

Advanced Graphics Functions for the PicoMite Firmware

User Manual

Revision 0

For updates to this manual and more details on MMBasic go to

<http://geoffg.net/picomite.html>

and <http://mmbasic.com>

PicoMite Advanced Graphics

This manual applies to the versions of the PicoMite firmware that support LCD display panels (ie, not VGA or HDMI video).

The PicoMite Advanced Graphics incorporates a suite of advanced graphic controls for touch sensitive LCD panels. These include on screen switches, buttons, indicator lights, keyboard, etc. MMBasic will draw the control and animate it (i.e. a switch will appear to depress when touched). All that the BASIC program needs to do is invoke a single command to specify the basic details of the control.

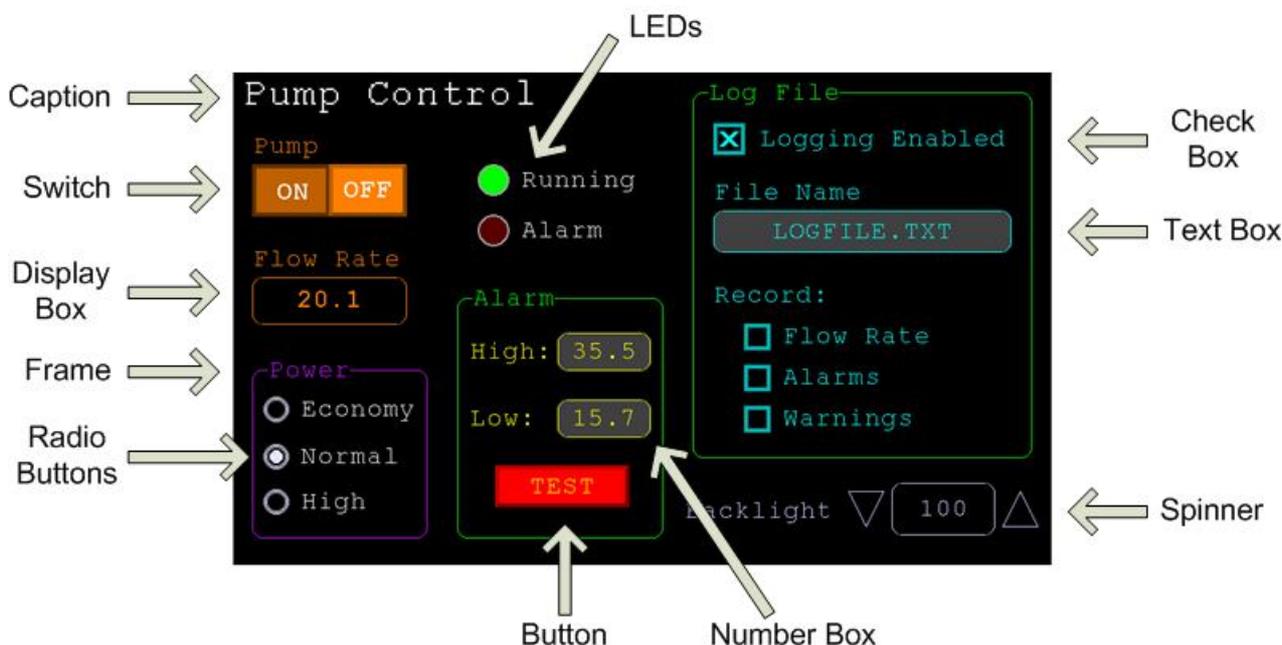
To use the GUI controls the memory required for the GUI controls must be allocated first by using the command `OPTION GUI CONTROLS`. Typically you would use the command like this:

```
OPTION GUI CONTROLS 75
```

This will set the maximum number of controls that you can define to 75. This option is permanent (i.e. it will be remembered on power down). By default the maximum number of controls is set to zero and in this case the GUI features will not be available and no memory will be used.

Defining Controls

These are some of the advanced GUI controls that you can use:



Each control has a reference number called '#ref' in the description of the control. This can be any number between 1 and the upper limit set by the `OPTION CONTROL` command. This reference number is used to identify a control. For example, a check box can be created with a reference number of #10:

```
GUI CHECKBOX #10, "Test", 100, 100, 50, rgb(BLUE)
```

Once created the user can check and uncheck the box using the touch feature of the LCD panel without the running BASIC program being involved. When needed the program can determine the check box value by using its reference number in the `CtrlVal()` function:

```
IF CtrlVal(#10) THEN ..
```

The # character is optional but serves to remind the programmer that this is not an ordinary number.

In the following commands any arguments that are in italic font (e.g. *Width*, *Height*) are optional and if not specified will take the value of the previous command that did specify them. This means for example, that a number of radio buttons with the same size and colour can be specified with only the first button having to list all the details. Note that with the colour specification this is different to the basic drawing commands which default to the last `COLOUR` command.

All strings used in GUI controls and the `MsgBox` can display multiple lines by using the tilde character (~) to separate each line in the string. For example, a push button's caption can be "ALARM~TEST" and this would be displayed as two lines. For all controls the font used for the caption will be whatever is set with the `FONT` command and the colours will be whatever was set by the last `COLOUR` command.

If the display is capable of transparent text these commands will allow the use of -1 for the background colour. This means that the text is drawn over the background with the background image showing through the gaps in the letters.

The advanced graphics controls are:

Frame

GUI FRAME #ref, caption\$, StartX, StartY, Width, Height, Colour

This will draw a frame which is a box with round corners and a caption. A frame does not respond to touch but is useful when a group of controls need to be visually brought together. It can also be used to surround a group of radio buttons and MMBasic will arrange for the radio buttons surrounded by the frame to be exclusive – that is, when one radio button is selected any other button that was selected and within the frame will be deselected.

LED

GUI LED #ref, caption\$, CenterX, CenterY, Diameter, Colour

This will draw an indicator light (it looks like a panel mounted LED). When its value is set to one it will be illuminated and when it is set to zero it will be off (a dull version of its colour attribute). The LED can be made to flash by setting its value to the number of milliseconds that it should remain on before turning off.

The caption will be drawn to the right of the LED and will use the colours set by the COLOUR command. The LED control is not animated when touched but its reference number can be found using TOUCH(REF) and TOUCH(LASTREF) in the touch interrupts and any required animation can be done in MMBasic.

Check Box

GUI CHECKBOX #ref, caption\$, StartX, StartY, Size, Colour

This will draw a check box which is a small box with a caption. Both the height and width are specified with the 'Size' parameter. When touched an X will be drawn inside the box to indicate that this option has been selected and the control's value will be set to 1. When touched a second time the check mark will be removed and the control's value will be zero. The caption will be drawn to the right of the Check Box and will use the colours set by the COLOUR command.

Push Button

GUI BUTTON #ref, caption\$, StartX, StartY, Width, Height, FColour, BColour

This will draw a momentary button which is a square switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When the touch is removed the value will revert to zero. Caption can be a single string with two captions separated by a vertical bar (|) character (e.g. "UP|DOWN"). When the button is up the first string will be used and when pressed the second will be used.

Switch

GUI SWITCH #ref, caption\$, StartX, StartY, Width, Height, FColour, BColour

This will draw a latching switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When touched a second time the switch will be released and the value will revert to zero. Caption can be a single string with two captions separated by a | character (e.g. "ON|OFF"). When this is used the switch will appear to be a toggle switch with each half of the caption used to label each half of the toggle switch.

Radio Button

GUI RADIO #ref, caption\$, CenterX, CenterY, Radius, Colour

This will draw a radio button with a caption. When touched the centre of the button will be illuminated to indicate that this option has been selected and the control's value will be 1. When another radio button is selected the mark on this button will be removed and its value will be zero. Radio buttons are grouped together when surrounded by a frame and when one button in the group is selected all others in the group will be deselected. If a frame is not used all buttons on the screen will be grouped together.

The caption will be drawn to the right of the button and will use the colours set by the COLOUR command.

Display Box

GUI DISPLAYBOX #ref, StartX, StartY, Width, Height, FColour, BColour

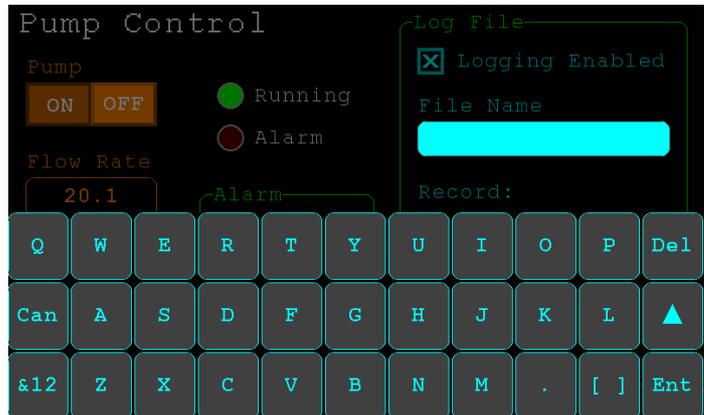
This will draw a box with rounded corners. Any string can be displayed in the box by using the CtrlVal(r) = command. This is useful for displaying text, numbers and messages. This control is not animated when touched but its reference number can be found using TOUCH(REF) and TOUCH(LASTREF) in the touch interrupts and any required animation can be done in MMBasic.

Text Box

GUI TEXTBOX #ref, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a QWERTY keyboard will appear on the screen as shown on the right. Using this virtual keyboard any text can be entered into the box including upper/lower case letters, numbers and any other characters in the ASCII character set. The new text will replace any text previously in the box.

