	Z-Inhibit DRO	12	Yes	80
1	Port Bit-test DRO (diagnostic)	12	Yes	81
1	Anti-dive limit DRO	12	Yes	82
1	X Machine Coord DRO	12	Yes	83
1	Y Machine Coord DRO	12	Yes	84
1	Z Machine Coord DRO	12	Yes	85

Type	Function	FCode	IsOEM	OEMCode
1	A Machine Coord DRO	12	Yes	86
1	B Machine Coord DRO	12	Yes	87
1	C Machine Coord DRO	12	Yes	88
1	Blend factor DRO	12	Yes	89
1	reserved	12	Yes	90
1	G73 Pull-off value	12	Yes	91
1	Tangential lift threshold angle	12	Yes	92
1	Tangential lift Z level	12	Yes	93
1	reserved	12	Yes	94
1	reserved	12	Yes	95
1	reserved	12	Yes	96
1	CV Feedrate	12	Yes	97
1	Feed override increment value	12	Yes	98
1	Blended Velocity DRO	13	No	
1	Elapsed DRO	14	No	
1	Estimate DRO	15	No	
1	Curr Line no DRO	16	No	
1	Spindle requested DRO	17	No	
1	Feedrate DRO	18	No	
1	Tool	24	No	
1	Rot A diameter	25	No	
1	Rot B diameter	26	No	
1	Rot C diameter	27	No	
1	Jog Inc DRO	28	No	
1	X Fixture Orig Off DRO	30	No	
1	X Fixture Off DRO	30	No	
1	Y Fixture Off DRO	31	No	
1	Y Fixture Orig Off DRO	31	No	
1	Z Fixture Orig Off DRO	32	No	
1	Z Fixture Off DRO	32	No	
1	A Fixture Orig Off DRO	33	No	
1	A Fixture Off DRO	33	No	
1	B Fixture Off DRO	34	No	
1	B Fixture Orig Off DRO	34	No	
1	C Fixture Off DRO	35	No	
1	C Fixture Orig Off DRO	35	No	
1	Current Tool length DRO	36	No	
4	Cycle start	0	No	
4	Pause	1	No	
4	Rewind	2	No	
4	Stop	3	No	
4	Single	4	No	
4	Resume	5	No	
4	Edit File	6	No	
4	Zero All	7	No	
4	Zero X	8	No	

Type	Function	FCode	IsOEM	OEMCode
4	Zero Y	9	No	
4	Zero Z	10	No	
4	Zero A	11	No	
4	Zero B	12	No	
4	Zero C	13	No	
4	Feedrate reset	14	No	
4	Estimate Job	15	No	
4	Run from here	16	No	
4	GotoZs	17	No	
4	Coord System	18	No	
4	Verify	20	No	
4	Reset	21	No	
4	Ref X	22	No	
4	Ref Y	23	No	
4	Ref Z	24	No	
4	Ref A	25	No	
4	Ref B	26	No	
4	Ref C	27	No	
4	Joystick Toggle	28	No	
4	Softlimits Toggle	29	No	
4	Radius Tracking Toggle	30	No	
4	Jog Toggle	31	No	
4	Program Run	32	Yes	1
4	MDI	32	Yes	2
4	Toolpath	32	Yes	3
4	Positioning	32	Yes	4
4	Diagnostics	32	Yes	5
4	Corrections	32	Yes	6
4	Tables	32	Yes	7
4	Inc Inc Up	32	Yes	100
4	Inc Inc Down	32	Yes	101
4	Reset Interp	32	Yes	102
4	Jog mode toggle	32	Yes	103
4	Goto Safe Z	32	Yes	104
4	Home Z then X then Y, A, B, C - set machine coords	32	Yes	105
4	Units	32	Yes	106
4	Mach coords	32	Yes	107
4	Feed raise	32	Yes	108
4	Feed lower	32	Yes	109
4	Spindle CW, reset THC height	32	Yes	110
4	Slow Jog Up	32	Yes	111
4	Slow Jog Dn	32	Yes	112
4	Flood toggle	32	Yes	113
4	Mist toggle	32	Yes	114
4	Edit G-code	32	Yes	115
4	Zero radius DRO on X	32	Yes	116
4	Zero radius DRO on Y	32	Yes	117
4	Zero radius DRO on Z	32	Yes	118

Type	Function	FCode	IsOEM	OEMCode
4	Software limits	32	Yes	119
4	Touch button for Tool length offset	32	Yes	120
4	Tool Tab Save	32	Yes	121
4	Fixture Tab Save	32	Yes	122
4	Torch Enable Toggle	32	Yes	123
4	Torch Cal Zero	32	Yes	124
4	Enc Load X	32	Yes	125
4	Enc To X	32	Yes	126
4	Enc Load Y	32	Yes	127
4	Enc To Y	32	Yes	128
4	Enc Load Z	32	Yes	129
4	Enc To Z	32	Yes	130
4	Tool Path Toggle	32	Yes	132
4	Zero X Encoder	32	Yes	133
4	Zero Y Encoder	32	Yes	134
4	Zero Z Encoder	32	Yes	135
4	Tool Offset Tog	32	Yes	136
4	Fixture Off	32	Yes	137
4	Go Home	32	Yes	138
4	Part X Offset Touch	32	Yes	139
4	Part Y Offset Touch	32	Yes	140
4	Part Z Offset Touch	32	Yes	141
4	Part A Offset Touch	32	Yes	142
4	Part B Offset Touch	32	Yes	143
4	Part C Offset Touch	32	Yes	144
4	Tool X Offset Touch	32	Yes	145
4	Tool Z Offset Touch	32	Yes	146
4	Joy Throttle select	32	Yes	147
4	Touch Corr Enable Toggle	32	Yes	148
4	Auto Lim Override Toggle	32	Yes	149
4	OverRide Limits	32	Yes	150
4	SS on Act4 Toggle	32	Yes	151
4	reserved	32	Yes	152
4	reserved	32	Yes	153
4	reserved	32	Yes	154
4	Units/rev - Units/min toggle	32	Yes	155
4	Set this line as next to execute	32	Yes	156
4	Jog Follow	32	Yes	157
4	Joystick ON	32	Yes	158
4	Joystick OFF	32	Yes	159
4	Regen toolpath display	32	Yes	160
4	Zero X-Z to stock as defined in DROs (Turn)	32	Yes	161
4	Coordinate mode (G90/91)	32	Yes	162
4	Raise spindle speed	32	Yes	163
4	Lower spindle speed	32	Yes	164
4	Laser Probe Enable Toggle	32	Yes	165
4	Zero laser grid at current location	32	Yes	166
4	Z inhibit toggle	32	Yes	167
4	Ignore Tool Change toggle	32	Yes	168
4	Close current file	32	Yes	169
4	Re-load last file	32	Yes	170

Type	Function	FCode	IsOEM	OEMCode
4	Jog increment cycle	32	Yes	171
4	Clear error label	32	Yes	172
4	Spindle CCW toggle	32	Yes	173
4	Parallel Port Encoder3 MPG Jog Toggle	32	Yes	174
4	Cycle axis controlled by MPG	32	Yes	175
4	Block Delete "switch" toggle	32	Yes	176
4	Optional Stop "switch" toggle	32	Yes	177
4	Offline toggle	32	Yes	178
4	Display Abs Machine coordinates (i.e. macnine coords ON)	32	Yes	179
4	Display Work + G92 coordinates (i.e. Machine coords OFF)	32	Yes	180
4	Display Work coords (i.e. not with G92)	32	Yes	181
4	Home X, Home Z (Turn)	32	Yes	184
4	Select X for MPG	32	Yes	185
4	Select Y for MPG	32	Yes	186
4	Select Z for MPG	32	Yes	187
4	Select A for MPG	32	Yes	188
4	Select B for MPG	32	Yes	189
4	Select C for MPG	32	Yes	190
4	Select Jog Increment 1	32	Yes	191
4	Select Jog Increment 2	32	Yes	192
4	Select Jog Increment 3	32	Yes	193
4	Select Jog Increment 4	32	Yes	194
4	Select Jog Increment 5	32	Yes	195
4	Select Jog Increment 6	32	Yes	196
4	Select Jog Increment 7	32	Yes	197
4	Select Jog Increment 8	32	Yes	198
4	Select Jog Increment 9	32	Yes	199
4	Select Jog Increment 10	32	Yes	200
4	Feed override Off	32	Yes	201
4	Feed override Jog	32	Yes	202
4	Feed override Feed	32	Yes	203
4	Jog mode Continuous	32	Yes	204
4	Jog mode Step	32	Yes	205
4	Joystick On	32	Yes	206
4	Joystick Off	32	Yes	207
4	Clear Z tool offset (Turn)	32	Yes	208
4	Clear X tool offset (Turn)	32	Yes	209
4	Set stock correction to Zero (Turn)	32	Yes	210
4	Home X Home Z (Turn)	32	Yes	211
4	Home X (Turn)	32	Yes	212
4	Home Z (Turn)	32	Yes	213
4	Show recent G-code files list	32	Yes	214
4	Display history	32	Yes	215
4	Load G-code	32	Yes	216
4	Tool flip toggle (Turn front/rear toolposts)	32	Yes	217
4	Z-inhibit ON	32	Yes	218
4	Z-inhibit OFF	32	Yes	219
4	Port Bit-Test Set (diagnostic)	32	Yes	220
4	Anti-dive enabled toggle	32	Yes	221
4	THC Anti-dive OFF	32	Yes	222
4	THC Anti-dive ON	32	Yes	223

