Pi-cromite
User
Manual

MMBasic Ver 5.05.01

This manual is distributed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Australia license (CC BY-NC-SA 3.0)

The Pi-cromite is a new addition to the Micromite family using the Raspberry Pi. The Pi-cromite firmware implements most of the features of the standard Micromite and the Micromite Plus as described in the *Micromite User Manual* and the *Micromite Plus Manual*. It has a number of differences and additional features and they are described in this document.
The focus of this manual is to describe just the features that are **unique** to the Pi-cromite. For general Micromite programming you should refer to the *Micromite User Manual* and the *Micromite Plus Manual* in addition to this manual.

Contents

Introduction

This section provides an introduction for users who are familiar with the Micromite and the Micromite Plus and need a summary of the extra features in the Pi-cromite.

The Pi-cromite is an extension of the standard Micromite and the Micromite Plus; most of the features of these two versions are also in the Pi-cromite. This includes features of the BASIC language, input/output, communications, etc. Some commands have changed slightly but for the main part Micromite programs will run unchanged on the Pi-cromite.

The following summarises features of the Pi-cromite as compared to the standard Micromite and the Micromite Plus:

Raspberry Pi

The Pi-cromite is based on the Raspberry Pi. The Raspberry Pi is available a number of versions and has up to fifteen times the program space of the MX series used in the standard Micromite and is many times faster. The Pi3 is 5x faster than even a 252MHz PIC32MZ.

High Speed Double Precision Floating Point

The Pi-cromite uses the built in hardware floating point capability of the Broadcom processor which is much faster than floating point on the standard Micromite and uses double precision floating point.

I/O Pins

The Pi-cromite has 26 free I/O pins. The Broadcom chip does not support analogue input.

The Pi-cromite has one I^2C port which can be implemented on any pair of pins, one SPI port, one high speed PWM channel (low speed PWM and servo control can be implemented on any/all I/O pins) and two serial COM ports.

High Speed LCD Panels

The Pi-cromite supports eleven different sized LCD display panels from 1.44" to 8

Socket I/O

The Pi-cromite can support opening a socket which allows network based applications to be programmed directly in Basic. The TRANSMIT command will automatically create valid HTTP headers including file lengths making web programming simple

Micromite Family Summary

The Micromite Family consists of three major types, the standard Micromite, the Micromite Plus and the Pi-cromite. All use the same BASIC interpreter and have the same basic capabilities however they differ in the number of I/O pins, the amount of memory, the displays that they support and their intended use.

Standard Micromite — Comes in a 28-pin or 44-pin package and is designed for small embedded controller applications and supports small LCD display panels. The 28-pin version is particularly easy to use as it is easy to solder and can be plugged into a standard 28-pin IC socket.

Micromite Plus — This uses a 64-pin and 100-pin TQFP surface mount package and supports a wide range of touch sensitive LCD display panels from 1.44" to 8" in addition to the standard features of the Micromite. It is intended as a sophisticated controller with easy to create on-screen controls such as buttons, switches, etc.

Micromite eXtreme — This comes in 64, 100-pin and 144-pin TQFP surface mount packages. The eXtreme version has all the features of the other two Micromites but is faster and has a larger memory capacity plus the ability to drive a VGA monitor for a large screen display. It works as a powerful, self contained computer with its own BASIC interpreter and instant start-up.

Pi-cromite — Runs on all versions of the Raspberry Pi with a 40-pin I/O connector. No analogue input capability but 5x faster than a Micromite eXtreme when running on a Pi 3.