Ent is the enter key, Can is the cancel key and will close the text box and return it to its original state, the triangle is the shift key, the [] key will insert a space and the &12 key will select an alternate key selection with numbers and special characters (there are two sets of special characters and the shift key will switch between them).



The displayed string can be set by assigning a string to the box using the CtrlVal(r) = command. The value of the control can also be set to a string starting with two hash characters (##) and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return an empty string. When a key is pressed the ghost text will vanish and be replaced with the entered text.

MMBasic will try to position the virtual keyboard on the screen to not obscure the text box that caused it to appear. A pen down interrupt will be generated just before the keyboard is deployed and a key up interrupt will be generated when the Enter or Cancel keys are touched and the keyboard is hidden.

If necessary the virtual keyboard can be dismissed by the program (same as touching the cancel button) with the command: GUI TEXTBOX CANCEL.

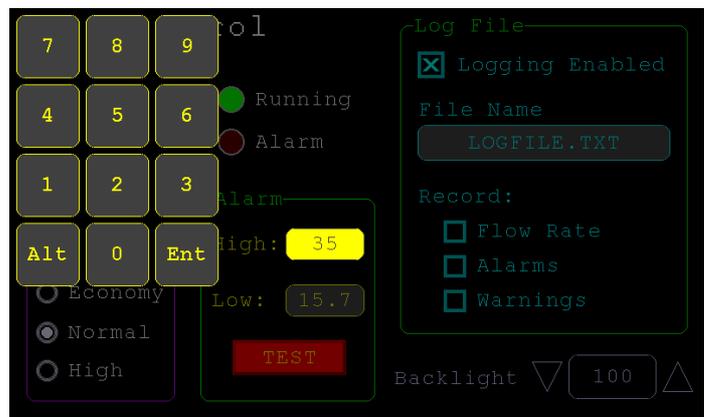
Number Box

GUI NUMBERBOX #ref, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a numeric keypad will appear on the screen as shown on the right. Using this virtual keypad any number can be entered into the box including a floating point number in exponential format. The new number will replace the number previously in the box.

The Alt key will select an alternative key selection and the other special keys are the same as with the text box.

The displayed number can also be set by assigning a number (float or integer) to the box using the CtrlVal(r) = command.



Similar to the Text Box, the value of the control can set to a literal string with two leading hash characters (e.g. "##Hint") and in that case the string (without the leading two characters) will be displayed in the box with reduced brightness. Reading this will return zero and when a key is pressed the ghost text will vanish.

MMBasic will try to position the virtual keypad on the screen to not obscure the number box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will

be generated when the Enter key is touched and the keypad is hidden. Also, when the Enter key is touched the entered text will be evaluated as a number and the NUMBERBOX control redrawn to display this number.

Formatted Number Box

GUI FORMATBOX #ref, Format, StartX, StartY, Width, Height, FColour, BColour

This will draw a box with rounded corners. When the box is touched a numeric keypad will appear similar to a Number Box. The difference is that the Formatted Number Box will require the user to enter numbers according to a specific format for dates, time, etc. Invalid keys on the keypad will be disabled and the user will be guided in their entry with guide text. This means that the programmer can be assured that the entry made by the user will always be in a fixed format.

The type of entry is controlled by the 'Format' argument as follows:

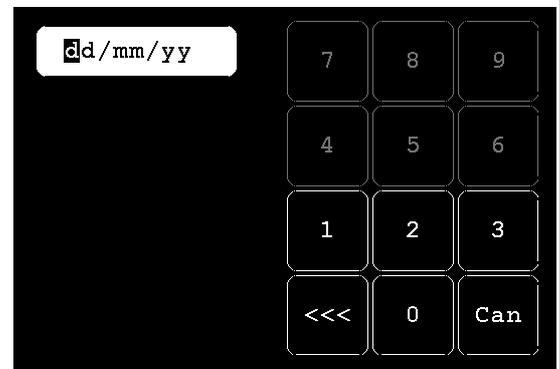
DATE1	Date in UK/Aust/NZ format (dd/mm/yy)
DATE2	Date in USA format (mm/dd/yy)
DATE3	Date in international format (yyyy/mm/dd)
TIME1	Time in 24 hour notation (hh:mm)
TIME2	Time in 24 hour notation with seconds (hh:mm:ss)
TIME3	Time in 12 hour notation (hh:mm AM/PM)
TIME4	Time in 12 hour notation with seconds (hh:mm:ss AM/PM)
DATETIME1	Both date (UK fmt) and time (12 hour) (dd/mm/yy hh:mm AM/PM)
DATETIME2	Both date (UK fmt) and time (24 hour) (dd/mm/yy hh:mm)
DATETIME3	Both date (USA fmt) and time (12 hour) (mm/dd/yy hh:mm AM/PM)
DATETIME4	Both date (USA fmt) and time (24 hour) (mm/dd/yy hh:mm)
LAT1	Latitude in degrees, minutes and seconds (d° mm' ss" N/S)
LAT2	Latitude with seconds to one decimal place (dd° mm' ss.s" N/S)
LONG1	Longitude in degrees, minutes and seconds (ddd° mm' ss" E/W)
LONG2	Longitude with seconds to one decimal place (ddd° mm' ss.s" E/W)
ANGLE1	Angle in degrees and minutes (ddd° mm')

For example:

```
GUI FORMATBOX #1, DATE1, 300, 150, 200, 50
```

would create a data entry box and when it is touched a keypad will appear as shown on the right. Note that:

- The display box is filled with a guide string to prompt the user as to the data required.
- Because the day of the month can only start with a digit from 0 to 3 all other keys are disabled. This also happens with other numbers that have a limited range.
- The value of the control retrieved via CtrlVal(#1) is a string. As an example, if the user entered the date for the 8th of May 2020 the returned string would be "08/05/20" (i.e. the UK/Aust/NZ format as specified by DATE1).



The value of the control can be pulled apart using the string functions or, in some cases, the string can be used directly. For example, if using the above format box to get a date from the user the Raspberry Pi Pico's internal clock could then be directly set as follows:

```
DATE$ = CtrlVal(#1)
```

The RTC SETTIME command will accept a single string argument in the format of dd/mm/yy hh:mm so similarly the RTC time could be set as follows if the formatted box used DATETIME2 for 'Format':

```
RTC SETTIME CtrlVal(#1)
```

You can use the USA style DATETIME4 to get the date/time. In that case you would use this to set the RTC:

```
RTC SETTIME MID$(CtrlVal(#1),4,3) + LEFT$(CtrlVal(#1),2) + RIGHT$((CtrlVal(#1),9)
```

MMBasic will try to position the virtual keypad on the screen so as to not obscure the format box that caused it to appear. A pen down interrupt will be generated when the keypad is deployed and a key up interrupt will be generated when all the required data has been entered and the keypad is hidden.

Spin Box

```
GUI SPINBOX #ref, StartX, StartY, Width, Height, FColour, BColour, Step,  
Minimum, Maximum
```

This will draw a box with up/down icons on either end. When these icons are touched the number in the box will be incremented or decremented by the 'StepValue', holding down the touch will repeat at a fast rate. 'Minimum' and 'Maximum' set a limit on the value that can be entered.

'StepValue', 'Minimum' and 'Maximum' are optional and if not specified 'StepValue' will be 1 and there will be no limit on the number entered. A pen down interrupt will be generated every time up/down is touched or when automatic repeat occurs.

Caption

```
GUI CAPTION #ref, text$, StartX, StartY, Alignment, FColour, BColour
```

This will draw a text string on the screen. It is similar to the basic drawing command TEXT, the difference being that MMBasic will automatically dim this control if a keyboard or number pad is displayed.

'Alignment' is zero to three characters (a string expression or variable is also allowed) where the first letter is the horizontal alignment around X and can be L, C or R for LEFT, CENTER, RIGHT and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE, BOTTOM.

A third character can be used to indicate the rotation of the text. This can be 'N' for normal orientation, 'V' for vertical text with each character under the previous running from top to bottom, 'I' the text will be inverted (i.e. upside down), 'U' the text will be rotated counter clockwise by 90° and 'D' the text will be rotated clockwise by 90°. The default alignment is left/top with no rotation.

If the colours are not specified this control will use the colours set by the COLOUR command.

Circular Gauge

```
GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max,  
nbrdec, units$, c1, ta, c2, tb, c3, tc, c4
```

This will define a graphical circular analog gauge with a digital display in the centre showing the value and units. If specified the gauge will be coloured to provide a graphical indication of the signal level (e.g. green for OK, yellow for warning, etc).

'StartX' and 'StartY' are the coordinates of the centre of the gauge while 'Radius' is the distance from the centre to the outer edge.

'min' is the value associated with the minimum value of the gauge and 'max' is the maximum value. When CtrlVal() is used to assign a value (floating point or integer) to the gauge the analogue portion of the gauge will be drawn to a length proportional to the range between 'min' and 'max'.

At the same time the digital value will be drawn in the centre of the gauge using the current font settings (set with the FONT command). 'nbrdec' specifies the number of decimal places to be used in this display. Under the digital value the 'units\$' will be displayed (this can be skipped or a zero length string used if not required).

Normally the analog graph is drawn using the colour specified in 'Fcolour' however a multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change.

Specifically, 'c1' is the colour to be used for values up to 'ta'. 'c2' is the colour to be used for values between 'ta' and 'tb', 'c3' is used for values between 'tb' and 'tc' and c4 is used for values above 'tc'. Colours and thresholds not required can be left off then list. For example, for a two colour gauge only 'c1', 'ta' and 'c2' need to be specified.