Type	Function	FCode	IsOEM	OEMCode
4	Flood ON	32	Yes	224
4	Flood OFF	32	Yes	225
4	Mist ON	32	Yes	226
4	Mist OFF	32	Yes	227
4	Load Teach file	32	Yes	228
4	Toolpath Machine/Job toggle	32	Yes	229
4	Display wizard selection window	32	Yes	230
4	Load the normal screens when wizard done	32	Yes	231
4	Simple Complex screen toggle	32	Yes	232
4	Output 4 ON	32	Yes	233
4	Output 4 OFF	32	Yes	234
4	Output 5 ON	32	Yes	235
4	Output 5 OFF	32	Yes	236
4	Output 6 ON	32	Yes	237
4	Output 6 OFF	32	Yes	238
4	Set Help context	32	Yes	239
4	Def-Ref all axes	32	Yes	240
4	Tangential toggle	32	Yes	241
4	Save XYZ to G59.254 work offset	32	Yes	242
4	do G0G53 to G59.254 offset location	32	Yes	243
4	Move to G59.254 with midpoint selection	32	Yes	244
4	Toggle Jog Mode through Cont/Step/MPG as relevant	32	Yes	245
4	Force Referenced on all axes	32	Yes	246
4	CV feed toggle	32	Yes	247
4	CV feed OFF	32	Yes	248
4	CV feed ON	32	Yes	249
4	Disable movement on axis X	32	Yes	250
4	Disable movement on axis Y	32	Yes	251
4	Disable movement on axis Z	32	Yes	252
4	Disable movement on axis A	32	Yes	253
4	Disable movement on axis B	32	Yes	254
4	Disable movement on axis C	32	Yes	255
4	Engine OFFline	32	Yes	257
4	Engine ONline	32	Yes	258
4	Select encoder jog on axis X	32	Yes	259
4	Select encoder jog on axis Y	32	Yes	260
4	Select encoder jog on axis Z	32	Yes	261
4	Select encoder jog on axis A	32	Yes	262
4	Select encoder jog on axis B	32	Yes	263
4	Select encoder jog on axis C	32	Yes	264
4	Select Step value 1	32	Yes	265
4	Select Step value 2	32	Yes	266
4	Select Step value 3	32	Yes	267
4	Select Step value 4	32	Yes	268
4	Select Step value 5	32	Yes	269
4	Select Step value 6	32	Yes	270
4	Select Step value 7	32	Yes	271
4	Select Step value 8	32	Yes	272
4	Select Step value 9	32	Yes	273
4	Select Step value 10	32	Yes	274
4	Set Jog mode STEP	32	Yes	275

Type	Function	FCode	IsOEM	OEMCode
4	Set Jog mode CONT	32	Yes	276
4	Code for OEMTriggers runs the macro in SetTriggerMacro	32	Yes	277
6	Reset LED	0	No	
6	Inch LED	1	No	
6	MMs LED	2	No	
6	Idle LED	3	No	
6	Start LED	4	No	
6	Pause LED	5	No	
6	Tool change LED	6	No	
6	X ref LED	7	No	
6	Y ref LED	8	No	
6	Z ref LED	9	No	
6	A ref LED	10	No	
6	B ref LED	11	No	
6	C ref LED	12	No	
6	Dwell LED	13	No	
6	Joystick enable LED	14	No	
6	Fixture LED	16	No	
6	Active 1 LED	21	No	
6	Active 2 LED	22	No	
6	Active 3 LED	23	No	
6	Active 4 LED	24	No	
6	Digitize In LED	25	No	
6	Index LED	26	No	
6	Limit OV LED	27	No	
6	X++ Limit LED	28	No	
6	X-- Limit LED	29	No	
6	X-- Home LED	30	No	
6	Y++ Limit LED	31	No	
6	Y-- Limit LED	32	No	
6	Y-- Home LED	33	No	
6	Z++ Limit LED	34	No	
6	Z-- Limit LED	35	No	
6	Z-- Home LED	36	No	
6	A++ Limit LED	37	No	
6	A-- Limit LED	38	No	
6	A-- Home LED	39	No	
6	B++ Limit LED	40	No	
6	B-- Limit LED	41	No	
6	B-- Home LED	42	No	
6	C++ Limit LED	43	No	
6	C-- Limit LED	44	No	
6	C-- Home LED	45	No	
6	Enable 1 LED	46	No	

Type	Function	FCode	IsOEM	OEMCode
6	Enable 2 LED	47	No	
6	Enable 3 LED	48	No	
6	Enable 4 LED	49	No	
6	Enable 5 LED	50	No	
6	Enable 6 LED	51	No	
6	Digitize Out LED	55	No	
6	G92 LED	56	Yes	10
6	Spindle CW LED	56	Yes	11
6	Mist LED	56	Yes	12
6	Flood LED	56	Yes	13
6	Jog mode Cont LED	56	Yes	14
6	Jog mode Incr LED	56	Yes	15
6	Mach coords warn LED	56	Yes	16
6	Feed override LED	56	Yes	17
6	Estimating LED	56	Yes	18
6	Emergency LED	56	Yes	19
6	A radius corr. LED	56	Yes	20
6	B radius corr. LED	56	Yes	21
6	B radius corr. LED	56	Yes	22
6	Software limits LED	56	Yes	23
6	Torch En LED	56	Yes	24
6	True spindle Acc LED	56	Yes	25
6	True spindle Dec LED	56	Yes	26
6	Tool Path LED	56	Yes	27
6	Tool Offset on LED	56	Yes	28
6	Part Offset on LED (always in 6.11)	56	Yes	29
6	Throttle is Slow Jog LED	56	Yes	30
6	Throttle is Feedrate LED	56	Yes	31
6	reserved	56	Yes	32
6	Auto Lim override LED	56	Yes	33
6	Override Limits/home switches LED	56	Yes	34
6	SS on Act4 LED	56	Yes	35
6	THC Arc Good LED	56	Yes	36
6	Torch Up active LED	56	Yes	37
6	Torch Down active LED	56	Yes	38
6	Feed per Min	56	Yes	39
6	Feed per Rev	56	Yes	40
6	X Scale LED	56	Yes	41
6	Y Scale LED	56	Yes	42
6	Z Scale LED	56	Yes	43
6	A Scale LED	56	Yes	44
6	B Scale LED	56	Yes	45
6	C Scale LED	56	Yes	46
6	reserved	56	Yes	47
6	Abs Coordinate Mode LED	56	Yes	48
6	Incremental Coordinate Mode LED	56	Yes	49
6	Threading Sync Mode LED (Turn)	56	Yes	50
6	Laser Probe enabled LED	56	Yes	51
6	Z-Inhibit ON LED	56	Yes	52
6	Ignore Tool Change ON LED	56	Yes	53

Type	Function	FCode	IsOEM	OEMCode
6	CV Mode ON LED	56	Yes	54
6	M30 Repeats Enabled LED	56	Yes	55
6	CV mode OFF LED	56	Yes	56
6	MPG Jog On LED	56	Yes	57
6	Engine NOT using enhanced mode LED	56	Yes	58
6	MPG Jogs X axis LED	56	Yes	59
6	MPG Jogs Y axis LED	56	Yes	60
6	MPG Jogs Z axis LED	56	Yes	61
6	MPG Jogs A axis LED	56	Yes	62
6	MPG Jogs B axis LED	56	Yes	63
6	MPG Jogs C axis LED	56	Yes	64
6	Block Delete On LED	56	Yes	65
6	Optional Stop On LED	56	Yes	66
6	Offline indicator LED	56	Yes	67
6	Threading feed related to true Spindle speed LED	56	Yes	68
6	Index signal awaited LED (Turn)	56	Yes	69
6	Anti-dive enabled LED	56	Yes	70
6	Spindle speed stable LED	56	Yes	71
6	IJ Mode is Absolute LED	56	Yes	72
6	IJ Mode is Incremental LED	56	Yes	73
6	G-code teaching file is open LED	56	Yes	74
6	Offset in effect on at least one axis	56	Yes	75
6	reserved	56	Yes	76
6	Output 4 Active LED	56	Yes	77
6	Output 5 Active LED	56	Yes	79
6	Output 6 Active LED	56	Yes	80
6	Pause Active LED	56	Yes	81
6	Tangential control Active LED	56	Yes	82
6	Single Step mode Active LED	56	Yes	83
6	Jogging enabled LED	56	Yes	84
6	CV feed enabled LED	56	Yes	85
6	Axis inhibited X LED	56	Yes	86
6	Axis inhibited Y LED	56	Yes	87
6	Axis inhibited Z LED	56	Yes	88
6	Axis inhibited A LED	56	Yes	89
6	Axis inhibited B LED	56	Yes	90
6	Axis inhibited C LED	56	Yes	91
6	Diameter mode active (Turn) LED	56	Yes	92
6	Timing signal active (Turn) LED	56	Yes	93

6.3 Signal codes

	SigName	SigInput	SigCode
	XLimitPlus	Yes	0
	XLimitMinus	Yes	1
	Xhome	Yes	2
	YLimitPlus	Yes	3
	YLimitMinus	Yes	4
	YHome	Yes	5
	ZLimitPlus	Yes	6
	ZLimitMinus	Yes	7
	ZHome	Yes	8
	ALimitPlus	Yes	9
	ALimitMinus	Yes	10
	AHome	Yes	11
	BLimitPlus	Yes	12
	BLimitMinus	Yes	13
	BHome	Yes	14
	CLimitPlus	Yes	15
	CLimitMinus	Yes	16
	CHome	Yes	17
	Activation1	Yes	18
	Activation2	Yes	19
	Activation3	Yes	20
	Activation4	Yes	21
	Digitize	Yes	22
	Index	Yes	23
	LimitOverride	Yes	24
	Emergency	Yes	25
	THCOn	Yes	26
	THCOff	Yes	27
	THCUp	Yes	28
	THCDown	Yes	29
	OEMTrigger1	Yes	29
	OEMTrigger2	Yes	30
	OEMTrigger3	Yes	31
	DigTrigger	No	0
	Enable1	No	1
	Enable2	No	2
	Enable3	No	3
	Enable4	No	4
	Enable5	No	5
	Enable6	No	6
	ExtAct1	No	7
	ExtAct2	No	8
	ExtAxt3	No	9
	ChargePump	No	10

7. Appendix 2 - Screen Layout files (.SET & .SSET)

7.1 Roles of Screen Designer and Mach2ScreenTweak

The screen layouts used by Mach2 are stored in files (.SET for full screens and SSET for simplified screens). The file to use is set by the Layout menu in Mach2 and is persistent when the program is re-loaded.

Standard Layout files are distributed with Mach2. You can modify the details of individual DROs, buttons etc. (i.e. Controls) displayed by these using Screen Designer (see chapter 8).

You can make your own layouts by combining screens from the standard layouts and ones of your own (created with Screen Designer) using the utility Mach2ScreenTweak. This utility also allows you to export a layout to a comma separated variables (CSV) file so that it can be imported into Microsoft Excel or Microsoft Access for analysis. Mach2ScreenTweak will also import a layout in CSV format so allowing you to make alterations using the Microsoft utilities or your own programs.