	Micromite		**Micromite Plus**		**Micromite eXtreme**			**Pi-cromite**
	28-pin DIP	44-pin SMD	64-pin SMD	100-pin SMD	100-pin SMD	144-pin SMD	64-pin SMD	Raspberry Pi
Maximum CPU Speed	48 MHz	48 MHz	120 MHz	120 MHz	252MHz	252 MHz	252 MHz	1200MHz
Maximum BASIC Program Size	59 KB	59 KB	100 KB	100 KB	540 KB	540 KB	540 KB	1024KB
RAM Memory Size	52 KB	52 KB	108 KB	108 KB	460 KB	460 KB	460 KB	1024KB
Clock Speed (MHz)	5 to 48	5 to 48	5 to 120	5 to 120	200 to 252	200 to 252	200 to 252	700-1200
Total Number of I/O pins	19	33	45	77	75	115	46	26
Number of Analog Inputs	10	13	28	28	40	48	24	0
Number of Serial I/O ports	2	2	3 or 4	3 or 4	3 or 4	3 or 4	3 or 4	2
Number of SPI Channels	1	1	2	2	3	3	2	1
Number of I^2C Channels	1	1	1 + RTC	1 + RTC	2 + RTC	2 + RTC	1 + RTC	1
Number of 1-Wire I/O pins	19	33	45	77	75	115	46	26
PWM or Servo Channels	5	5	5	5	6	6	6	26
Serial Console	✓	✓	✓	✓	✓	✓	✓	Native Linux
USB Console			✓	✓	✓	✓	✓	Native Linux
PS2 Keyboard and LCD Console			✓	✓	✓	✓	✓	Native Linux
SD Card Interface			✓	✓	✓	✓	✓	Native Linux
Supports ILI9341 LCD Displays	✓	✓	✓	✓	✓	✓	✓	✓
Supports Ten LCD			✓	✓	✓	✓	✓	✓+

Panels from 1.44" to 8" (diameter)								ILI9481
Supports VGA Displays					✓	✓		
Sound Output (WAV/tones)			✓	✓	✓	✓	✓	Native Linux
Supports PS2 Mouse Input					✓	✓	✓	Native Linux
Floating Point Precision	Single	Single	Single	Single	Double	Double	Double	Double
Power Requirements	3.3V 30 mA	3.3V 30 mA	3.3V 80 mA	3.3V 80 mA	3.3V 160 mA	3.3V 160 mA	3.3V 160 mA	5V 100mA – 1.6A

Suitable Raspberry Pi and Installation Requirements

The Pi-cromite firmware supports any Raspberry Pi with the 40-pin I/O header. The code should work properly on A+, B+, Pi Zero (W), Pi2B, Pi3B but not currently on Model A, B, or B (revision 2).

It is recommended that Raspbian Lite is installed on single CPU versions of the Pi. Full Raspbian can be used on multi-CPU versions.

The code has been developed and tested on the Pi 3 Model B version 1.2 (full Raspbian) and the Pi Zero W V1.1 (Raspbian Lite) using Netbeans IDE V8.2 running on a W10 PC.

In order for the firmware to work pigpio software version 68 must be installed see: http://abyz.co.uk/rpi/pigpio/download.html for details of how to install this version.

Pins may be reserved for Linux use allowing devices like I2S audio DACs to be accessed. See the command ***OPTION PINS*** for more details.

All system devices and Linux commands can be accessed from MMBasic. See the ***SYSTEM*** command for more details.

The 3.3V rail on a Pi 3 is very noisy when the Pi is powered with the official Pi mains adapter. Devices such as IR receivers should be powered from the 5V rail using a 3.3V linear regulator to avoid issues.

There are known timing issues with the Pi 3 that may affect serial communications and Bitstream output – see the ***OPTION WAVETIME*** command for more details.

The Pi-cromite firmware should be copied to a suitable directory on the Pi before use. When first run the firmware creates a hidden file ***.options*** in the same directory. This file should be deleted before any new version is installed.

The firmware must be set as executable before running using ***chmod +x mmbasic***

The firmware requires privileged access to the Pi hardware and so must be run using the sudo command
sudo ./mmbasic

NB The Pi-cromite firmware sits in a tight loop polling for input and so will use 100% of one CPU. It is running at priority zero most of the time so the operating system is able to time-slice processor access. During time-sensitive I/O operations the priority will be raised and other processes locked out.