When colours and thresholds are specified the background of the gauge will be drawn with a dull version of the gauge colour at that level ("ghost colouring") so that the user can appreciate how close to the various thresholds the actual value is. Also the digital value displayed in the centre will also change to the colour specified by the current value.

If only one colour is required for the whole analogue graph it can be specified by just using 'c1' and leaving all the following parameters off.



Bar Gauge

GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4

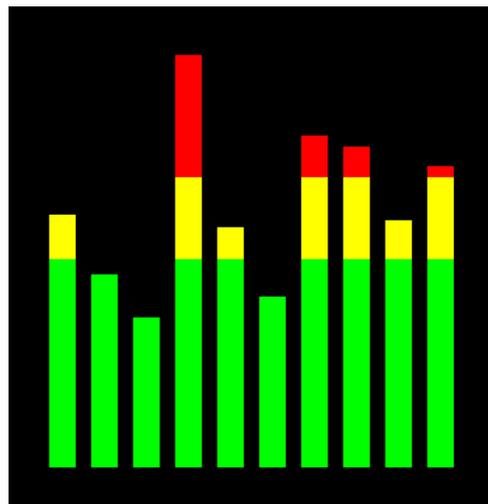
This will define either a horizontal or vertical bar gauge. The gauge can be coloured to provide a graphical indication of the signal level (e.g. green for OK, yellow for warning, etc) and many bar graphs can be packed close together so that a number of values can be displayed simultaneously using a small amount of screen space (as shown in the image which consists of ten bar gauges).

If the width is less than the height the bar gauge will be drawn vertically with the analogue graph growing from the bottom towards the top. Otherwise, if the width is more than the height, it will be drawn horizontally with the analogue graph growing from the left towards the right. In both cases 'StartX' and 'StartY' reference the top left coordinate of the bar graph while 'width' is the horizontal width and 'height' the vertical height.

The bar graph does not have a digital display of its value but other than that the parameters are the same as for the circular gauge (described above).

'min' and 'max' specify the range of values for the bar and, if specified, 'c1' to 'c4' and 'ta' to 'tc' specify the colours and thresholds for the analogue bar image. Note that unlike the circular bar gauge a "ghost image" of the colours is not shown in the background.

As with the circular gauge, if only one colour is required for the whole gauge it can be specified by just using 'c1' and leaving all the following parameters off.



Area

GUI AREA #ref, StartX, StartY, Width, Height

This will define an invisible area of the screen that is sensitive to touch and will set TOUCH(REF) and TOUCH(LASTREF) accordingly when touched or released. It can be used as the basis for creating a custom control which is defined and managed by the BASIC program.

Interacting with Controls

Using the following commands and functions the characteristics of the on screen controls can be changed and their value retrieved.

- = CTRLVAL(#ref)
This is a function that will return the current value of a control. For controls like check boxes or switches it will be the number one (true) indicating that the control has been selected by the user or zero (false) if not. For controls that hold a number (e.g. a SPINBOX) the value will be the number (normally a floating point number). For controls that hold a string (e.g. TEXTBOX) the value will be a string. For example:

```
PRINT "The number in the spin box is: " CTRLVAL(#10)
```
- CTRLVAL(#ref) =
This command will set the value of a control. For off/on controls like check boxes it will override any touch input and can be used to depress/release switches, tick/untick check boxes, etc. A value of zero is off or unchecked and non zero will turn the control on. For a LED it will cause the LED to be illuminated or turned off. It can also be used to set the initial value of spin boxes, text boxes, etc. For example:

```
CTRLVAL(#10) = 12.4
```
- GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]
This will change the foreground colour of the specified controls to 'colour'. This is especially handy for a LED which can change colour.
- GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]
This will change the background colour of the specified controls to 'colour'.
- = TOUCH(DOWN)
This is a function that will return true if the screen is currently being touched (faster than TOUCH(X or Y)).

- = TOUCH(UP)
This is a function that will return true if the screen is NOT being touched (faster than TOUCH(X or Y))
- = TOUCH(LASTX)
This is a function that will return the X coordinate of the last location that was touched.
- = TOUCH(LASTY)
This is a function that will return the Y coordinate of the last location that was touched.
- = TOUCH(REF)
This is a function that will return the reference number of the control currently being touched. If no control is currently being touched it will return zero.
- = TOUCH(LASTREF)
This is a function that will return the reference number of the control that was last touched.
- GUI DISABLE #ref1 [, #ref2, #ref3, etc]
This will disable the controls in the list. Disabled controls do not respond to touch and will be displayed dimmed. The keyword ALL can be used as the argument and that will disable all controls on the currently displayed page. For example:
GUI DISABLE ALL
- GUI ENABLE #ref1 [, #ref2, #ref3, etc]
This will undo the effects of GUI DISABLE and restore the controls in the list to normal operation. The keyword ALL can be used as the argument for all controls on the currently displayed page.
- GUI HIDE #ref1 [, #ref2, #ref3, etc]
This will hide the controls in the list. Hidden controls will not respond to touch and will be replaced on the screen with the current background colour. The keyword ALL can be used as the argument.
- GUI SHOW #ref1 [, #ref2, #ref3, etc]
This will undo the effects of GUI HIDE and restore the controls in the list to being visible and capable of normal operation. The keyword ALL can be used as the argument for all controls.
- GUI DELETE #ref1 [, #ref2, #ref3, etc]
This will delete the controls in the list. This includes removing the image of the control from the screen using the current background colour and freeing the memory used by the control. The keyword ALL can be used as the argument and that will cause all controls to be deleted.

Audio Feedback

When using the GUI controls the OPTION TOUCH command can optionally specify an output pin that is used to drive a Piezo buzzer. The firmware will use this to make a click sound when a touch sensitive control is touched and this provides an audio feedback to the user.

The BASIC program can also access this buzzer using the following command:

```
GUI BEEP msec
```

Where 'msec' is the number of milliseconds that the beeper should be driven. A time of 3ms produces a click while 100ms produces a short beep.

MsgBox()

The MsgBox() function will display a message box on the screen and wait for user input. While the message box is displayed all controls will be disabled so that the message box has the complete focus.

The syntax is:

```
r = MsgBox(message$, button1$ [, button2$ [, button3$ [, button4$]]])
```

All arguments are strings. 'message\$' is the message to display. This can contain one or more tilde characters (~) which indicate a line break. Up to 10 lines can be displayed inside the box. 'button1\$' is the caption for the first button, 'button2\$' is the caption for the second button, etc. At least one button must be specified and four is the maximum. Any buttons not included in the argument list will not be displayed.

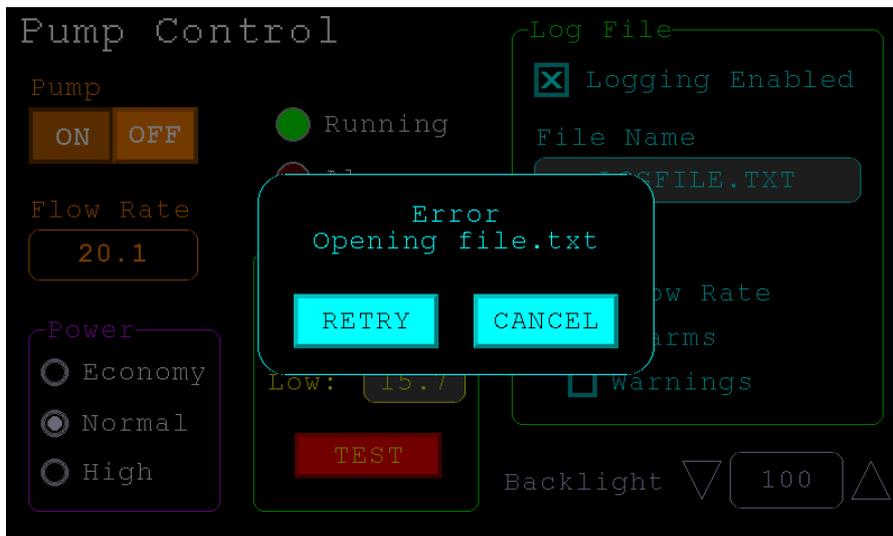
The font used will be the default font set using the FONT command and the colours used will be the defaults set by the COLOUR command. The box will be automatically sized taking into account the dimensions of the default font, the number of lines to display and the number of buttons specified.

When the user touches a button the message box will erase itself, restore the display (e.g. re enable all controls) and return the number of the button that was touched (the first button will return 1, the second 2, etc). Note that, unlike all other GUI controls the BASIC program will stop running while the message box is displayed, interrupts however will be honoured and acted upon.

To illustrate the usage of a message box will the following program fragment will attempt to open a file and if an error occurs the program will display an error message using the MsgBox() function. The message has two lines and the box has two buttons for retry and cancel.

```
Do
  On Error Skip
  Open "file.txt" For Input As #1
  If MM.ErrNo <> 0 Then
    if MsgBox("Error~Opening file.txt","RETRY","CANCEL") = 2 Then Exit Sub
  EndIf
Loop While MM.ErrNo <> 0
```

This would be the result if the file "file.txt" did not exist:



Advanced Graphics Programming Techniques

When programming using the advanced GUI commands there are a number of hints and techniques to consider that will make it easier to develop and maintain your program.

The User Should Be In Control

Traditional character based programs are normally in control of the interaction with the user. For example, the program may display a menu and prompt the user to select an action. If the user selects an invalid option the program would display an error message and display the menu options again.