Warning: If you create an invalid layout then Mach2 may behave in unstable ways. Support cannot be given to problems encountered when using customised screens so you should always retain the standard layouts so that you can use them to demonstrate and report difficulties to Support.

7.2 Using Mach2ScreenTweak

7.2.1 Introduction

Mach2ScreenTweak is a utility program aimed at OEMs, dealers and expert users of Mach2Mill and Mach2Turn who make use of the "Screen Designer". It is designed to complement rather that replace that program.

Mach2ScreenTweak is really concerned with manipulating the individual screens within a Mach2 Layout. In other words it processes .SET files.

The following is a summary of its main features:

- Import screen(s) from another layout (perhaps locally designed screens imported into the standard ones)
- Delete screen(s) from a layout
- Promote/Demote a screen in a layout so its screen number as used in Screen Designer corresponds to its "position" in normal use
- Allocate F-key numbers to screen selection buttons and input new caption text for these buttons after screens have been moved or added
- Scale down or up the size of a screen or all screens in a layout to work with a different resolution of display
- Re-order the DROs in a group so that the arrow keys traverse them logically (in columns or rows) rather than in the order in which they were created in Screen Designer
- Export a comma-separated-variable file with a record for each control on the screens of a layout to analyse and document a layout.

The code is written in Visual Basic (Version 6.0) and the source is available on request for not-for-profit use. Hopefully this will encourage others to develop and distribute screens, utilities, filters, wizards etc. in the open way in which Mach2 and its predecessors evolved.

7.2.2 Installation

The utility is distributed as a .ZIP file including the executable Mach2ScreenTweak.exe, this manual and the standard Microsoft Common Dialog ActiveX controls in COMDLG32.OCX

The .ZIP also includes a sample layout (TweakSamp.set) that contains 1024 resolution screens with Caption Labels that can be used to try out appending screens from an Additional Layout.

Mach2ScreenTweak.exe can be copied into any convenient folder and run from there via a desktop or quick-start area shortcut.

If your system does not already have it, then you will need to copy COMDLG32.OCX into the folder C:\Windows\System32.

The utility can be uninstalled by deleting the .EXE file and, if specially installed, the COMDLG32.OCX

7.2.3 The main screen and its buttons

Figure 7.1 shows the main screen after loading the standard 1024.SET from Mach2Mill Release 1.00.

You will find it worthwhile "playing" before you read on in this manual. Just avoid the *Save As* operation!

The **Principal Layout** table shows the contents of the persistent screen and the 15 user screens of the layout. The buttons control what is to be done with the screens in the Principal Layout.

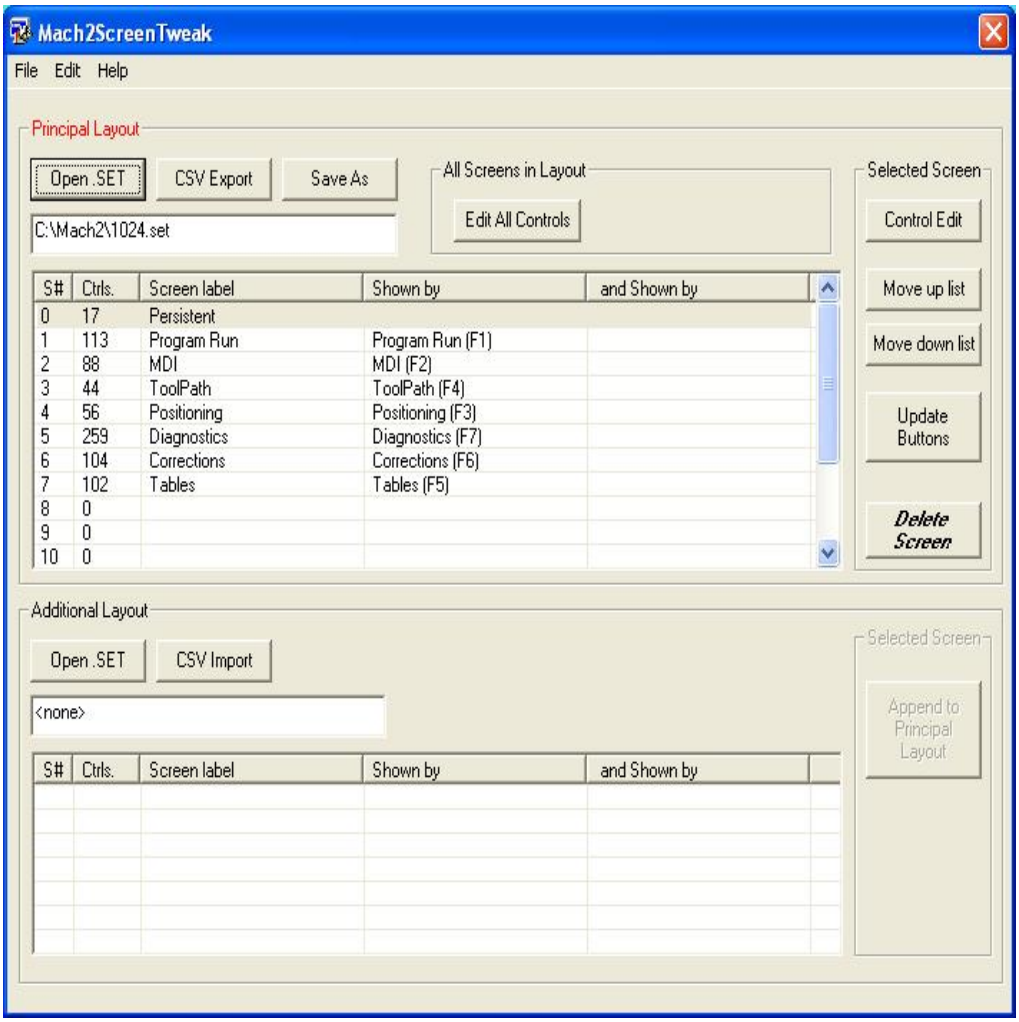


Figure 7.1 – ScreenTweak main screen

7.2.4 Manipulate all screens in layout

7.2.4.1 Open .SET

This prompts for a Layout file to open as the Principal Layout. You can select Mill (.SET) or Turn (.LSET) files to be displayed in the dialog.

When opened a copy is held within the program and the .SET file is closed so it can be used in Mach2 or Screen Designer while Mach2Tweak is in use. When a file is opened then the other buttons will be enabled.

All changes are performed on the Principal Layout. The Additional Layout (see below) is used as a source to import screens.

The Principal Layout contains information on colors to be used for controls which will be saved by Mach2ScreenTweak but is not otherwise processed by it.

This button is also implemented in the File menu.

7.2.4.2 CSV Save

This will prompt for the name of a .CSV file to contain information about each control in the Layout. The format is self-evident when studied in conjunction with the definition of Layout file to be provided in the Mach2Mill/Turn manuals.

The .CSV file can be imported into Microsoft Excel or, probably better, Microsoft Access for analysis and documenting the implemented controls in the standard or custom screens.

This button is also implemented in the File menu.

Notice that in the current release the code is not able to export the actual VB Script attached to buttons. This may be upto 64 kBytes for any one button. ScreenTweak replaces it with a descriptive message. This means that an exportedlayout is not suitable for re-import. It can however be used for its main purpose of analysing the codes and positions of controls.

7.2.4.3 Save As

The Save As button prompts for a file in which to save the modified Layout. You may of course save into the originally opened file but this is discouraged by the absence of a Save button.

This button is also implemented in the File menu.

7.2.4.4 Edit All Controls

The *Edit All Controls* button opens the dialog for manipulation of individual controls in the Layout. For full detail see section 4.

Not all operations are available, or meaningful, across the whole Layout and have to be performed on individual screens.

7.2.4.5 Edit Undo

The Edit>Undo allows you to "undo" operations performed on the Principal Layout. Undo will cause the Principal Layout to revert to its condition before the last operation or, if a sequence of several identical ones have been performed one after the other, to its condition before the first one of the sequence.

For example, suppose an MDI screen is moved up by three consecutive clicks on *Move Up* from being screen number 6. Undo would move it back from being screen 3 to screen 6. If, however, it was moved up 4 steps by *Move Up* the moved down by one *Move Down* Undo would make it move from screen 3 to screen 2, i.e. where it was before the *Move Down* which is the last operation and so is what is undone.

7.2.5 Manipulate selected screen

An individual screen can be selected by clicking its entry in the table. Screens are identified by their number (as used in Screen Designer), their Screen Caption, the captions on

button(s) that display them (typically on the Persistent screen) and as a last resort the number of controls on the screen. Full details of this identification are given in section 5 of this document.

7.2.5.1 Delete

Deletes the selected screen and moves all those below it in the list up to close the gap. *Undo* will of course undo a sequence of consecutive *Delete*s.

7.2.5.2 Move Up/Down list

Moves the selected screen up or down the list as appropriate.

This has no real significance in normal use with Mach2, but it is very convenient that the screens have the same "position" in use as when they are displayed in Screen Designer. If this is not so it is difficult to remember which screen number to select in Screen Designer. Well I can never remember the number for Mach2Mill Diagnostics!

7.2.5.3 Update Buttons

When screens are moved up and down the list then the buttons (typically on the Persistent screen but they can be anywhere) are updated to display the correct screen by=ut their shortcut keys (probably F-keys) and captions are not changed.

Update Buttons will alter any button that had a Function key shortcut in the original layout to have the F-key number corresponding to its screen number. For example screen 2 would be displayed in Mach2 on pressing F2. This is not done if the existing shortcut is not a function key. So the MDI (M) screen from older Layouts would continue to be selected by letter-M.

In addition *Update Buttons* will prompt for the new text to be on the button caption so that, for example, if it is being changed to F9 then you can enter "Short Diags (F9)"

This function is more obvious in use than it appears when explained (I hope!)

7.2.6 The Additional Layout

An additional layout (which can be the same file as the Principal one) is opened and summarised on the lower part of the screen. Its only role is to be a source of screens to be appended to the Principal Layout.

A layout stored in a CSV file, such as would be exported (see above) can also be loaded.

Notice that the current release of the import and export functions do not handle fields with " (dQuote) reliably so import should not be used.