NB Do not run or use the PIGPIO Daemon

40-pin Pi-cromite Pinout

Pin				
1	3V3			
2	5V			
3	DIGITAL_IN	DIGITAL_OUT	I2C-SDA*	
4	5V			
5	DIGITAL_IN	DIGITAL_OUT	I2C-SCL*	
6	GND			
7	DIGITAL_IN	DIGITAL_OUT	COM2-TX	
8	DIGITAL_IN	DIGITAL_OUT	COM1-TX	
9	GND			
10	DIGITAL_IN	DIGITAL_OUT	COM1-RX	
11	DIGITAL_IN	DIGITAL_OUT	COM2-RX	
12	DIGITAL_IN	DIGITAL_OUT	COUNT	PWM-HS
13	DIGITAL_IN	DIGITAL_OUT	SSD1963-RS	ILI9341-CS*
14	GND			
15	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB3	
16	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB4	
17	3V3			
18	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB5	
19	DIGITAL_IN	DIGITAL_OUT	SPI-OUT	
20	GND			
21	DIGITAL_IN	DIGITAL_OUT	SPI-IN	
22	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB6	
23	DIGITAL_IN	DIGITAL_OUT	SPI-CLK	
24	DIGITAL_IN	DIGITAL_OUT	COUNT	TOUCH-IRQ*
25	GND			
26	DIGITAL_IN	DIGITAL_OUT	TOUCH-CS*	
27	ID_SD			
28	ID_SC			
29	DIGITAL_IN	DIGITAL_OUT	COUNT	ILI9341-DC*
30	GND			
31	DIGITAL_IN	DIGITAL_OUT	SSD1963-RESET	ILI9341-RESET*
32	DIGITAL_IN	DIGITAL_OUT	SSD1963-WR	
33	DIGITAL_IN	DIGITAL_OUT	COUNT	IR
34	GND			
35	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB0	
36	DIGITAL_IN	DIGITAL_OUT	SSD1963-RD	
37	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB7	
38	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB1	
39	GND			
40	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB2	

* Recommended usage for compatibility between PCB designs

Pi-cromite Features

Double Precision Floating Point

The Pi-cromite uses the hardware floating point capability of the Broadcom chip and can therefore process floating point calculations faster than the Micromite and Micromite Plus. All floating point uses double precision calculations.

Twenty Six PWM Channels

All pins can be used for PWM output up to 20KHz as well as for driving servos. In addition pin 12 can be used for PWM output up to 25MHz

MM.DEVICE$

On the Pi-cromite the read only variable MM.DEVICE$ will return " Pi-cromite running on H/W version: hhhhhh ".

CPU command

The Pi-cromite does not support dynamically changing the CPU speed or the sleep function. Accordingly the commands CPU speed and CPU SLEEP are not available. However the Pi-cromite does support “CPU SLEEP time” where time is specified in seconds.

OPTION CONTROLS command

The Pi-cromite does not support the OPTION CONTROLS command instead the maximum number of GUI controls is set to 500.

Longstring handling

The Pi-cromite supports a comprehensive set of commands and functions for handling long strings stored in integer arrays

Mouse control in editor

While in the editor the left mouse button can be used to position the cursor. Just click on any character and the edit point will move to just before that character with the cursor on the character. Click just after the last character in a line to go to the end of that line. Click well past the end of a line to go to the start of the next line.

The mouse wheel can be used to scroll up and down the file.

Delete and Backspace

Teraterm, Putty, and the Raspian desktop console window generate different codes for backspace and delete. The Pi-cromite defaults to support the console window and Putty. Use OPTION TERATERM ON to accept the default codes from Teraterm. Alternatively configure teraterm in the Keyboard setup window

Ctrl-C and Ctrl-Z

Use Ctrl-C to terminate a running program, to exit GUI TEST LCDPANEL, to exit GUIT TEST TOUCH or to exit from auto input mode.

Use Ctrl-Z at the command prompt to exit MMBasic and return to the Linux command prompt

I2C

Before using I2C the pins to be used must be selected with OPTION I2C SDApinno, SCLpinno

Then you can use I2C exactly the same as the Micromite with the following limitations:

The implementation does not support 10-bit addressing (i.e. options 0 and 1 only)

The implementation is Master only.

DATE$ and TIME$

Date$ and Time$ are derived from the Linux system clock. They are returned in standard MMBasic format when read but cannot be set by MMBasic (except by using the SYSTEM command with the requisite Linux syntax)

VAR command

The Pi-cromite does not support the VAR command. On the Pi-cromite variables can be saved and restored from a file as the SDcard is always available

SETPIN command

The Pi-cromite does not support analogue input so SETPIN nnn,AIN is not allowed but see function ADC and command OPTION ADC

Automatic Console re-sizing

The Pi-cromite does not support the OPTION DISPLAY command. Instead the code will automatically set the display width and height based on the size of the terminal window

FONTS

The Pi-cromite does not support loadable fonts but includes an additional small 6x8 font as #7

Commands (Pi-cromite Only)