However graphical based programs such as that created using the advanced GUI commands are different. Usually the program just starts running doing what it normally does (e.g. control temperature, speed, etc) and it is the user's job to select and change parameters without being prompted. This is a different way of programming and is often hard for the traditional programmer to get used to this different technique.

As an example, consider a program that is to control a cutting device. The traditional program would prompt the user for the speed and cutting time. When both have been entered the program would prompt to start the cutting cycle. However, a graphical based program would display two number boxes where the user could enter the speed and time along with a run button. The number boxes could be filled with default values and the run button would be disabled if the user entered an invalid speed or time. When the run button is touched the cutting cycle would start.

A good example of this type of graphical interface is the dialogue box used on a Windows/IOS/Android computer to set the time and date. It displays a number of boxes where the user can enter the date/time along with an OK button that tells the program to accept the data entered. At no time is the user forced to make a selection from a menu. Also, the current time/date is already displayed in the entry boxes so the user can accept them as the default if they wanted to do so.

If you need some inspiration as to how your graphical program should look and feel check your nearest GUI based operating system to see how they operate.

Program Structure

Typically a program would start by defining the controls (which MMBasic will draw on the screen), then it would set the defaults and finally it would drop into a continuous loop where it would do whatever job it was design to do. For example, take the case of a simple controller for a motor where the user could select the speed and cause the motor to run by pressing an on screen button.

To implement this function the program would look something like this:

```
GUI CAPTION #1, "Speed (rpm)", 200, 50           ' label the number box
GUI NUMBERBOX #2, 200, 100, 150, 40            ' define and draw the number box
CtrlVal(#2) = 100                               ' default value for the speed
GUI BUTTON #3, "RUN", 200, 350, 0, RGB(red)    ' define and draw the RUN button

DO                                               ' this runs in a loop forever
  IF CtrlVal(#3)<10 OR CtrlVal(#3)>200 THEN      ' check the speed setting
    GUI DISABLE #3                             ' disable RUN if it is invalid
  ELSE                                          ' otherwise
    GUI ENABLE #3                              ' enable the RUN button
  ENDIF

  IF CtrlVal(#3) = 1 THEN                      ' if the button is pressed
    SetMotorSpeed CtrlVal(#2)                  ' make the motor run
  ELSE                                          ' otherwise the button is up
    SetMotorSpeed 0                            ' therefore set motor speed to zero
  ENDIF
LOOP
```

Note that the user is not prompted to do anything; the program just sits in a loop reacting to the changes that the user has made to the controls (i.e. the user is in control).

Disable Invalid Controls

As in the above example, disabling a control will prevent a user from using it and MMBasic will redraw it in a dull colour to indicate that it is not available. This is the equivalent of an error message in a traditional text

based program and is more user friendly than popping up a message box which must be dismissed before anything else can be done.

There are many times that a control could be invalid, for example when an input is not ready or simply when an option or action does not apply. Later, when the control becomes valid you can use the GUI ENABLE command to return it to use. Another example is when a GUI NUMBERBOX keypad is displayed MMBasic will automatically disable all other controls on the screen so that it is obvious to the user where their input is required.

Disabling a control still leaves it on the screen, so that the user knows that it is there but it will be dimmed and will not respond to touch. Not responding to touch also means that the user cannot change it and an interrupt will not be generated when it is touched. This is handy for you the programmer because you do not have to check if the control is valid before acting on it.

Use Constants for Control Reference Numbers

The advanced controls use a reference number to identify the control. To make it easy to read and maintain your program you should define these numbers as constants with easy to recognise names.

For example, in the following program fragment MAIN_SWITCH is defined as a constant and this constant is used wherever the reference number for that control is required:

```
CONST MAIN_SWITCH = 5
CONST ALARM_LED = 6
' ...
GUI SWITCH MAIN_SWITCH, "ON|OFF", 330, 50, 140, 50, RGB(white), RGB(blue)
GUI LED ALARM_LED, 215, 220,30, RGB(red)
' ...
IF CtrlVal(MAIN_SWITCH) = 0 THEN ... ' for example turn the pump off
IF ALARM THEN CtrlVal(ALARM_LED) = 1
```

It is much easier to remember what MAIN_SWITCH does than remembering what control the number 5 refers to. Also, when you have a lot of controls it is much easier to renumber the controls when all their numbers are defined at the one place at the start of the program.

The reference number must be a number between 1 and the value set with the OPTION CONTROL command. Increasing this number will consume more RAM and decreasing it will recover some RAM.

The Main Program Is Still Running

It is important to realise that your main BASIC program is still running while the user is interacting with the GUI controls. For example, it will continue running even while a user holds down an on screen switch and it will keep running while the virtual keyboard is displayed as a result of touching a TEXTBOX control.

For this reason your main program should not arbitrarily update touch sensitive screen controls, because they might change the on screen image while the user is using them (with undefined results). Normally when a BASIC program using GUI controls starts it will initialise controls such as a SPINBOX, NUMBERBOX and TEXTBOX to some initial value but from then on the main program should just read the value of these controls – it is the responsibility of the user to change these, not your program.

However, if you do want to change the value of such an on-screen control you need some mechanism to prevent both the program and the user making a change at the same time. One method is to set a flag within the key down interrupt to indicate that the control should not be updated during this time. This flag can then be cleared in the key up interrupt to allow the main program to resume updating the control.

Note that this discussion only applies to controls that respond to touch. Controls such as CAPTION and LED can be changed at any time by the main program and often are.

Touch Interrupts

When the Advanced GUI controls are activated (by setting the number of GUI controls to a non-zero number) the GUI INTERRUPT command is used to setup a touch interrupt.

The syntax is:

```
GUI INTERRUPT down [, up]
```

Where 'down' is the subroutine to call when a touch down has been detected. And optionally 'up' is the subroutine to call when the touch has been lifted from the screen ('up' and 'down' can point to the same subroutine if required).

As an example, the following program will print out the X and Y coordinates of any touch on the screen:

```
GUI INTERRUPT MyInt
DO : LOOP

SUB MyInt
    PRINT TOUCH(X) TOUCH(Y)
END SUB
```

Specifying the number zero (single digit) as the argument will cancel both up and down interrupts. ie:

```
GUI INTERRUPT 0
```

Use Interrupts and SELECT CASE Statements

Everything that happens on a screen using the advanced controls will be signalled by an interrupt, either touch down or touch up. So, if you want to do something immediately when a control is changed, you should do it in an interrupt. Mostly you will be interested in when the touch (or pen) is down but in some cases you might also want to know when it is released.

Because the interrupt is triggered when the pen touches any control or part of the screen you need to discover what control was being touched. This is best performed using the TOUCH(REF) function and the SELECT CASE statement.

For example, in the following fragment the subroutine PenDown will be called when there is a touch and the function TOUCH(REF) will return the reference number of the control being touched. Using the SELECT CASE the alarm LED will be turned on or off depending on which button is touched. The action could be any number of things like raising an I/O pin to turn on a physical siren or printing a message on the console.

```
CONST ALARM_ON = 15
CONST ALARM_OFF = 16
CONST ALARM_LED = 33
GUI INTERRUPT PenDown
' ...
GUI BUTTON ALARM_ON, "ALARM ON ", 330, 50, 140, 50, RGB(white), RGB(blue)
GUI BUTTON ALARM_OFF, "ALARM OFF ", 330, 150, 140, 50, RGB(white), RGB(blue)
GUI LED ALARM_LED, 215, 220, 30, RGB(red)
' ...
DO : LOOP      ' the main program is doing something

' this sub is called when touch is detected
SUB PenDown
    SELECT CASE TOUCH(REF)
        CASE ALARM_ON
            CtrlVal(ALARM_LED) = 1
        CASE ALARM_OFF
            CtrlVal(ALARM_LED) = 0
    END SELECT
END SUB
```

The SELECT CASE can also test for other controls and perform whatever actions are required for them in their own section of the CASE statement.

The important point is that the maintenance of the controls (e.g. responding to the buttons and turning the alarm LED off or on) is done automatically without the main program being involved – it can continue doing something useful like calculating some control response, etc.

Touch Up Interrupt

In most cases you can process all user input in the touch down interrupt. But there are exceptions and a typical example is when you need to change the characteristics of the control that is being touched. For example, if you wanted to change the foreground colour of a button from white to red when it is down. When it is returned to the up state the colour should revert to white.

Setting the colour on the touch down is easy. Just define a touch down interrupt and change the colour in the interrupt when that control is touched. However, to return the colour to white you need to detect when the touch has been removed from the control (i.e. touch up). This can be done with a touch up interrupt.

To specify a touch up interrupt you add the name of the subroutine for this interrupt to the end of the GUI INTERRUPT command. For example:

```
GUI INTERRUPT IntTouchDown, IntTouchUp
```

Within the touch up subroutine you can use the same structure as in the touch down sub but you need to find the reference number of the last control that was touched. This is because the touch has already left the screen and no control is currently being touched. To get the number of the last control touched you need to use the function TOUCH(LASTREF)

The following example shows how you could meet the above requirement and implement both a touch down and a touch up interrupt:

```
SUB IntTouchDown
  SELECT CASE TOUCH(REF)
    CASE ButtonRef
      GUI FCOLOUR RGB(RED), ButtonRef
  END SELECT
END SUB

SUB IntTouchUp
  SELECT CASE TOUCH(LASTREF)
    CASE ButtonRef
      GUI FCOLOUR RGB(WHITE), ButtonRef
  END SELECT
END SUB
```

Keep Interrupts Very Short

Because a touch interrupt indicates a request by the user it is tempting to do some extensive programming within an interrupt. For example, if the touch indicates that the user wants to send a message to another controller it sounds logical to put all that code within the interrupt. But this is not a good idea because MMBasic cannot do anything else while your program is processing the interrupt and sending a message could take many milliseconds.