7.2.6.1 Append to Principal Layout

An entry in the Additional Layout table is selected. When the Append to Principal Layout button is clicked then its controls are placed on the first empty screen of the Principal Layout. This appears like "appending" the screen from the Additional to the Principal Layout.

There are some point to note:

- The screens in the Additional Layout are "identified" by number, label and captions of buttons that display them in Mach2. When imported to the Principal Layout, as "new" screens, the buttons are not likely to exist and of course the screen number changes. You can use the count of controls as a guide to the correct screen having been appended. If you always use Screen Caption labels (i.e those starting with a §) in your Additional Layouts then you will know what is happening.
- It is possible that the screen appended contains buttons that select other screens in the Additional Layout (or indeed unusually the screen itself). These would not have any meaning when in the context of the Principal Layout so

they are not appended with all the other controls. This "disappearance" of controls is noted in a confirmation dialog after the append.

- Sequences of appends can be undone like any other operation on the Principal Layout.

7.2.7 Control Manipulation

Although ScreenTweak is not a replacement for Screen Designer it does have some limited features which manipulate controls rather than whole screens. These are displayed on the dialog accessed by the *Edit all Controls* and *Control Edit* buttons on the main screen. The screen is shown in figure 7.2

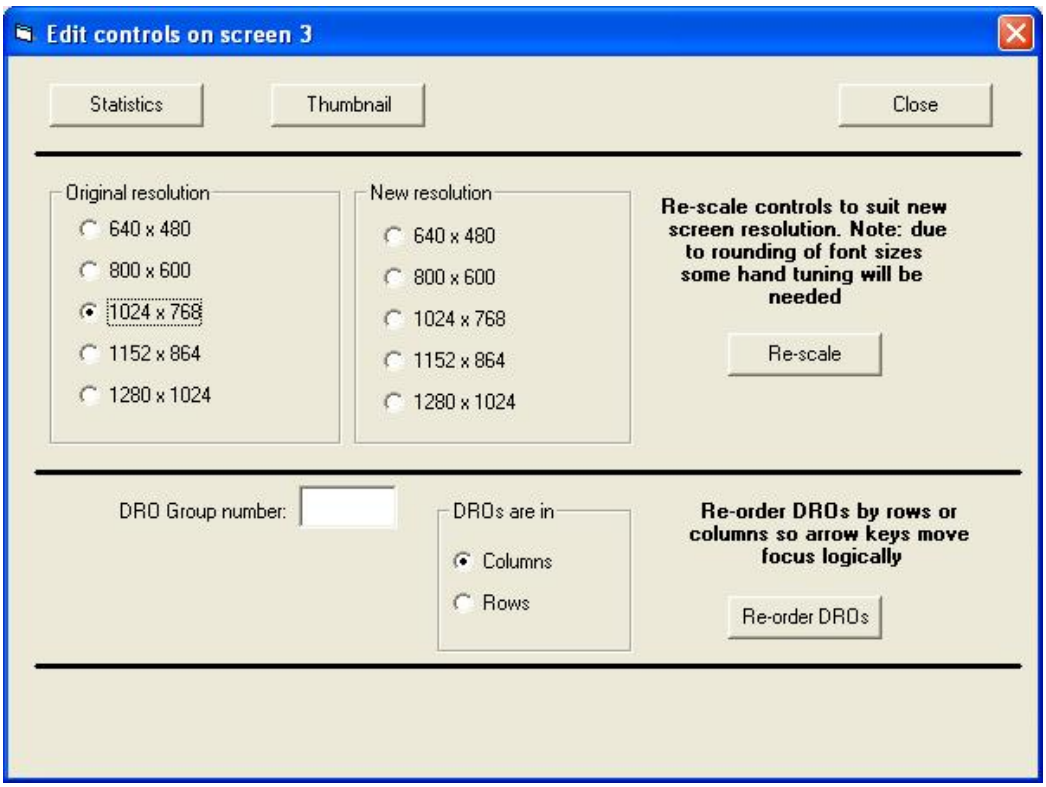


Figure 7.2 – Editing controls

7.2.7.1 Re-scale controls

The controls on an individual screen or all screens of a layout can be re-scaled to fit different resolutions of display.

Mach2ScreenTweak will try to guess the current resolution by looking at the "bottom right" control in the selected screen or the layout. If this is wrong then you can override it by clicking the appropriate radio button. Now choose the resolution of the display on which you want to use the screen or layout and click *Re-scale*.

The following should be noticed:

- The dialog automatically closes to avoid you being tempted to repeatedly Re-scale (getting smaller and smaller or bigger and bigger!)
- Because the font size of captions on buttons is fixed in Mach2 you will probably find that the caption is too big at a new resolution. Screen Designer has to be used to abbreviate captions but you have the controls reasonably placed.
  Screens with bitmap buttons do not suffer from this problem.
- Scaling down and then back up again will, of course, cause you loss of resolution due to rounding effects.
- Mach2 from 6.11 onward, optionally, supports automatic expansion of screens to fill high definition systems.

7.2.7.2 Re-order DROs

In Mach2 the arrow keys can be used to cycle through a group of DROs (e.g. all the axis DROs). The order of this cycling is the order of original creation of the DROs in Screen Designer and may not be logical.

You can re-order the DROs on a selected screen to run down a column or across the screen as a row using the radio buttons to choose the logic. Multiple columns/rows can be ordered and small errors in the placing of the DROs are ignored.

7.2.8 Screen captions and other workarounds

Screen Designer uses an admirably simple structure for Layout data. This causes some difficulties when mechanically processing Layouts. For example there is no link, other than the visual one, between labels like "Dwell" and the LED indicating it. Screens are only identified by their overall appearance rather than a "caption". This can cause problems in Mach2ScreenTweak.

7.2.8.1 Screen captions

The caption problem is overcome by use of a special "Caption Label" on screens. When you design a screen to be "tweaked" you should place somewhere on it a label that starts with the "curly-S" section symbol. For example you might have **§Compact Diagnostics.** The § is easily typed by the sequence Alt-0167 on the numeric pad (Num Lock being ON). You can also paste it from the Windows Character Map utility (in Accessories>System Tools).

The Caption Label can, of course, be visible if you have the space or can be off the bottom of the screen. **Hint:** you can work on a bit of screen not normally seen in Mach2 by temporarily switching off View>Toolbar and View>Status Bar in Screen Designer.

The main screens issued with Mach2 have Caption Labels which are "hidden" in normal use below the Reset button.

7.2.8.2 Buttons identify screens

As many existing screens will not have Caption Labels, Mach2ScreenTweak looks to see what buttons are programmed to display a screen and will use the button captions of the first two found to help identify a screen. Notice, as mentioned above, that this is little help on screens appended from the Additional Layout.

7.3 Layout file format

The screen layouts are stored in binary files and must not be opened with Microsoft Word, Notepad or the like. To view a layout in "clear text" export it using Mach2ScreenTweak.

For reference, the file structure of .SET files is given here.

7.3.1 Overall file format

The file consists of (a) a count of controls across all screens of the layout, (b) a record defining each control and (c) a record giving the colours to be used for displaying controls.

This is illustrated in figure 7.3

Int32 is a four byte signed integer. As usual it is stored in

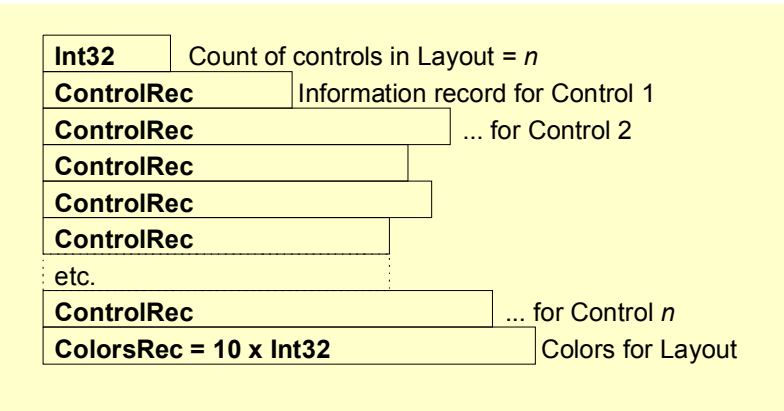


Figure 7.3 – Layout file format

the file with the least significant byte first.

The structure of *ControlRec* and *ColorsRec* is defined below. Notice that *ControlRec*s are variable in length (they contain several variable length strings) and *ColorsRec* consists of ten *Int32*s (i.e. 40 bytes)

7.3.2 ControlRec

Figure 7.4 shows the structure of each *ControlRec*. The values are integers or strings.

Integers are as defined already, some of which indicate True/False boolean values by being Zero = True, nonZero = False.

Strings follow the C Language structure of a one byte character count (unsigned so allowing 0 to 255 characters) followed by the characters, one per byte, using the Windows character set. This is illustrated in figure 7.5. Care should be taken if you try to put a " (double-quote) in a string as this character is used as the delimiter in CSV files.

Note: that although strings and hence *ControlRec*s are variable length they can be decoded by knowing the format and moving serially through the file interpreting the character counts of the strings. Direct access to records in the layout file are not possible.

Int32	Control Function
Int32	Control Type
Int32	Displayed on Screen number
C-String	Text on control
C-String	G-code text
C-String	Path of bitmap file
Int32	HorizCode
Int32	VertCode
C-String	Label
Int32	Color
Int32	OEM Code value
Int32	HotKey code
Int32	Flash flag
Int32	Red/Green flag
C-String	Format code
Int32	Tabbing group
Int32	PosX1
Int32	PosY1
Int32	PosX2
Int32	PosY2

Figure 7.4 – Structure of record for each Control

Byte8	Count of characters in string *n = 0 to 255*							
Char8	**Char8**	**Char8**	**Char8**	**Char8**	etc.	**Char8**	**Char8**	**Char8**

Figure 7.5 – Storage of character strings

Most of the fields of the ControlRec are dedicated to data for a particular type of control. The values for other controls will be "undefined". The following explains the purpose of each value. The detailed coding is to be found at the Mach Developers' Network (MachDN) site. (See Frontispiece for current link)

7.3.2.1 Screen

The number of the screen (1 to 15) on which control will appear. Value 0 will appear on all screens (i.e. persistent control)

7.3.2.2 Type

The coded type of each control as follows:

Type code	Control
1	DRO
2	Scrollbar
3	Bitmap
4	Conventional Button
5	Jogball
6	LED

7	Label
8	Bitmap Button
9	Manual Data Entry (MDI)
10	G-code window
11	Toolpath display

7.3.2.3 Function

The function is a code starting from 0 for each type of control defining what it displays or does. The code numbers can be obtained by counting (from 0) the radio buttons in the Screen Designer dialog box that sets up the relevant type of control. The following functions are extended by OEMcodes (see below):

Control type	Function code that is extended by OEMcode
DRO	12
Conventional button	32
LED	56
Bitmap button	32

The full list of functions is listed in Layouts on the MachDN site.