BITSTREAM pin, nbr, dur%()	Generates a stream of 'nbr' Hi/Lo (OR Lo/Hi) transitions on pin number 'pin'. The duration of each state after the transition is determined by the contents of array 'dur%()' which is specified in micro-seconds. The arrays can be INTEGER or FLOAT but the pulse length must be specified in microseconds. The minimum pulse length and pulse increment is 1uS The pin must previously have been set as a digital output. The direction of the first pulse will be determined by the state of the pin when BITSTREAM is called. If it is high then the first pulse will be at zero, if it is low the first pulse will be at 3.3V. The BITSTREAM command will automatically create an end transition to the last pulse. **NB There is a known clock issue on the Pi 3. See OPTION WAVETIME for details**
BOX x1, y1, w, h [, lw] [,c] [,fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, w, and h must all be arrays or all be single variables /constants otherwise an error will be generated. lw, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
CIRCLE x, y, r [,lw] [, a] [, c] [, fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x, y and r must all be arrays or all be single variables /constants otherwise an error will be generated. lw, a, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
CLS CONSOLE	Clears the text console
CURSOR MOVE x,y	Moves the text console cursor position to the X,Y position specified
CURSOR COLOUR "fg", "bc"	Set the foreground and background colours for the text console. Only valid if OPTION COLOURCODE ON is set. Valid colours must be in quotes and are the normal 8 primaries RED, BLUE WHITE, BLACK, GREEN, CYAN, MAGENTA, YELLOW
CURSOR UNDERLINE mode	Sets/removes underline on the subsequent output. Valid modes are ON and OFF
CURSOR REVERSE mode	Sets/removes reverse colours on the subsequent output. Valid modes are ON and OFF
CURSOR BOLD mode	Sets/removes bold text mode on the subsequent output. Valid modes are ON and OFF

FORK unix_command	Executes the unix_command as a new process. Non-blocking. Output from the process is directed to the null device. Only one process may be started at any one time. Returns an error if a second process is attempted to be started before the first terminates e.g. FORK aplay sound.wav The status of the process may be tested with MM.FORK (see below) Note that the unix command does not need quotes or specific field separators but commas may be used instead of spaces if preferred. Note also that the C-library parser does not accept "-" characters in filenamesas these are treated as specific to command syntax.
FORK KILL	Stops execution of any forked process. Returns an error if no forked process exists.
GUI STARTLINE n	Sets the row in the graphics memory which will appear at the top of the screen (landscape or reverse landscape) or left of the screen (portrait or reverse portrait) for a 4.3” SSD1963 display initialised with OPTION LCDPANEL SSD1963_4P
LINE x1, y1, x2, y2 [, LW [, C]]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, x2, and y2 must all be arrays or all be single variables /constants otherwise an error will be generated. lw and c can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
LONGSTRING APPEND array%(), string$	Append a normal MMBasic string to a long string variable. array%() is a long string variable while string$ is a normal MMBasic string expression.
LONGSTRING CLEAR array%()	Will clear the long string variable array%(). ie, it will be set to an empty string.
LONGSTRING COPY dest%(), src%()	Copy one long string to another. dest%() is the destination variable and src%() is the source variable. Whatever was in dest%() will be overwritten.
LONGSTRING CONCAT dest%(), src%()	Concatenate one long string to another. dest%() is the destination variable and src%() is the source variable. src%() will the added to the end of dest%() (the destination will not be overwritten).
LONGSTRING LCASE array%()	Will convert any uppercase characters in array%() to lowercase. array%() must be long string variable.
LONGSTRING LEFT dest%(), src%(), nbr	Will copy the left hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%(). ie, copy from the beginning of src%(). src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING LOAD array%(), nbr, string$	Will copy 'nbr' characters from string$ to the long string variable array%() overwriting whatever was in array%().
LONGSTRING MID dest%(), src%(), start, nbr	Will copy 'nbr' characters from src%() to dest%() starting at character position 'start' overwriting whatever was in dest%(). ie, copy from the middle of src%(). 'nbr' is optional and if omitted the characters