Instead your program should update a global variable to indicate what is requested and leave the actual execution to the main program. For example, if the user did touch the "send a message" button your program could simply set a global variable to true. Then the main program can monitor this variable and if it changes perform the logic and communications required to satisfy the request.

Remember the commandment "Thou shalt not hang around in an interrupt".

Multiple Screens

Your program might need a number of screens with differing controls on each screen. This could be implemented by deleting the old controls and creating new ones when the screen is switched. But another way to do this is to use the GUI SETUP and GUI PAGE commands. These allow you to organise the controls onto pages and with one simple command you can switch pages. All controls on the old page will be automatically hidden and controls on the new page will be automatically shown.

To allocate controls to a page you use the GUI SETUP nn command where nn refers to the page in the range of 1 to 32. When you have used this command any newly created controls will be assigned to that page. You can use GUI SETUP as many times that you want. For example, in the program fragment below the first two controls will be assigned to page 1, the second to page 2, etc.

```
GUI SETUP 1
GUI Caption #1, "Flow Rate", 20, 170,, RGB(brown),0
GUI Displaybox #2, 20, 200, 150, 45

GUI SETUP 2
GUI Caption #3, "High:", 232, 260, LT, RGB(yellow)
GUI Numberbox #4, 318, 6,90, 12, RGB(yellow), RGB(64,64,64)

GUI SETUP 3
GUI Checkbox #5, "Alarms", 500, 285, 25
GUI Checkbox #6, "Warnings", 500, 325, 25
```

By default only the controls setup as page 1 will be displayed and the others will be hidden.

To switch the screen to page 3 all you need do is use the command GUI PAGE 3. This will cause controls #1 and #2 to be automatically hidden and controls #5 and #6 to be displayed. Similarly GUI PAGE 2 will hide all except #3 and #4 which will be displayed.

You can specify multiple pages to display at the one time, for example, GUI PAGE 1, 3 will display both pages 1 and 3 while hiding page 2. This can be useful if you have a set of controls that must be visible all the time. For

example, GUI PAGE 1, 2 and GUI PAGE 1, 3 will leave the controls on page 1 visible while the others are switched on and off.

It is perfectly legal for a program to modify controls on other pages even though they are not displayed at the time. This includes changing the value and colours as well as disabling or hiding them. When the display is switched to their page the controls will be displayed with their new attributes.

It is possible to place the GUI PAGE commands in the touch down interrupt so that pressing a certain control or part of the screen will switch to another page.

Note that when ALL is used for the list of controls in commands such as GUI ENABLE ALL this only refers to the controls on the pages that are currently selected for display. Controls on other pages will be unaffected.

All programs start with the equivalent of the commands GUI SETUP 1 and GUI PAGE 1 in force. This means that if the GUI SETUP and GUI PAGE commands are not used the program will run as you would expect with all controls displayed.

A typical usage of the GUI PAGE command is shown below.

Two buttons (which are always displayed) allow the user to select between the first page and the second page. The switch is done in the touch down interrupt.

```
GUI SETUP 1
GUI Button #10, "SELECT PAGE ONE", 50, 100, 150, 30, RGB(yellow), RGB(blue)
GUI Button #11, "SELECT PAGE TWO", 50, 140, 150, 30, RGB(yellow), RGB(blue)
```

```
GUI SETUP 2
GUI Caption #1, "Displaying First Page", 20, 20
```

```
GUI SETUP 3
GUI Caption #2, "Displaying Second Page", 20, 50
```

```
Page 1, 2
```

```
GUI INTERRUPT TouchDown
```

```
Do
```

```
    ' the main program loop
```

```
Loop
```

```
Sub TouchDown
```

```
    If Touch(REF) = 10 Then GUI Page 1, 2
```

```
    If Touch(REF) = 11 Then GUI Page 1, 3
```

```
End Sub
```

Multiple Interrupts

With many screen pages the interrupt subroutine could get long and complicated. To work around that it is possible to have multiple interrupt subroutines and switch dynamically between them as you wish (normally after switching pages). This is done by redefining the current interrupt routines using the GUI INTERRUPT command.

For example, this program fragment uses different interrupt routines for pages 4 and 5 and they are specified immediately after switching the pages.

```
GUI PAGE 4
GUI INTERRUPT P4keydown, P4keyup
..
GUI PAGE 5
GUI INTERRUPT P5keydown, P5keyup
..
```

Using Basic Drawing Commands

There are two types of objects that can be on the screen. These are the GUI controls and the basic drawing objects (PIXEL, LINE, TEXT, etc). Mixing the two on the screen is not a good idea because MMBasic does not track the position of the basic drawing objects and they can clash with the GUI controls.

As a result, unless you are prepared to do some extra programming, you should use either the GUI controls or the basic drawing objects – but you should not use both. So, for example, do not use TEXT but use GUI CAPTION instead. If you only use GUI controls MMBasic will manage the screen for you including erasing and redrawing it as required, for example when a virtual keyboard is displayed.

Note that the CLS command (used to clear the screen) will automatically set any GUI controls on the screen to hidden (i.e. it does a GUI HIDE ALL before clearing the screen).

The main problem with mixing basic graphics and GUI controls occurs with the Text Box, Formatted Box and Number Box controls which display a virtual keyboard. This can erase any basic graphics and MMBasic will not know to restore them when the keyboard is removed. If you want to mix basic graphics with GUI controls you should:

- Intercept the touch down interrupt for the Text Box, Formatted Box and Number Box controls as that indicates that a virtual keyboard is about to be displayed and that will give you the opportunity to redraw your non GUI basic graphics in anticipation of this event (for example, draw them in a dimmed state to appear as if they are disabled).
- Intercept the touch up interrupt for the same controls as that indicates that the virtual keyboard has been removed and you could then redraw any non GUI graphics in their original state.

Overlapping Controls

Controls can be defined to overlap on the display, this mostly occurs with GUI AREA which, as an example, you might want to capture a touch that was intended for (say) a GUI BUTTON. This will allow you to create your own animation for the button rather than that provided by MMBasic. In this case the control that you wish to respond to the touch (i.e. GUI AREA) should have a lower reference number (i.e. #ref) than the control that it is covering (i.e. the GUI BUTTON). This is because when the screen is touched MMBasic will check the current list of active controls starting with control number 1 and working upwards. When a match is made MMBasic will take the appropriate action and terminate the search. This results in the lower numbered control effectively masking out a higher numbered control covering the same screen area as the touched location.

Options, Commands and Functions

Options

	Permanent?	
OPTION GUI CONTROLS NbrOfGUIControls	✓	<p>Specifies the maximum number of GUI controls that can be defined. Each control uses 52 bytes and the total memory used must be rounded up to the next 2048 byte multiple. For example, specifying 70 controls will use 4KB of RAM.</p> <p>By default the number of GUI controls is set to zero so this option must be used before any GUI controls are defined.</p>

Commands

CTRLVAL(#ref) =	<p>This command will set the value of an advanced control. '#ref' is the control's reference number.</p> <p>For off/on controls like check boxes it will override any touch input and can be used to depress/release switches, tick/untick check boxes, etc. A value of zero is off or unchecked and non-zero will turn the control on. For a LED it will cause the LED to be illuminated or turned off. It can also be used to set the initial value of spin boxes, text boxes, etc.</p> <p>For example:</p> <pre style="text-align: center;">CTRLVAL(#10) = 12.4</pre> <p>All controls expect to be assigned a number (float or integer) except Frame, Caption, Display Box, Text Box and Format Box which expect a string.</p>
GUI AREA #ref, startX, startY, width, height	<p>This will define an invisible area of the screen that is sensitive to touch and will generate touch down and touch up interrupts. It can be used as the basis for creating custom controls which are defined and managed by the program.</p> <p>'#ref' is the control's reference number. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions.</p>
GUI BCOLOUR colour, #ref1 [, #ref2, #ref3, etc]	<p>This will change the background colour of the specified controls to 'colour' which is an RGB value for the drawing colour.</p> <p>'#ref' is the control's reference number.</p>
GUI BARGAUGE #ref, StartX, StartY, width, height, FColour, BColour, min, max, c1, ta, c2, tb, c3, tc, c4	<p>Define either a horizontal or vertical analogue bar gauge.</p> <p>'#ref' is the control's reference number.</p> <p>'StartX' and 'StartY' are the top left coordinates of the bar while 'width' is the horizontal width and 'height' the vertical height. If the width is less than the height the bar gauge will be drawn vertically with the graph growing from the bottom towards the top. Otherwise it will be drawn horizontally with the graph growing from the left towards the right.</p> <p>'Fcolour' is the colour used for the gauge while 'Bcolour' is the background colour. 'min' is the minimum value of the gauge and 'max' is the maximum value (both floating point).</p> <p>A multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change.</p> <p>'width', 'height', 'FColour', 'BColour', 'min' and 'max' are optional and will default to the values used in the previous definition of a GUI BARGAUGE.</p> <p>'c1', 'ta', 'c2', 'tb', 'c3', 'tc' and 'c4' are optional and if not specified the gauge will</p>