7.3.2.4 OEMCode

DROs, LEDs and conventional and bitmap buttons which have special purposes and are not included on the radio buttons in Screen Designer are accessed via the "OEM" radio button and an "OEM" code. This is really an extension to the Function. Public OEM codes are listed in Layouts on the MachDN site. **Hint:** To manipulate these codes in a database it is often convenient to combine Function and OEMCode by a formula such as (Function x 1000 + OEMCode) where OEMCode is set to zero if the Function does not use it.

7.3.2.5 Text

For conventional Buttons, this is the caption on the button.

7.3.2.6 GText

For Conventional and Bitmap buttons whose function is 33 this string is the G-code line to issue.

7.3.2.7 BitMapPath

For Bitmap buttons this is the filename of the .BMP file on the user's machine. (e.g. C:\Mach2\MyStart.bmp)

7.3.2.8 Horiz & Vert

For Jogball controls the code gives the axes to be jogged for horizontal and vertical "movement" of the "ball". Axis coding is X = 0, Y = 1, Z = 2, A = 3, B = 4, C = 5

7.3.2.9 Label

The string displayed in a Label control. Some values like File, Error are "intelligent" and are replaced by what they describe at runtime. Labels starting with the S mark (§) are used as screen captions.

7.3.2.10 Color

Used for LED controls. Green = 0, Red = 1, Yellow = 2

7.3.2.11 HotKey

For Conventional and Bitmap buttons this gives the scan code of the HotKey to activate the button (or 0 for no HotKey).

7.3.2.12 Flash Flag

LED will flash if this flag is non-zero

7.3.2.13 RedGreen Flag

LED will change colour from Red to Green when signal is active. Color value is ignored

7.3.2.14 Tabbing Group

For DROs this defines a group of DROs The DROs in a group are selected cyclically by the up/down arrow cursor keys. The left/right arrow keys move between different DRO groups.

7.3.2.15 PosX1, Y1, X2, Y2

These values define the top left and bottom right coordinates of the control (in pixels) on the screen. The top left corner of the screen is X = 0, Y = 0. Controls may be drawn partially or wholly off the visible screen at the current resolution without raising an error.

Color32	Background
Color32	DRO
Color32	DROSelected
Color32	MDI
Color32	MDISelected
Color32	Label
Color32	DRONumerals
Color32	MDIText
Color32	G-code
Color32	G-codeText

Figure 7.6 Colors record

7.3.3 ColorsRec

The format of the *ColorsRec* is given in figure 7.6. A Color32 value has its most significan byte = 0 and the following three bytes the unsigned intensity (i.e. range 0 to 255) of the colors Red, Green and Blue in decreasing significance. Thus Yellow (i.e. Red and Green) would be coded using hexadecimal notation as 0x00FFFF00 i.e. decimal 16776960.

The color chooser dialog in Screen Designer displays the individual RGB values for a control.

8. Appendix 3 – General utility programs

8.1 KeyGrabber

8.1.1 Overview

Chapter 8 gives a general description of the control of Mach2 including Human Interface Devices (HIDs) and keyboard emulators.

KeyGrabber is a utility written by Les Newell, to whom very many thanks are due, which translates signals from the keyboard, a keyboard emulator or the buttons, Point of View control or axes control of one or more HIDs into keycodes in Mach2's input buffer. These keycodes can then be interpreted by Mach2 as jogging hotkeys, screen button/DRO hotkeys or to activate/deactivate simulated signals. They could be entered into DROs or the MDI box if you want to implement a numeric keypad or the like..

KeyGrabber, unlike some general profiler programs, knows that it is running with Mach2 (actually it assumes responsibility for starting Mach2 or ScreenDesigner) and so will direct the HID buttons and the specially defined keycodes from the keyboard emulator or, indeed, the actual keyboard to Mach2 even if another program has the focus. Hence its name – it grabs keycodes for Mach2's use.

Many keyboard emulators were originally developed for use by the computer gaming community to interface the controls of an arcade style cabinet to the PC using the MAME standard codes. Some, however, can be configured to generate other codesets. KeyGrabber has a facility for programming the Ultimarc IPAC/2 and IPAC/4 with a set of codes different from any that can be produced by the PC keyboard. It also identifies these codes using the signal names of the IPAC.

KeyGrabber will implement Typematic repeating of keycodes from any of its possible inputs. This is useful for operating screen buttons for spindle speed control, and feed and jograte override. This Typematic hat three settings for each input. It can be set to be off, to be at one speed or to change (typically increase) speed after a defined number of repeats.

HIDs can be configured to have up to four separate meanings to each button. This is particularly useful on devices where the buttons are easy to label such as a membrane keyboard like the Saitek P8000 (aka Dash/2). The definitions are defined as pages 1 to 4. You can define arbitrary buttons on the HID to be the page selectors. Buttons such as "Fire" and "Shift" are often convenient choices.

When Mach2 is running without KeyGrabber it not only recognises the keycodes but interprets them according to whether the ***Shift Ctrl*** or ***Alt*** keys on the keyboard are simultaneously depressed. These keys are called **modifiers**. For example the standard layout uses Alt-R for the Cycle Start hotkey, the *Ctrl* key will switch Jog modes between Inc and Cont while it is depressed and *Shift* will cancel a jograte override. KeyGrabber can be used to define arbitrary HID buttons to produce the *Shift Ctrl* and *Alt* modifiers

Finally in this overview, low resolution encoders are a useful human interface to Mach2 for implementing an MPG style axis jog dial and for rotary panel controls for speed and feed override. KeyGrabber will interpret a pair of inputs to a keyboard emulator as quadrature A and B signals and translate these into a keydown/keyup of a user defined keycode for each clockwise "click" and a similar sequence for another keycode for each counterclockwise "click". These codes can operate the incremental jog hotkeys or the hotkeys for up/down buttons within Mach2.

Although, in principle re-mapping codes and the other functions of KeyGrabber are straightforward, the detail can be confusing. You may find that this appendix is easier to understand if you experiment with the software and devices as you read it.

8.1.2 Installation

8.1.2.1 The files

The KeyGrabber software is automatically installed by the Mach2 installer. You may wish to setup a desktop or quickstart bar shortcut to it. If you do not start the KeyGrabber program then its installation will have no effect on your system.

8.1.2.2 Windows compatibility

You will get the best results out of KeyGrabber by making sure that your Windows has been upgraded to include DirectX version 9. See the Microsoft website for full details of upgrades.

8.1.2.3 Running KeyGrabber and then Mach2

KeyGrabber can be run by double-clicking the KeyGrabber.exe file, in the Mach2 folder, or a shortcut to it. You will need to do this to set up how keys and buttons are to be interpreted. You can then run ScreenDesigner or Mach2 and choose the Mach2 profile by hand.

For everyday use, most Mach2 users will either use the standard Mach2Mill or Mach2Turn shortcuts that are setup when Mach2 is installed. These can easily be adapted to run KeyGrabber first and then automatically start Mach2 with the originally specified profile (i.e. Mill, Turn or indeed your own custom Mach2 profile).

Right click on the shortcut, choose Properties from the menu and modify its Target property by replacing "Mach2.exe" with "KeyGrabber.exe". Leave the text starting /p alone – this is what specifies the Mach2 XML profile to use. See chapter 5 for further details of Mach2 profiles. You may want to use a different KeyGrabber configuration for each Mach2 profile. Details of how to do this are given below.

8.1.2.4 Shortcut Icons

If you setup a shortcut on the desktop as described above then you will find that its Icon is the KeyGrabber smiley face. This is a useful reminder that you are using keygrabber. If you wish you can replace the Icon in the shortcut with one from Mach2 or indeed from any other place.

Right-click on the shortcut and choose *Properties* from the menu. Click *Change Icon…* Use *Browse* to choose the program file containing the icon you wish use for the shortcut. This is probably Mach2.exe. Then choose the icon out of the possible set in the display. Some programs may have several others only offer one.

If you have a graphics program which will create Windows Icons then you can design your own and use it by browsing to its .ico file rather than a .exe file.

8.1.3 Configuring KeyGrabber

When KeyGrabber runs for the first time it will create a file called Default.grab and this name is displayed in the caption of its window.

You can use File>Save As to save your current configuration under a different name and File>Open (or information in the shortcut Target described later) to specify that you want to use a non-default configuration.

There are four area to configure. You may not wish to use all of them and may, of course, not have the hardware to do so.

Keyboard Keys

Keyboard Encoders

Human Interface Devices (HID)

Misc Settings

The KeyGrabber window has a tab for each. Additional tabs are displayed when HID devices are enabled.