	from 'start' to the end of the string will be copied src%() and dest%() must be long string variables. 'start' and 'nbr' must be an integer constants or expressions.
LONGSTRING REPLACE array%() , string$, start	Will substitute characters in the normal MMBasic string string$ into an existing long string array%() starting at position ‘start’ in the long string.
LONGSTRING RIGHT dest%(), src%(), nbr	Will copy the right hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%(). ie, copy from the end of src%(). src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING TRIM array%(), nbr	Will trim ‘nbr’ characters from the left of a long string. array%() must be a long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING UCASE array%()	Will convert any lowercase characters in array%() to uppercase. array%() must be long string variable.
NANO	Allows you to edit the current program using the Linux nano editor. MMBasic will automatically write the current program (if any) to a temporary file and open it in nano. To return to MMBasic save the file to the temporary file and exit nano. The edited file will then be restored into MMBasic memory
OPTION ADC type, address	Provides support for 2 types of 4-channel I2C ADC. Valid types are ADS1015 and MCP3424. The address is the 7-bit I2C address of the device. The ADC must be connected SDA to Pi pin 3, SCL to Pi pin 5. When the option is specified the I2C port is automatically opened as soon as MMBasic starts. I2C speed is set to 400KHz. See the ADC function to read values. For the Pimoroni Enviro pHAT use OPTION ADC ADS1015, &H49
OPEN “SOCKET [,interrupt] ,[count]” as #n	Opens a socket for use in a MMBasic program. Once the socket is open then PRINT and INPUT commands and functions can be used exactly like any other file I/O. Print output is buffered until a TRANSMIT HTML command is sent which will create a valid HTTP header The optional parameters specify a MMBasic interrupt routine ‘interrupt’ to be called when ‘count’ characters have been received on the socket. See Appendix B for an example of socket programming.
OPTION AUTOREFRESH mode	If OPTION AUTOREFRESH ON is set drawing and GUI commands will update the screen immediately. If set to OFF then the screen will only be updated when a REFRESH command is called. Defaults to ON when MMBasic is first started - not stored.
OPTION AUTORUN mode	If OPTION AUTORUN ON is set the code will try and load and run the file "AUTORUN.BAS" on start up. The file must be in the same directory as MMBasic. If the file does not exist you will get a "file not found" error in which case just disable the option using OPTION AUTORUN OFF. The filename is case sensitive.
OPTION CLOCK type	This command is used to specify which of the two main Pi clocks are used for MMBasic. The default allows hardware PWM to be used on pin 12 but will cause external commands using the PCM clock, such as output on the i2s port, to fail. Valid values are ‘PCM’ (default) and ‘PWM’. See Appendix D for an example of usage.

OPTION I2C SDApin, SCLpin	Specifies which pins are to be used for I2C. Any valid pin may be selected. They can be changed by using OPTION I2C DISABLE and then re-defining them. Must be run once before I2C can be used. The values are permanently stored in the .options file. This option is automatically set if OPTION ADC is in use
OPTION LCDPANEL HDMI	Sets up an HDMI display for graphics/GUI output. The code automatically reads in the display size and MM.HRES and MM.VRES are set automatically for you. The display must be set up in full 32-bit colour (RGBA). See Appendix E for more information.
OPTION LCDPANEL SSD1963_4P	Sets the 4.3" SSD1963 display up in 480 x 864 (landscape or reverse landscape) or 864x480 (portrait or reverse portrait) pixel mode. The screen viewport is 480x272 or 272x480 and the position of the viewport is controlled by GUI STARTLINE n. This mode of operation allows display updates to be done on a non-visible part of the graphics memory and then the viewport moved to see the updated image. The 4P display controller is fully compatible with TOUCH, MOUSE, CURSOR and GUI controls
OPTION LCDPANEL spidisplay, orientation, DCpin, RESETpin, CSpin, SPIspeed	Initialises a SPI TFT. Additional display types include the ILI9481 and ILI9486. A new parameter SPIspeed can be used to set the transfer rate. Valid values are 1,000,000 to 30,000,000. The ILI9481 defaults to 16,000,000 all other displays default to 30,000,000. See the Micromite User manual, ILI9341 section, for further details of parameter usage.
OPTION PINS pinmask	Specify pins to be excluded from MMBasic e.g. 'OPTION PINS &B10100' will exclude pins 3 and 5 from use by MMBasic OPTION LIST will report the 'pinmask' if non-zero SETPIN reservedpin,DOUT will report a "reserved on boot" error See Appendix D for an example of usage.
OPTION SERIAL CONSOLE n	Sets the device to be used as the MMbasic console device. By default this is 0 which is the terminal used to start the process. Alternatives are 1 and 2. Selecting one of these sets the console to use the COM1 or COM2 port. Changing the selection will automatically terminate MMBasic and it can be restarted. With the console set to COM1 or COM2 it is possible to run MMBasic as a background process using the Linux command: **sudo ./mmbasic &>/dev/null &** This could be included as part of the boot process by adding the following lines to /etc/rc.local **echo "starting MMBasic"** **cd /home/pi** **sudo ./mmbasic &>/dev/null &** This will automatically run MMBasic on a "Pi" as a pseudo embedded application and if OPTION AUTORUN is set and the file "AUTORUN.BAS" exists then the user application will also start automatically. A standard USB/Uart can be used to connect to the COM port at anytime to access the MMBasic console.
OPTION SOCKET sockno	Sets the socket number to use when opening a socket. This defaults to 80 but can be anything between 1 and 65535
OPTION TERATERM mode	Sets the treatment of backspace and delete characters. Default mode is 'OFF'. Setting mode to 'ON' will allow teraterm to be used properly in the editor. Default should suit Putty and the Linux console window