	<p>use less colours. If all are omitted the gauge will be drawn using 'Fcolour'. The section <i>Advanced Graphics</i> has a more detailed description.</p>
<p>GUI BUTTON #ref, caption\$, startX, startY, width, height [, FColour] [, BColour]</p>	<p>This will draw a momentary button which is a square switch with the caption on its face. When touched the visual image of the button will appear to be depressed and the control's value will be 1. When the touch is removed the value will revert to zero.</p> <p>#ref' is the control's reference number. 'caption\$' is the string to display on the face of the button. It can be a single string with two captions separated by a character (e.g. "UP DOWN"). When the button is up the first string will be used and when pressed the second will be used.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours.</p> <p>'width', 'height', FColour' and 'BColour' are optional and default to that used in previous controls or set with the COLOUR command.</p>
<p>GUI CAPTION #ref, text\$, startX, startY [,align\$] [, FColour] [, BColour]</p>	<p>This will draw a text string on the screen.</p> <p>#ref' is the control's reference number.</p> <p>'text\$' is the string to display. 'startX' and 'startY' are the top left coordinates.</p> <p>'align\$' is zero to three characters (a string expression or variable is also allowed) where the first letter is the horizontal alignment around X and can be L, C or R for LEFT, CENTER, RIGHT and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE, BOTTOM. A third character can be used in the string to indicate the rotation of the text. This can be 'N' for normal orientation, 'V' for vertical text with each character under the previous running from top to bottom, 'I' the text will be inverted (i.e. upside down), 'U' the text will be rotated counter clockwise by 90° and 'D' the text will be rotated clockwise by 90°. The default alignment is left/top with no rotation.</p> <p>'FColour' and 'BColour' are RGB values for the foreground and background colours. On a display that supports transparent text BColour can be -1 which means that the background will show through the gaps in the characters.</p> <p>FColour' and 'BColour' are optional and default to the colours set by the COLOUR command.</p>
<p>GUI CHECKBOX #ref, caption\$, startX, startY [, size] [, colour]</p>	<p>This will draw a check box which is a small box with a caption. When touched an X will be drawn inside the box to indicate that this option has been selected and the control's value will be set to 1. When touched a second time the check mark will be removed and the control's value will be zero.</p> <p>#ref' is the control's reference number.</p> <p>The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command.</p> <p>'startX' and 'startY' are the top left coordinates while 'size' set the height and width (the box is square). 'colour' is an RGB value for the drawing colour. 'size' and 'colour' are optional and default to that used in previous controls.</p>
<p>GUI DELETE #ref1 [,#ref2, #ref3, etc] or GUI DELETE ALL</p>	<p>This will delete the controls in the list. This includes removing the image of the control from the screen using the current background colour and freeing the memory used by the control.</p> <p>#ref' is the control's reference number. The keyword ALL can be used as the argument and that will delete all controls.</p>
<p>GUI DISABLE #ref1 [,#ref2, #ref3, etc] or GUI DISABLE ALL</p>	<p>This will disable the controls in the list. Disabled controls do not respond to touch and will be displayed dimmed.</p> <p>#ref' is the control's reference number. The keyword ALL can be used as the argument and that will disable all controls.</p> <p>GUI ENABLE can be used to restore the controls.</p>

<p>GUI DISPLAYBOX #ref, startX, startY, width, height, FColour, BColour</p>	<p>This will draw a box with rounded corners that can be used to display a string '#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. ' FColour ' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>Any text can be displayed in the box by using the CtrlVal(r) = command. This is useful for displaying text, numbers and messages.</p> <p>This control does not respond to touch.</p>
<p>GUI ENABLE #ref1 [,#ref2, #ref3, etc] or GUI ENABLE ALL</p>	<p>This will undo the effects of GUI DISABLE and restore the control(s) to normal operation.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will disable all controls.</p>
<p>GUI FCOLOUR colour, #ref1 [, #ref2, #ref3, etc]</p>	<p>This will change the foreground colour of the specified controls to 'colour' which is an RGB value for the drawing colour.</p> <p>'#ref' is the control's reference number.</p>
<p>GUI FORMATBOX #ref, Format, startX, startY, width, height, FColour, BColour</p>	<p>This will draw a box with rounded corners that can be used to create a virtual keypad for entry of data using a specific format.</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. ' FColour ' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>The 'Format' argument specifies the format of the entry as follows:</p> <p>DATE1 Date in UK/Aust/NZ format (dd/mm/yy)</p> <p>DATE2 Date in USA format (mm/dd/yy)</p> <p>DATE3 Date in international format (yyyy/mm/dd)</p> <p>TIME1 Time in 24 hour notation (hh:mm)</p> <p>TIME2 Time in 24 hour notation with seconds (hh:mm:ss)</p> <p>TIME3 Time in 12 hour notation (hh:mm AM/PM)</p> <p>TIME4 Time in 12 hour notation with seconds (hh:mm:ss AM/PM)</p> <p>DATETIME1 Date (UK fmt) and time (12 hour) (dd/mm/yy hh:mm AM/PM)</p> <p>DATETIME2 Date (UK fmt) and time (24 hour) (dd/mm/yy hh:mm)</p> <p>DATETIME3 Date (USA fmt) and time (12 hour) (mm/dd/yy hh:mm AM/PM)</p> <p>DATETIME4 Date (USA fmt) and time (24 hour) (mm/dd/yy hh:mm)</p> <p>LAT1 Latitude in degrees, minutes and seconds (dd° mm' ss" N/S)</p> <p>LAT2 Latitude with seconds to one decimal place (dd° mm' ss.s" N/S)</p> <p>LONG1 Longitude in degrees, minutes and seconds (ddd° mm' ss" E/W)</p> <p>LONG2 Longitude seconds to one decimal place (ddd° mm' ss.s" E/W)</p> <p>ANGLE1 Angle in degrees and minutes (ddd° mm')</p> <p>For example, this command:</p> <p style="padding-left: 40px;">GUI FORMATBOX #1, LAT1, 50, 50, 300, 50</p> <p>would create a format box which would accept the entry of latitude in the format of dd° mm' ss" N/S. The value of CtrlVal(#1) would be a string which includes the numbers and separating characters. For example an entry of 17 degrees, 32 minutes and 1 second south would result in the string 17° 32' 01" S</p> <p>MMBasic will try to position the virtual keypad on the screen so as to not obscure the format box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will be generated when the entry is complete and the keypad is hidden. .</p>

<p>GUI FORMATBOX CANCEL</p>	<p>This will dismiss a virtual keypad if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keypad is not displayed this command will do nothing.</p>
<p>GUI FRAME #ref, caption\$, startX, startY, width, height, colour</p>	<p>This will draw a frame which is a box with round corners and a caption. '#ref' is the control's reference number.</p> <p>'caption\$' is a string to display as the caption. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'colour' is an RGB value for the drawing colour. 'width', 'height' and 'colour' are optional and default to that used in previous controls.</p> <p>A frame is useful when a group of controls need to be visually brought together. It is also used to surround a group of radio buttons and MMBasic will arrange for the radio buttons surrounded by the frame to be exclusive. i.e. when one radio button is selected any other button that was selected and within the frame will be automatically deselected.</p> <p>A frame does not respond to touch.</p>
<p>GUI GAUGE #ref, StartX, StartY, Radius, FColour, BColour, min, max, nbrdec, units\$, c1, ta, c2, tb, c3, tc, c4</p>	<p>Define a graphical circular analogue gauge with a digital display in the centre. '#ref' is the control's reference number.</p> <p>'StartX' and 'StartY' are the coordinates of the centre of the gauge, 'Radius' is the distance from the centre to the outer edge.</p> <p>'min' is the minimum value of the gauge and 'max' is the maximum value (both floating point).</p> <p>'nbrdec' specifies the number of decimal places to be used when drawing the digital value in the centre of the gauge. Under this 'units\$' will be displayed.</p> <p>'Fcolour' is the colour used for the gauge while 'Bcolour' is the background colour. A multi colour gauge can be created using 'c1' to 'c4' for the colours and 'ta' to 'tc' for the thresholds used to determine when the colour will change. When colours and thresholds are specified the background of the gauge will be drawn with a dull version of the colour at that level. Also the digital value will change to the colour specified by the current value.</p> <p>'Radius', 'FColour', 'BColour', 'min', 'max', 'nbrdec' and 'units\$' are optional and will default to the values used in the previous definition of a GUI GAUGE.</p> <p>'c1', 'ta', 'c2', 'tb', 'c3', 'tc' and 'c4' are optional and if not specified the gauge will use less colours. If all are omitted the gauge will be drawn using 'Fcolour'.</p> <p>The section <i>Advanced Graphics</i> has a more detailed description.</p>
<p>GUI HIDE #ref1 [,#ref2, #ref3, etc] or GUI HIDE ALL</p>	<p>This will hide the controls in the list. Hidden controls do not respond to touch and will not be visible.</p> <p>'#ref' is the control's reference number. The keyword ALL can be used as the argument and that will hide all controls.</p> <p>GUI SHOW can be used to restore the controls.</p>
<p>GUI INTERRUPT down [, up]</p>	<p>This command will setup an interrupt that will be triggered on a touch on the LCD panel and optionally if the touch is released.</p> <p>'down' is the subroutine to call when a touch down has been detected. 'up' is the subroutine to call when the touch has been lifted from the screen ('up' and 'down' can point to the same subroutine if required).</p> <p>Specifying the number zero (single digit) as the argument will cancel both of these interrupts. ie: GUI INTERRUPT 0.</p>
<p>GUI LED #ref, caption\$, centerX, centerY, radius, colour</p>	<p>This will draw an indicator light which looks like a panel mounted LED. A LED does not respond to touch.</p> <p>'#ref' is the control's reference number.</p> <p>The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command.</p> <p>'centerX' and 'centerY' are the coordinates of the centre of the LED and 'radius'</p>