The tabs are described in the following sections:

8.1.4 Configuring Keyboard Keys

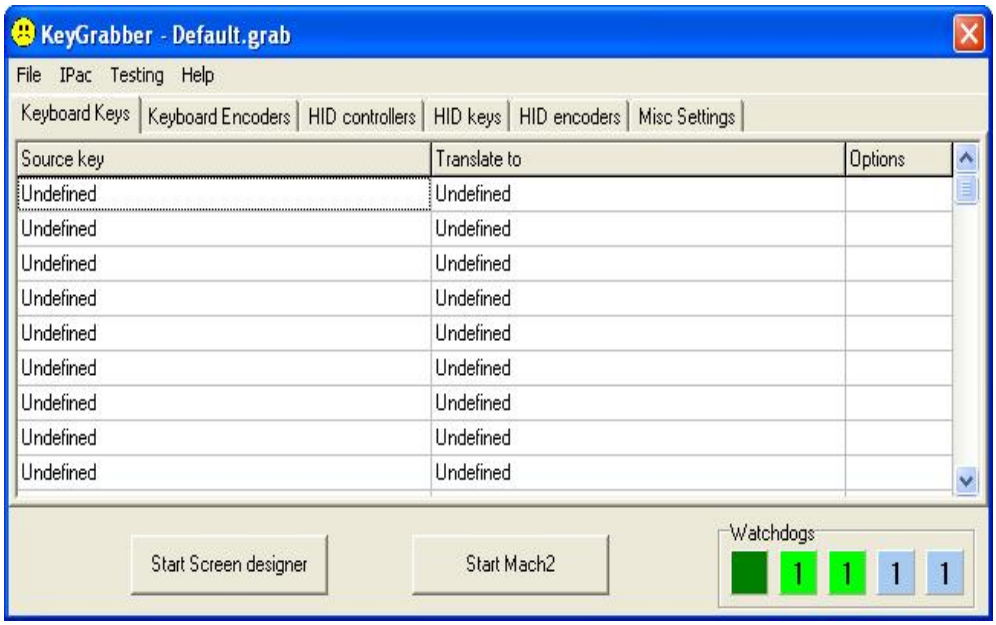


Figure 8.1 – Configure Keyboard Keys table

This tab shows a table of keycodes which may be received from the Windows keyboard port and the corresponding codes to be sent to Mach2. These codes can, of course, come from an actual keyboard or a keyboard emulator. See figure 8.1.

Either double-click or right-click and choose Define in an entry in the *Source key* column. This allows you to define the code you want to be processed. You will be presented with a dialog asking you to Press a Key. On an emulator this corresponds to making the circuit connected to the emulator input pin. If the code corresponds to one programmed into an IPAC then you will be told the terminal label and pin for this input. If the code corresponds

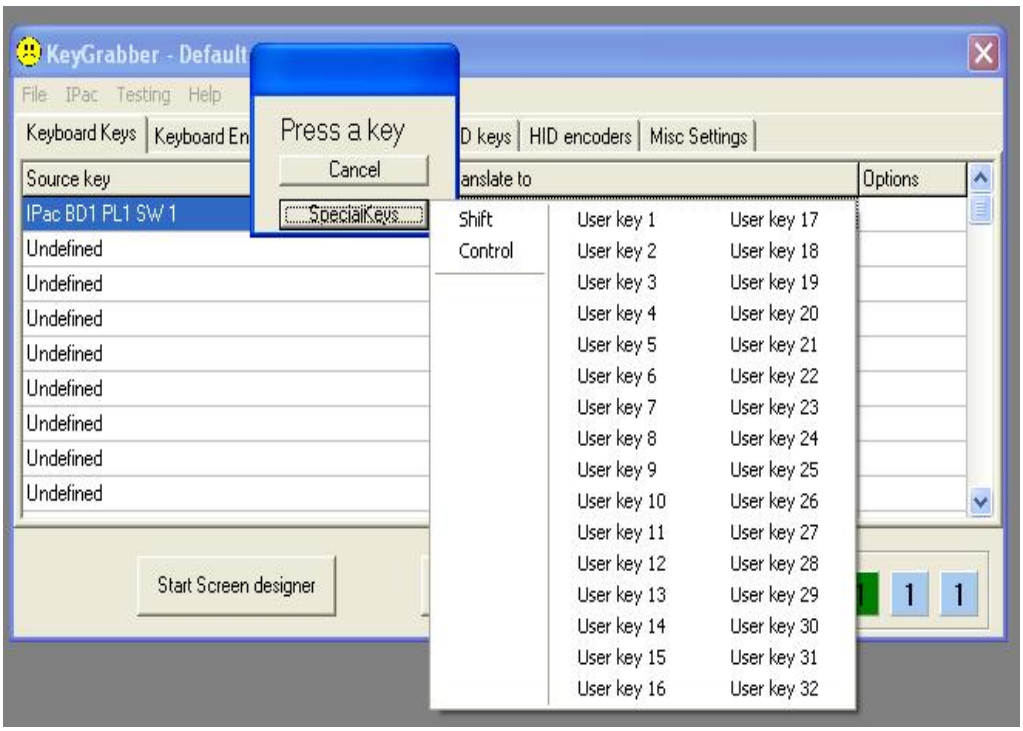


Figure 8.2 – Defining special Keys

to a key on the keyboard then you will be shown the character on the key (e.g. Q) or a description of the key (e.g. LEFT for the left cursor arrow).

Where meaningful, a default value of the code to be sent to Mach2 will have been put into the *Translate to* column. A double-click or right-click in this entry allows you to redefine it to any translated value. Good keys are the ones on the numeric keypad, function keys or multimedia special function keys. Any keyboard emulator code that does not have a default but which you do not want to translate can be entered by making the circuit connected to the emulator input pin again with the *Translate To* Press a Key box showing.

An alternative to pressing a "key" is to choose *Special Keys*. This displays the options shown in figure 8.2. You can say that the "key" which you are defining is to be the *Shift* or *Ctrl* modifier or one of 32 unique user key values. These are chosen so as not conflict with other actual codes. The *Alt* modifier is not available from the keyboard as Windows makes priority use of it.

When Mach2 (or Screen Designer) is running then any keycode that is **not** in the first column, which will be most of them on you system, will be ignored by KeyGrabber and sent by Windows to the application which has the focus when the keycode arrives. The corollary of this is that any key whose code **is** in the list will not be seen by any application other than Mach2 while Mach2 is running with the KeyGrabber. You will be warned when you try to setup keys on the standard keyboard that this will happen. It does become rather annoying but it might help you being too confused when you first "grab" A, Q, Z and M and find that Microsoft Word cannot input Monday (ondy) or Zebra (ebr) or worse you cannot get a password with any of these letters in it accepted!

The final Options column allows you to choose if typematic is to apply to the keycode and if modifiers will be sent with it. Double-click or right-click and define the cell you want to set. Figure 8.3 shows the dialog.

The parameters for Typematic which apply to all codes are defined on the Misc. Settings tab. If you select *Two stage typematic* then the *Translate to* keycode will be sent to Mach2 at the specified *First rate* until *Number at first rate* codes have been sent. Codes will then be sent at *Second rate.*

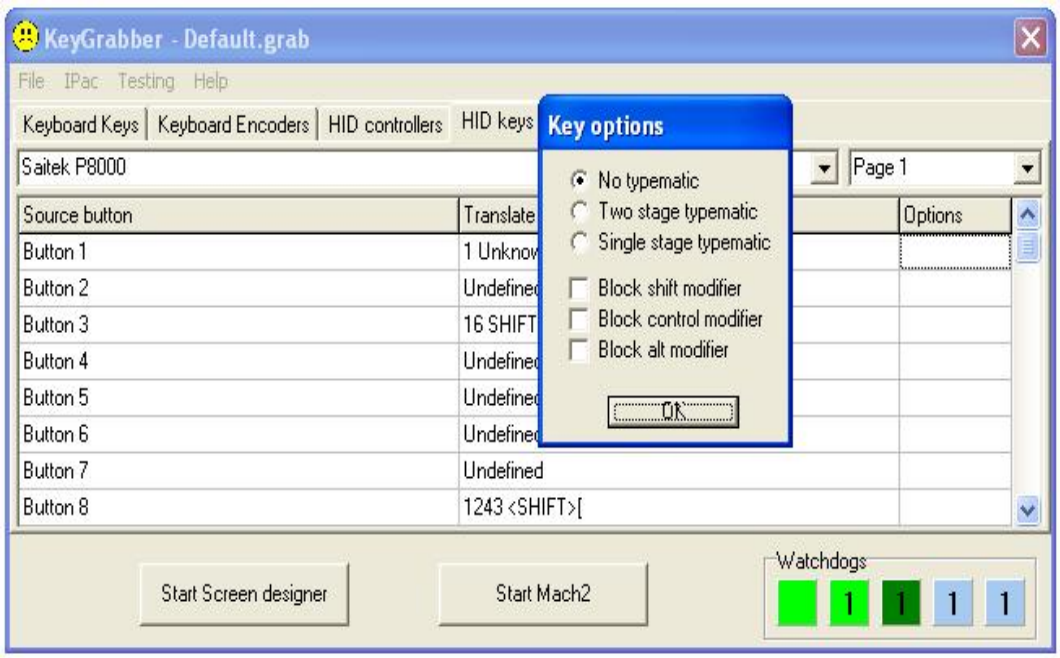


Figure 8.3 – Key translation options

The Block modifiers check boxes are provided to allow you to specify that the state of the corresponding modifier button/key is to be ignored (i.e. not sent to Mach2) for this *Source button.* This might be used for a signal coming from a switch like a Home switch where the same code is to be recognised irrespective of the *Shift*, *Ctrl* or *Alt* keys being depressed at the time.

Note: If you have a keyboard emulator which provides hardware key repeat (typematic) on its inputs then KeyGrabber will ignore this.

8.1.5 Configuring Keyboard Encoders

This tab shows the configuration of up to 16 encoders connected to keyboard emulator pins. These can be any device that produces a quadrature output, such as MPGs, digital pots etc.

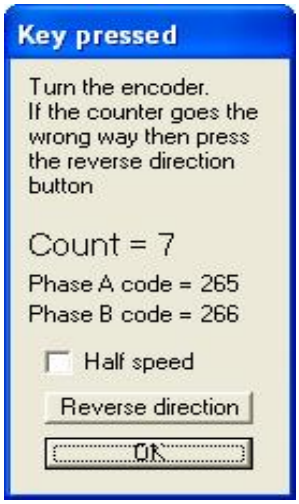


Figure 8.4 – Finding the encoder

Double-click or right-click and Define in either an A channel or B channel entry in the table. KeyGrabber will display a dialog showing an encoder count and two code values. See figure 8.4.

Rotate the encoder clockwise slowly. KeyGrabber will notice the two inputs that are changing and put their codes against the Phase A code and Phase B code labels. If the count value decreases then click the *Reverse Direction* button. If your encoder has click detents then you might find that it counts two for each click. If so check the *Half Speed* box.

Because KeyGrabber is looking for two related signals it will get very confused if more than two change during this setup process so make sure that you do not type on the keyboard or have signals or noise on other pins of the keyboard emulator.

When you are happy with the way your encoder counts then click OK.

You now have to configure the entries in the *Up Key* and *Down Key* columns of the table. Double-click (or right-click Define) the relevant cell and enter the required code (see figure 18.5).These will typically be the hotkeys that are setup in Config>Axis Hotkeys and used to jog the Mach2 axis involved or be the hotkey codes on the buttons to increment/decrement the feedrate or spindle speed.