OPTION WAVETIME scale!	Sets a scaling factor to adjust the timing of serial output and bitstream output.. Values of 'scale' > 1.0 will increase the speed of the output. Defaults to 1.0. This parameter may need to be adjusted to correct for a kernel issue on the Pi 3. Timings can be affected by factors like the presence (or not) of a HDMI monitor, the presence (or not) of a USB mouse and/or keyboard, and the use of Raspbian Lite or Full. The best way to determine if you have an issue is to open the serial port with a baudrate of 100 and transmit the character "U" repeatedly. Then measure the pulse length of the output which should be 10msec. The option is only active when MMBasic determines it is running on a Pi 3
PIXEL x, y [,c]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x and y must both be arrays or both be single variables /constants otherwise an error will be generated. c can be either an arrays or single variable/constant. See the Micromite User manual for full details of parameter usage.
PWM pin, freq, duty PWM pin, STOP	Sets up a PWM output on 'pin' with frequency 'freq' Hz and duty cycle of 'duty'. There is no limit on the number of pins used in this way. Frequencies and duty cycles are individually set per pin within the following restrictions: SPECIAL CASE PIN 12 This has true hardware PWM and the frequency can be set anywhere between 1Hz and 25MHz. Duty cycle is expressed as a percentage (like the Micromite). The underlying clock is 250MHz so with a 25MHz output the duty cycle will move in steps of 25/250 = 10%. At 250Hz the duty cycle will move in steps of 0.0001% ALL OTHER PINS These use a PWM based on a 1uS clock and the valid frequencies will be exact divisors of 1,000,000. The maximum frequency is 20KHz where the duty cycle will move in steps of 2%. If a frequency of say 433Hz is input the actual frequency will round to the nearest of: 1,2,4,5,8,10,16,20,25,32,40,50,80,100,125,160,200,250,400,500,625,800,1000,1250,2000,2500,4000,5000,10000,20000 Stops PWM output on pin
REFRESH	Forces a screen update when OPTION AUTOREFRESH is OFF
RBOX x1, y1, w, h [, r] [,c] [,fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, w, and h must all be arrays or all be single variables /constants otherwise an error will be generated. r, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
SERVO pin, period SERVO pin, STOP	Sets up a PWM output on 'pin' with a fixed frequency of 50Hz. Valid periods are specified in milli-seconds and are in the 'valid' range 1.0-2.0. There is no limit on the number of pins used in this way. Periods are individually set per pin. Of course you can use the PWM command to generate any non-standard servo output but be aware this can damage a servo. Stops SERVO output on pin

SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]	Calculates pitch, roll and yaw angles from accelerometer and magnetometer inputs. Valid fusion types are MAHONY and MADGWICK. Usage is described in Appendix A
SPRITE	The BLIT and SPRITE commands can be used interchangeably. The command is available on all display types.
SPRITE x1, y1, x2, y2, w, h	Copies the memory area specified by top right coordinate x1, y1 and of width w and height h to a new location where the top right coordinate is x2, y2.
SPRITE CLOSE [#]n	Closes sprite n and releases all memory resources. Updates the screen (see SPRITE HIDE). Sprites which have been “copied” cannot be closed until all “copies” have been closed
SPRITE CLOSE ALL	Closes all sprites and releases all memory resources. It does not change the screen.
SPRITE COPY [#]n, [#]m, nbr	Makes a copy of sprite “n” to “nbr” of new sprites starting a number “m”. Copied sprites share the same loaded image as the original to save memory
SPRITE HIDE [#]n	Removes sprite n from the display and replaces the stored background. To restore a screen to a previous state sprites should be hidden in the opposite order to which they were written "LIFO"
SPRITE INTERRUPT sub	Specifies the name of the subroutine that will be called when a sprite collision occurs. See Appendix C for how to use the function SPRITE to interrogate details of what has collided
SPRITE LOAD [#]n, fname$ [,colour]	Loads the file fname$ as a sprite into buffer number n. The file must be in PNG format RGB888 or RGBA888. If the file extension .PNG is omitted then it will be automatically added. The parameter “colour” specifies the background colour for the sprite. Pixels in the background colour will not overwrite the background when the sprite is displayed. Colour defaults to zero
SPRITE MOVE	Actions a single atomic transaction that re-locates all sprites which have previously had a location change set up using the SPRITE NEXT command. Collisions are detected once all sprites are moved and reported in the same way as from a scroll
SPRITE NEXT [#]n, x, y	Sets the X and Y coordinate of the sprite to be used when the screen is next scrolled or the SPRITE MOVE command is executed. Using SPRITE NEXT rather than SPRITE SHOW allows multiple sprites to be moved as part of the same atomic transaction.
SPRITE NOINTERRUPT	Disables collision interrupts
SPRITE READ [#]n, x , y, w, h	Reads the display area specified by coordinates x and y, width w and height h into buffer number n. If the buffer is already in use and the width and height of the new area are the same as the original then the new command will overwrite the stored area.
SPRITE SCROLLH n [,col]	Scrolls the background and any sprites on layer 0 n pixels to the right. n can be any number between -31 and 31. Sprites on any layer other than zero will remain fixed in position on the screen. By default the scroll wraps the image round. If “col” is specified the colour will

	replace the area behind the scrolled image
SPRITE SCROLLR x, y, w, h, delta_x, delta_y [,col]	Scrolls the region of the screen defined by top-right coordinates “x” and “y” and width and height “w” and “h” by “delta_x” pixels to the right and “delta_y” pixels up. By default the scroll wraps the background image round. If “col” is specified the colour will replace the area behind the scrolled image. Sprites on any layer other than zero will remain fixed in position on the screen. Sprites in layer zero where the centre of the sprite (x+ w/2, y+ h/2) falls within the scrolled region will move with the scroll and wrap round if the centre moves outside one of the boundaries of the scrolled region.
SPRITE SCROLLV n [,col]	Scrolls the background, and any sprites on layer 0, n pixels up. n can be any number between -MM.VRES-1 and MM.VRES-1. Sprites on any layer other than zero will remain fixed in position on the screen. . By default the scroll wraps the image round. If “col” is specified the colour will replace the area behind the scrolled image
SPRITE SHOW [#]n, x, layer, [orientation]	Displays sprite n on the screen with the top left at coordinates x, y. Sprites will only collide with other sprites on the same layer, layer zero, or with the screen edge. If a sprite is already displayed on the screen then the SPRITE SHOW command acts to move the sprite to the new location. The display background is stored as part of the command such that it can be replaced when the sprite is hidden or moved further. The orientation is an optional parameter, valid values are: 0 - normal display (default if omitted) 1 - mirrored left to right 2 - mirrored top to bottom 3 - rotated 180 degrees (= 1+2)
SPRITE WRITE [#]n, x y	Overwrites the display with the contents of sprite buffer n with the top left at coordinates x, y. SPRITE WRITE overwrites the complete area of the display. The background that is overwritten is not stored so SPRITE WRITE is inherently higher performing than SPRITE SHOW but with greater functional limitations.
SYSTEM string$ [,array%()]	Executes the Linux operating system command in ‘string$’. If the optional parameter is specified, the output from the system command will be directed to the long string ‘array%()’ otherwise output will appear on the console (stdout). Output can also be directed to a file using standard Linux notation. See Appendix C for examples of usage.