	<p>is the radius of the LED. 'colour' is an RGB value for the drawing colour. 'radius' and 'colour' are optional and default to that used in previous controls.</p> <p>When a LED's value is set to a value of one it will be illuminated and when it is set to zero it will be off (a dull version of its colour attribute). The LED can be made to flash on then off by setting the value of the LED to a number greater than one which is the time in milliseconds that it should remain on.</p> <p>The colour can be changed with the GUI FCOLOUR command.</p>
<p>GUI NUMBERBOX #ref, startX, startY, width, height, FColour, BColour</p>	<p>This will draw a box with rounded corners that can be used to create a virtual numeric keypad for data entry.</p> <p>'#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', FColour and 'BColour' are optional and default to that used in previous controls.</p> <p>When the box is touched a numeric keypad will appear on the screen. Using this virtual keypad any number can be entered into the box including a floating point number in exponential format. The new number will replace the number previously in the box.</p> <p>The value of the control can set to a literal string (not an expression) starting with two hash characters. For example: CtrlVal(nnn) = "##Enter Number" and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return zero. When the control is used normally the ghost text will vanish.</p> <p>MMBasic will try to position the virtual keypad on the screen so as to not obscure the number box that caused it to appear. A pen down interrupt will be generated just before the keypad is deployed and a key up interrupt will be generated when the Enter key is touched and the keypad is hidden. Also, when the Enter key is touched the entered number will be evaluated as a number and the NUMBERBOX control redrawn to display this number.</p>
<p>GUI NUMBERBOX CANCEL</p>	<p>This will dismiss a virtual keypad if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keypad is not displayed this command will do nothing.</p>
<p>GUI PAGE #n [,#n2, #n3, etc]</p>	<p>This will switch the display to show controls that have been assigned (via the GUI SETUP command) to the page numbers specified on the command line (#n, #n2, etc). Any controls that were displayed but are not on the current list of pages will be automatically hidden. Any controls on a page that was displayed on the old screen and is also specified in the new PAGE command will remain unaffected.</p> <p>The default when a program starts running is PAGE 1 and GUI SETUP 1. This means that if these commands are not used the program will run as normal showing all GUI controls that have been defined.</p> <p>See also the GUI SETUP command.</p>
<p>GUI RADIO #ref, caption\$, centerX, centerY, radius, colour</p>	<p>This will draw a radio button with a caption.</p> <p>'#ref' is the control's reference number.</p> <p>The string 'caption\$' will be drawn to the right of the control using the colours set by the COLOUR command.</p> <p>'centerX' and 'centerY' are the coordinates of the centre of the button and 'radius' is the radius of the button. 'colour' is an RGB value for the drawing colour. 'radius' and 'colour' are optional and default to that used in previous controls.</p> <p>When touched the centre of the button will be illuminated to indicate that this option has been selected and the control's value will be 1. When another radio</p>

	<p>button is selected the mark on this button will be removed and its value will be zero. Radio buttons are grouped together when surrounded by a frame and when one button in the group is selected all others in the group will be deselected. If a frame is not used all buttons on the screen will be grouped together.</p>
<p>GUI REDRAW #ref1 [,#ref2, #ref3, etc] or GUI REDRAW ALL</p>	<p>This will redraw the controls on the screen. It is useful if the screen image has somehow been corrupted.</p> <p>#ref' is the control's reference number. The keyword ALL can be used as the argument and that will first clear the screen then redraw all controls. This is useful if the whole screen needs to be refreshed.</p>
<p>GUI SETUP #n</p>	<p>This will allocate any new controls created to the page '#n'.</p> <p>This command can be used as many times as needed while GUI controls are being defined. The default when a program starts running is GUI SETUP 1. See also the GUI PAGE command.</p>
<p>GUI SHOW #ref1 [,#ref2, #ref3, etc] or GUI SHOW ALL</p>	<p>This will undo the effects of GUI HIDE and restore the control(s) to being visible and capable of normal operation.</p> <p>#ref' is the control's reference number. The keyword ALL can be used as the argument and that will disable all controls.</p>
<p>GUI SPINBOX #ref, startX, startY, width, height, FColour, BColour, Step, Minimum, Maximum</p>	<p>This will draw a box with up/down icons on either end. When these icons are touched the number in the box will be incremented or decremented. Holding down the up/down icons will repeat the step at a fast rate.</p> <p>#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', 'FColour' and 'BColour' are optional and default to that used in previous controls.</p> <p>'Step' sets the amount to increment/decrement the number with each touch. 'Minimum' and 'Maximum' set limits on the number that can be entered. All three parameters can be floating point numbers and are optional. The default for 'Step' is 1 and 'Minimum' and 'Maximum' if omitted will default to no limit.</p>
<p>GUI SWITCH #ref, caption\$, startX, startY, width, height, FColour, BColour</p>	<p>This will draw a latching switch which is a square switch that latches when touched.</p> <p>#ref' is the control's reference number.</p> <p>'caption\$' is a string to display as the caption on the face of the switch. 'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', 'FColour' and 'BColour' are optional and default to that used in previous controls.</p> <p>When touched the visual image of the button will appear to be depressed and the control's value will be 1. When touched a second time the switch will be released and the value will revert to zero. Caption can consist of two captions separated by a character (e.g. "ON OFF"). When this is used the switch will appear to be a toggle switch with each half of the caption used to label each half of the toggle switch.</p>
<p>GUI TEXTBOX #ref, startX, startY, width, height, FColour, BColour</p>	<p>This will draw a box with rounded corners that can be used to create a virtual keyboard for data entry</p> <p>#ref' is the control's reference number.</p> <p>'startX' and 'startY' are the top left coordinates while 'width' and 'height' set the dimensions. 'FColour' and 'BColour' are RGB values for the foreground and background colours. 'width', 'height', 'FColour' and 'BColour' are optional and default to that used in previous controls. On a display that supports transparent text BColour can be -1 which means that the background will show through the gaps in the characters.</p>

	<p>When the box is touched a QWERTY keyboard will appear on the screen. Using this virtual keyboard any text can be entered into the box including upper/lower case letters, numbers and any other characters in the ASCII character set. The new text will replace any text previously in the box.</p> <p>The value of the control can set to a string starting with two hash characters. For example: <code>CtrlVal(nnn) = "##Enter Filename"</code> and in that case the string (without the leading two hash characters) will be displayed in the box with reduced brightness. This can be used to give the user a hint as to what should be entered (called "ghost text"). Reading the value of the control displaying ghost text will return an empty string. When the control is used normally the ghost text will vanish.</p> <p>MMBasic will try to position the virtual keyboard on the screen so as to not obscure the text box that caused it to appear. A pen down interrupt will be generated just before the keyboard is deployed and a key up interrupt will be generated when the Enter key is touched and the keyboard is hidden. .</p>
GUI TEXTBOX CANCEL	This will dismiss a virtual keyboard if it is displayed on the screen. It is the same as if the user touched the cancel key except that the touch up interrupt is not generated. If a keyboard is not displayed this command will do nothing.

Functions

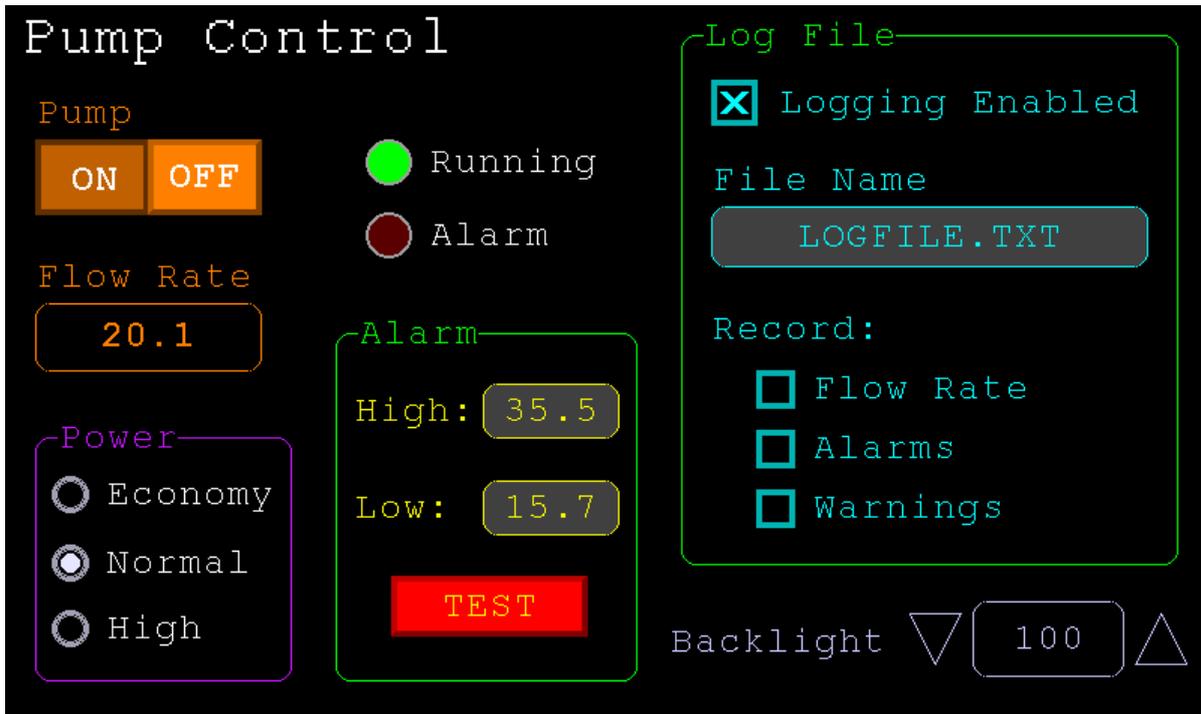
CTRLVAL(#ref)	<p>Returns the current value of an advanced control.</p> <p>'#ref' is the control's reference. For controls like check boxes or switches it will be the number one (true) indicating that the control has been selected by the user or zero (false) if not. For controls that hold a number (e.g. a SPINBOX) the value will be the number (normally a floating point number). For controls that hold a string (e.g. TEXTBOX) the value will be a string.</p>
MSGBOX (msg\$, b1\$ [,b2\$... b4\$])	<p>This function will display a message box on the screen with one to four touch sensitive buttons. All other controls will be disabled until the user touches one of the buttons. The message box will then be erased, the previous controls will be restored and the function will return the number of the button touched (the first button is number one)</p> <p>'msg\$' is the message to display. This can contain one or more tilde characters (~) which indicate a line break. Up to 10 lines can be displayed inside the box. 'b1\$' is the caption for the first button, 'b2\$' is the caption for the second button, etc. At least one button must be specified and four is the maximum. Any buttons not included in the argument list will not be displayed.</p>
CLICK(DOWN)	Will return true if the screen is currently being clicked on using the mouse.
CLICK(UP)	Will return true if the screen is currently NOT being clicked on.
CLICK(LASTX)	Will return the X coordinate of the last location that was clicked on.
CLICK(LASTY)	Will return the Y coordinate of the last location that was clicked on.
CLICK(REF)	Will return the reference number of the control that is currently being clicked on or zero if no control is currently being clicked on.
CLICK(LASTREF)	Will return the reference number of the last control that was clicked on.