Note 1: Some USB keyboard emulators are limited to a maximum of 8 keys simultaneously pressed. Each encoder can press up to 2 keys so do not use more than 4 encoders on this type of emulator (3 if you are using it for other functions as well). You cannot use an emulator which has repeats on these inputs.

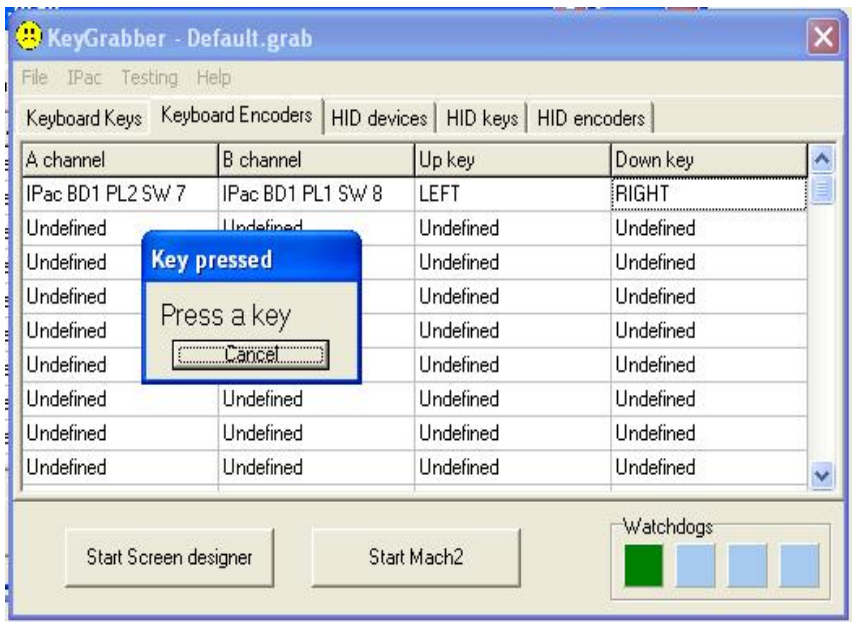


Figure 8.5 – Defining Up and Down keys

Note 2: The response speed of the keyboard system is limited so don't expect to connect up a 2000 cpr encoder and spin it fast! About 25 Hz (40 msecs for a cycle of the A or B inputs) seems to work reliably on a typical Mach2 configuration. This is 10 rpm on a 32 cpr digital potentiometer. For jogging or the like a few steps lost at speed would not really matter as you will probably be watching the work or DROs.

8.1.6 Configuring HIDs

8.1.6.1 Preparation for HIDs

Human Interface Devices (HIDs) support many different types of control. Example include an "analogue" or proportional joystick or throttle (termed by Microsoft, slightly confusingly for us, Axes), a Point of View (POV) pad or hat (like an 8 way digital joystick) and buttons.

If your HID has more than 32 buttons then you must have DirectX version 9 installed in Windows.

Plug in your HID's USB interface. Windows should recognise it with a chime or, if it is the first time you have connected, it should recognise the new device (by name) and install drivers for it. You **should not** install or run any profiler that was supplied with your HID.

You should then be able to check the basic operation of your HID using the Game Controllers option of Windows' Control Panel.

8.1.6.2 The HID Controllers tab

When you run KeyGrabber, the HID Controllers tab should show a list of all the devices you have currently plugged in. If you double-click in the *Use device?* Column this will toggle the clicked device in and out of use. Right-click or the Properties button will confirm the configuration of the device. Each device that is "in-use" has a flashing watchdog confirming (or otherwise) communication with it. A red signal indicates a problem.

When at least one HID is in-use, additional tabs are displayed.

HID Keys

HID Encoders

8.1.6.3 HID Keys

On this tab, see figure 8.6, use the drop-down list to choose the HID you wish to set up. The table on the HIDs tab is unlike the *Keyboard Keys* tab as Windows knows how many buttons a HID has (see Properties) and the table has one line for each HID key/button. You

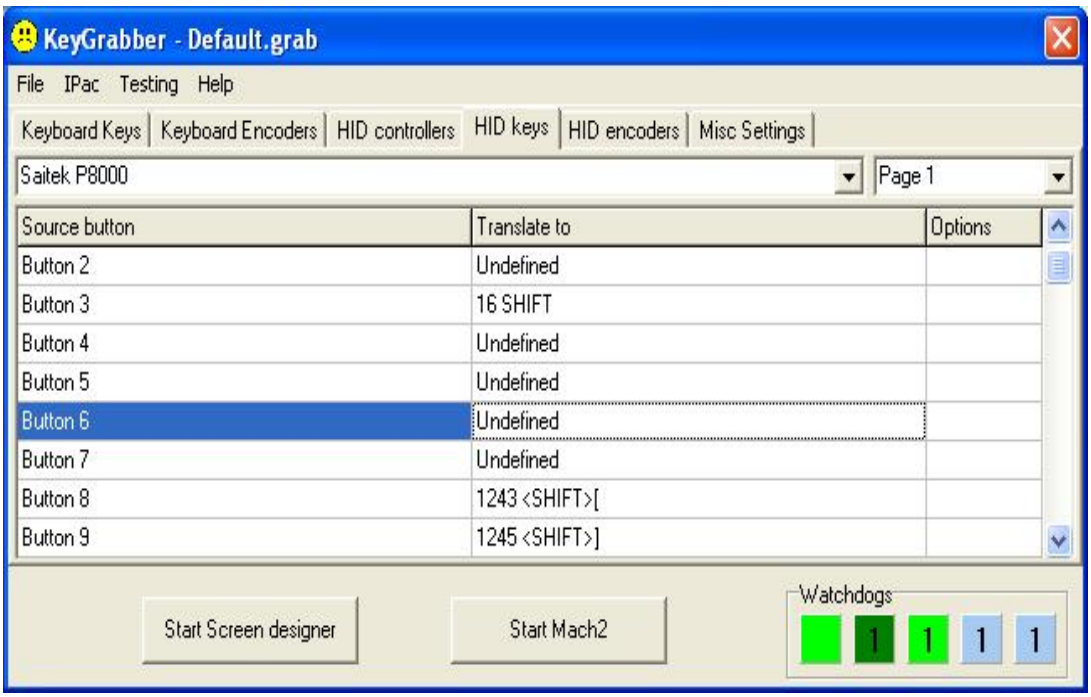


Figure 8.6 – After a HID button is pressed

select the Page you are defining by the drop-down. When Mach2 is running then Page 1 will be used for translation if no Page Select button is pressed. The other pages are selected by the buttons configured as described below. This selection is always instantaneous (i.e. it only applied while the button is depressed).

Now press a button on the HID. Its line is selected. By double-clicking or right-click and Define the*Translate to* column then the keycode that this button is to *Translate to* can be defined by pressing the appropriate key.

A key can be defined to select an alternative *Gain* for the Joystick axes. See Misc Settings for more details.

Special keys like *Shift* and *Ctrl* and *User defined* keys can be set up in exactly the same way as with the *Keyboard Keys* dialog but with the additional option to define a button as a HID Page Select code. By default when you apply a page select code this will be put on all pages. This can be overidden if required but if you do this you may make it difficult for the user to access .

Exactly the same procedure is followed to assign codes to POV controls and HID analog axis controls. They can be used for page selection. For details of how an axis position is converted to the relevant entry in the table see the section on Misc Settings below.

8.1.6.4 HID encoders

This feature is not currently implemented

8.1.6.5 Misc Settings

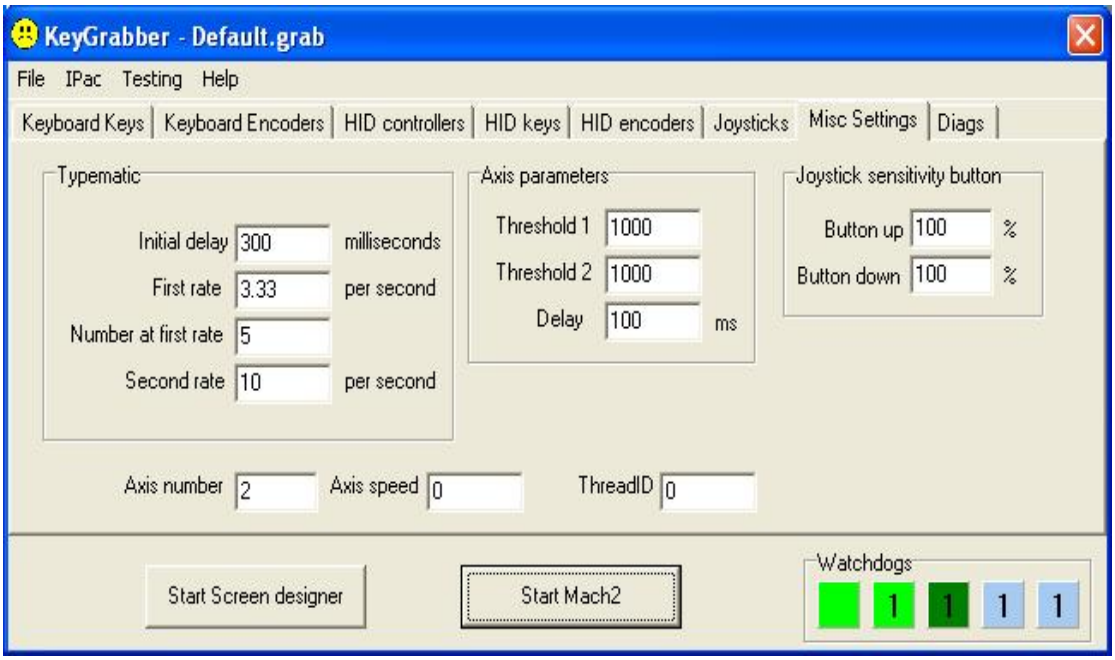


Figure 8.7 – Misc Settings dialog

Typematic settings are described in the section on Keyboard Keys Options. Notice that the same delay, rates and wait count apply to all keys with Typematic enabled whether they be on a keyboard, keyboard emulator or HID.

An Axis is a direction of analog control like a joystick or position of a throttle or steering wheel of a game controller. Each axis generates a value in the range -1000 to +1000 depending on its position. Some controllers generate axis signals from switches (rather like POV controls or indeed which double as POV controls). The switches will generally only produce the maximum positive and negative values.

The magnitude of values which trigger KeyGrabber to output a keycode are set in the dialog under Axis parameters. When the axis value numerically exceeds Threshold1 then the code in *Axis*n *Plus* or *Axis*n *Minus* will be selected. When the value exceeds Threshold2 then the code in *Axis*n *Plus2* or *Axis*n *Minus2* will be selected the other values remaining in the key-down state. This could happen very quickly and confuse Mach2 with what appear to be simultaneous key presses. The Delay time allows the two events to be separated. A time of 300 milliseconds is a useful general setting.

Axes can be joysticks and send "analog" values to Mach2. The sensitivity of these is set by their Gain. This can be altered for each individual axis at setup time (see below) but "high"

and "low" gain can be chosen dynamically for all axes using a HID key. When the configured key is pressed the Joystick Sensitivity *Button Down* value is used. If the key is not pressed then the *Button Up* value is used. This can be used to have high speed jogging for fast positioning with a button pressed and a very fine control when it is released.

8.1.7 Axes as joysticks

Axes which generate a range of values can very usefully be configured to jog axes and override feedrate, jogging speeds and spindle speed. These parameters are set up on the Joysticks tab (figure 8.8)

The dropdown selects which HID to configure. Move the control in the direction you want to configure to highlight the appropriate row of the table.

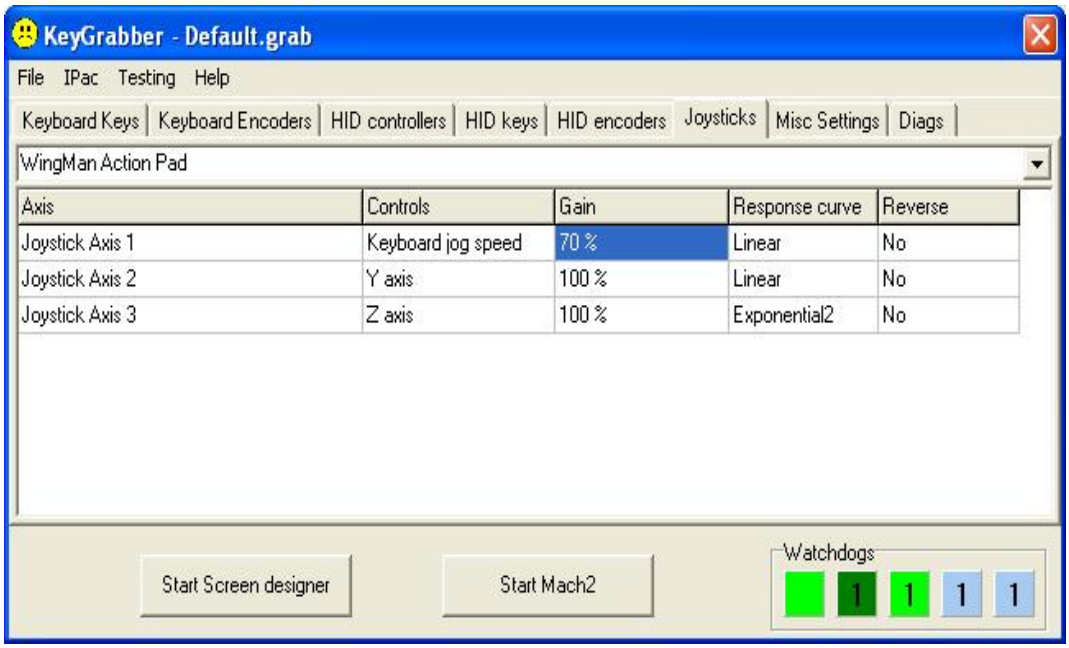


Figure 8.8 – Configuration of joysticks

The *Controls* column allows you to define the function for the axis. These should be self explanatory except for Axis Gain. This allows you to use a axis on the HID to control how sensitive the other axes will be.

The *Gain* column is used to set the relative sensitivity of each axis on the HID (e.g. how much displacement of a joystick gives a given jogging speed). This is overidden by the gain defined in Misc Settings if you have defined a *Joystick Sensitivity* button.

The *Response Curve* is the relationship between the displacement of the controld and the signal sent to Mach2. The Exponential values give a small output for positions near the center (zero) value but large values next the extremes. This can allow, for example, very sensitive control of jogging at low speeds but the possibility of very fast jogging when the joystick is held fully over.

Reverse allows positive values to be swapped for negative ones to cater for the way in which the HID is designed or mounted.

8.1.8 Multiple machines and KeyGrabber Profiles (.GRAB files)

If you use more than one machine controlled by a computer the you will probably want to configure your HID and keyboard emulator differently for each. This is done but having several .GRAB files. As mentioned above, you can Save As a configuration in a file other than Default.grab and can use File>Open to open a non-default configuration.

It would however be more convenient to automatically associate the correct .GRAB with its Mach2.

This is done by including the name of the .GRAB file to be used in the Target field of the properties of the shortcut used to run KeyGrabber. The .GRAB file name comes before /p

which specifies the Mach2 profile to use. Thus for example if the original shortcut to run Mach2 to control your lathe was:

```
C:\Mach2\Mach2 /p Mach2Turn
```

Then replacing it with a shortcut with a Target of:

```
C:\Mach2\KeyGrabber.exe Turn.grab /p Mach2Turn
```

Would run Mach2 with the Mach2 profile Mach2Turn.xml after setting up KeyGrabber with the Turn.grab configuration.

Hint: If you have several shortcuts for different KeyGrabber and Mach2 profiles then you are very likely to want to setup different icons for each of them as described above.

8.1.9 Keyboard Emulator programming

Your keyboard emulator will probably come programmed with the MAME standard gaming keycodes and may also allow you to program a set of your own codes. In this case you can of course use any values but given that KeyGrabber is going to translate them it is probably best to program in a set of codes that do not clash with those that the actual keyboard can generate. Values decimal 264 upwards are probably suitable.

KeyGrabber will automatically program the Ultimarc IPAC range of emulators with such a range of unused keycodes using the IPAC menu. You choose the IPAC configuration and the speed at which you wish to program. Start fast and slow down if the programming fails.

Beware: To program an IPAC you need to set the MAME/Alt jumper correctly. Programming an IPAC will loose any existing codes stored in its "Alt" memory. The MAME codes are not overwritten.

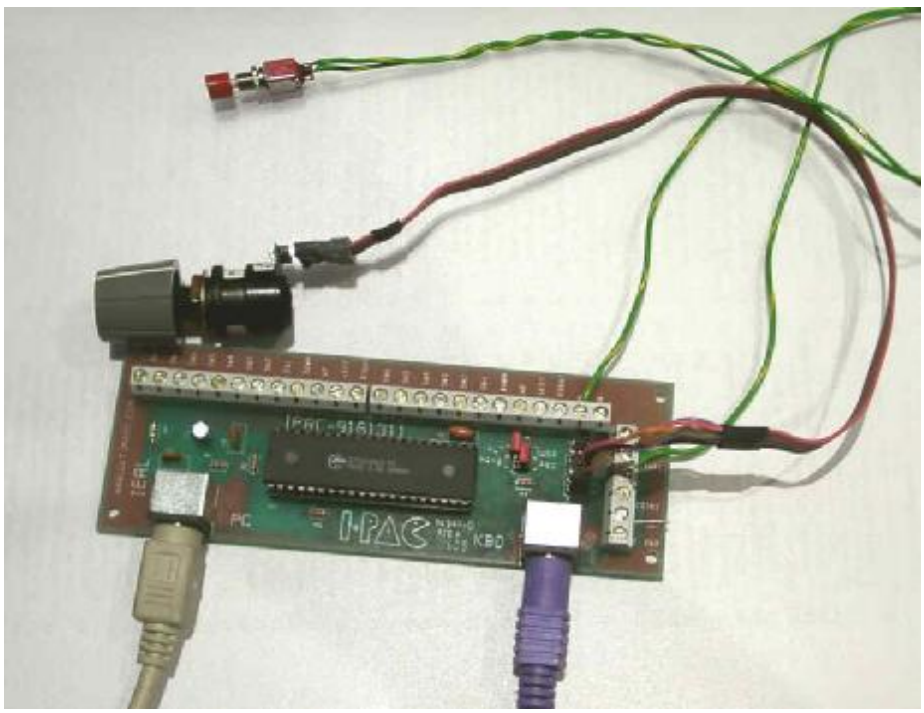


Figure 8.9 – Ultimarc IPAC/2 keyboard emulator

8.1.10 Testing and troubleshooting

The Testing menu allows you to display a dialog which will give you the code and mnemonic for any key seen by KeyGrabber. This will test the actual keyboard, switches on a keyboard emulator, encoders on an emulator, and the buttons, POV and axes on an enabled HID. You can use this to check that your wiring is correct and to discover the code that the keyboard or emulator is actually sending to KeyGrabber.

Note: For IPAC users. Notice that the IPAC supports LED outputs indicating the state of the Num Lock, Caps Lock and Scroll Lock signals of the PC keyboard. These outputs are

shared with three input terminals. These effectively cannot be used, because they are pulled lo, if the corresponding LED is on.

Windows 2000 command prompt windows intercept keyboard messages before they reach KeyGrabber. If you have a command prompt window or a DOS application in the foreground then KeyGrabber will not work from the keyboard.

9. Revision history

Rev 6.11-A6	13 November 2004	Detailed correction of typos and grammar
Rev 6.11-A5	12 November 2004	Wizard writing sections added Documentation of VB Script calls updated.
Rev 6.11- A3	10 November 2004	Manual split for original Mach2Mill document

10. Index

Hint: Where there is a choice, most index entries are made using the name of a thing (e.g. Axis drive) rather than an action (e.g. Tuning) so you will get better results thinking about the part on which you want information. Thus looking for "Axis drives - tuning" will give better results than looking for "Tuning - axis drives". For important information both entries will probably appear.

If you have difficulty because you tried to look something up and the index entry was missing, please take a moment to e-mail support@artofcnc.ca with a note of (a) the words you were looking up and (b) where in the manual you found the information you wanted - assuming you did!

H

I

K

L

M

N

O

P

Q

S

T

U

V

W