Obsolete Commands and Functions

PAGE	Replaced with "GUI PAGE"
------	--------------------------

Example Program

As a test you can enter the following "Pump Control" demonstration program. It will draw a selection of advanced controls as shown below. These controls are active so that you can test how they work.



```

.....
' Demonstration program for the Advanced Graphics
' It does not do anything useful except demo the various controls
'
' Geoff Graham, October 2015
.....

Option Explicit
Dim ledsY
mode 10
option console serial
Colour RGB(white), RGB(black)
' reference numbers for the controls are defined as constants
Const c_head = 1, c_pmp = 2, sw_pmp = 3, c_flow = 4, tb_flow = 5
Const led_run = 6, led_alarm = 7
Const frm_alarm = 20, nbr_hi = 21, nbr_lo = 22, pb_test = 23
Const c_hi = 24, c_lo = 25
Const frm_pump = 30, r_econ = 31, r_norm = 32, r_hi = 33
Const frm_log = 40, cb_enabled = 41, c_fname = 42, tb_fname = 43
Const c_log = 44, cb_flow = 45, cb_pwr = 46, cb_warn = 47
Const cb_alarm = 48, c_bright = 49, sb_bright = 50
' now draw the "Pump Control" display
'CLS
dim integer x, y, lb, rb, sb, xs, ys, cs, ss
'sprite load "mouse.spr",1
gui cursor on 0, mm.hres\2,mm.vres\2
GUI Interrupt TouchDown, TouchUp
' display the heading
Font 2,2 : GUI Caption c_head, "Pump Control", 10, 0
Font 3 : GUI Caption c_pmp, "Pump", 20, 60, , RGB(brown)
' now, define and display the controls
' first
GUI Switch sw_pmp, "ON|OFF", 20, 90, 150, 50, RGB(white),RGB(brown)
CtrlVal(sw_pmp) = 1
' the flow rate display box

```

```

Font 3 : GUI Caption c_flow, "Flow Rate", 20, 170,, RGB(brown),0
Font 4 : GUI Displaybox tb_flow, 20, 200, 150, 45
CtrlVal(tb_flow) = "20.1"
' the radio buttons and their frame
Font 3 : GUI Frame frm_pump, "Power", 20, 290, 170, 163,RGB(200,20,255)
GUI Radio r_econ, "Economy", 40, 318, 15, RGB(230, 230, 255)
GUI Radio r_norm, "Normal", 40, 364
GUI Radio r_hi, "High", 40, 408
CtrlVal(r_norm) = 1 ' start with the "normal" button selected
' the alarm frame with two number boxes and a push button switch
Font 3 : GUI Frame frm_alarm, "Alarm", 220, 220, 200, 233,RGB(green)
GUI Caption c_hi, "High:", 232, 260, LT, RGB(yellow)
GUI Numberbox nbr_hi, 318,MM.info(VPos)-
6,90,MM.info(FontHeight)+12,RGB(yellow),RGB(64,64,64)
GUI Caption c_lo, "Low:", 232, 325, LT, RGB(yellow),0
GUI Numberbox nbr_lo, 318,MM.info(VPos)-
6,90,MM.info(FontHeight)+12,RGB(yellow),RGB(64,64,64)
GUI Button pb_test, "TEST", 257, 383, 130, 40,RGB(yellow), RGB(red)
CtrlVal(nbr_lo) = 15.7 : CtrlVal(nbr_hi) = 35.5
' draw the two LEDs
Const ledsX = 240, coff = 50 ' define their position
ledsY = 90 : GUI LED led_run, "Running", ledsX, ledsY, 15, RGB(green)
ledsY = ledsY+49 : GUI LED led_alarm, "Alarm", ledsX, ledsY, 15, RGB(red)
CtrlVal(led_run) = 1 ' the switch defaults to on so set the LED on
' the logging frame with check boxes and a text box
Colour RGB(cyan), 0
GUI Frame frm_log, "Log File", 450, 20, 330, 355, RGB(green)
GUI Checkbox cb_enabled, "Logging Enabled", 470, 50, 30, RGB(cyan)
GUI Caption c_fname, "File Name", 470, 105
GUI Textbox tb_fname, 470, 135, 290, 40, RGB(cyan), RGB(64,64,64)
GUI Caption c_log, "Record:", 470, 205, , RGB(cyan), 0
GUI Checkbox cb_flow, "Flow Rate", 500, 245, 25
GUI Checkbox cb_alarm, "Alarms", 500, 285, 25
GUI Checkbox cb_warn, "Warnings", 500, 325, 25
CtrlVal(cb_enabled) = 1
CtrlVal(tb_fname) = "LOGFILE.TXT"
' define and display the spinbox for controlling the backlight
GUI Caption c_bright, "Backlight", 442, 415,,RGB(200,200,255),0
GUI Spinbox sb_bright, MM.info(HPos) + 8, 400, 200, 50,,10, 10, 100
CtrlVal(sb_bright) = 50
' All the controls have been defined and displayed. At this point
' the program could do some real work but because this is just a
' demo there is nothing to do. So it just sits in a loop.
Do
Loop
' the interrupt routine for touch down
' using a select case command it has a different process for each
' control
Sub TouchDown
local integer mbox
' print "down"
Select Case click(REF) ' find out the control touched
Case cb_enabled ' the enable check box
If CtrlVal(cb_enabled) Then
GUI Restore c_fname, tb_fname, c_log, cb_flow, cb_alarm, cb_warn
Else
mbox=MsgBox("Are you sure?", "YES","CANCEL")
' print mbox
if mbox=1 then
GUI Disable c_fname, tb_fname, c_log, cb_flow, cb_alarm, cb_warn
else
CtrlVal(cb_enabled)=1
endif
EndIf
Case sb_bright ' the brightness spin box
' BackLight CtrlVal(sb_bright)
Case sw_pmp ' the pump on/off switch

```

```

    CtrlVal(led_run) = CtrlVal(sw_pmp)
    CtrlVal(tb_flow) = Str$(CtrlVal(sw_pmp) * 20.1)
    CtrlVal(r_norm) = 1
Case pb_test ' the alarm test button
    CtrlVal(led_alarm) = 1
'     GUI beep 250
Case r_econ ' the economy radio button
    CtrlVal(tb_flow) = Str$(CtrlVal(sw_pmp) * 18.3)
Case r_norm ' the normal radio button
    CtrlVal(tb_flow) = Str$(CtrlVal(sw_pmp) * 20.1)
Case r_hi ' the high radio button
    CtrlVal(tb_flow) = Str$(CtrlVal(sw_pmp) * 23.7)
End Select
End Sub
' interrupt routine when the touch is removed
Sub TouchUp
Select Case click(LASTREF) ' use the last reference
Case pb_test ' was it the test button
    CtrlVal(led_alarm) = 0 ' turn off the LED
End Select
End Sub
'
Function MM.CURSOR( t As Integer) As Integer
Static Float lasttime
If Timer > lasttime + 20 Or timer < lasttime Then
    X = Mouse(x,1)
    Y = Mouse(y,1)
    Lb = Mouse(l,1)
    lasttime = Timer
    Gui Cursor x,y
EndIf
MM.CURSOR = -1
If t = 1 Then
    MM.CURSOR = lb
ElseIf t = 2 Then
    If lb Then MM.CURSOR = x
ElseIf t = 3 Then
    If lb Then MM.CURSOR = y
ElseIf t = 4 Then
    MM.CURSOR = lb
Else
    MM.CURSOR = lb
EndIf
End Function
'
Sub MM.Beep beptime As Integer
If beptime And MM.Info(Sound) = "OFF" Then Play Tone 600, 600, beptime
End Sub

```