TCP CLIENT server$, serverport	Opens a TCPIP socket as a client to a designated server. The server IP address is specified as a string of the form “192.168.1.149”. The serverport is the port number the intended server is expected to be listening on e.g. 80 for an HTML server.
TCP CLOSE	Closes an open TCPIP socket
TCP SEND string$	Sends a string to a connected device via TCPIP to the server specified in the TCP CLIENT statement.
TCP RECEIVE string$	Attempts to receive a message on an opened socket. If no message is waiting string$ be a blank string. If a message is waiting it will be returned in string$. If the message will not fit into a string then repeated calls can be used to read all the message. A blank string will then indicate end of message
TRANSMIT CODE nnn	Constructs and sends a numerical response to the open socket. Typical use would be “TRANSMIT CODE 404” to indicate page not found.
TRANSMIT FILE filename, content-type	This constructs an HTTP 1.1 header with the ’content-type’ specified and the length of the file, sends it and then sends the contents of the file to the open socket
TRANSMIT HTML	This constructs an HTTP 1.1 HTML header and transmits it together with any output created by print #n commands to the file number specified by a OPEN “SOCKET” command
TRANSMIT NOHEADER	Transmits any output created by print #n commands to the file number specified by a OPEN “SOCKET” command
TRANSMIT PAGE filename [,content-type]	This constructs an HTTP 1.1 header with the ’content-type’ specified and the length of the file, sends it and then sends the contents of the file to the open socket. The content-type will default to text/html if not specified. MMBasic will substitute current values for any MMBasic variables defined in the file inside curly brackets e.g. {myvar%} Variables can be simple or array elements. An opening curly bracket can be included in the output by using {{. See Appendix C for examples of usage.
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, x2, y2, x3,and y3 must all be arrays or all be single variables /constants otherwise an error will be generated c and fill can be either arrays or single variables/constants. See the Micromite Plus manual for full details of parameter usage.

UDP SERVER nnnnn	Opens a UDP socket on the port nnnnn
UDP CLOSE	Closes an open UDP socket
UDP CLIENT server$, serverport, myport	Opens a socket as a client to a designated server. The server IP address is specified as a string of the form “192.168.1.149”. The serverport is the port number the intended server is expected to be listening on e.g. 123 for an ntp server. myport is the port the Pi-cromite will use to receive information from the server
UDP SEND string$	Sends a string to a connected device via UDP. When in client mode this will be the server specified in the UDP CLIENT statement. When in server mode this command can only be used after receiving an incoming message from a client as this will determine the address to use for SEND. An error will be reported if SEND is used in server mode before the link is established.
UDP RECEIVE string$ [,sender$] [,senderport%]	Attempts to receive a message on an opened socket. If no message is waiting string$ and sender$ will be blank strings and senderport will be zero. If a message is waiting it will be returned in string$. If sender$ is included in the command this will contain the IP address of the sender as a string e.g.”192.168.1.149”. if senderport% is specified it will contain the port number on which the sender would expect any reply. The senders IP address and port are automatically stored within the Pi-cromite and are used for any subsequent UDP SEND command

Functions (Pi-cromite Only)

ADC(channel, range [,precision])	Returns the input voltage on the specified ADC channel (0-3). The range is specified as the full-scale in mV. Valid ranges for the MCP3424 are 256, 512, 1024, and 2048. Valid ranges for the ADS1015 also include 4096 and 6144. The MCP3424 must also have the number of bits of precision specified. Valid values are 12, 14, 16, and 18. The conversion time on the MCP3424 changes with the precision – see the manual for details. In addition the range can be specified as AUTO in this case the code does a first conversion at the maximum range and then chooses the lowest safe range for a second more accurate conversion. Care should be taken in using AUTO with moving signals
FIELD$(str$, field) or FIELD$(str$, field, delim$) or FIELD$(str$, field, delim$, quote$)	Extract a substring (ie, field) from 'str$'. Each is separated by any one of the characters in the string 'delim$' and the field number to return is specified by 'field' (the first field is field number 1). Any leading and trailing spaces will be trimmed from the returned string. Note that 'delim$' can contain a number of characters and the fields will then be separated by any one of these characters. if delim$ is not provided it will default to a comma (,) character. 'quote$' is the set of characters that might be used to quote text. Typically it is the double quote character (") and any text that is surrounded by the quote character(s) will be treated as a block and any 'delim$' characters within that block will not be used as delimiters. This function is useful for splitting apart comma-separated-values (CSV) in data streams produced by GPS modules and test equipment. For example: PRINT FIELD$("aaa,bbb,ccc", 2, ",") Will print the string: bbb PRINT FIELD$("text1, 'quoted, text', text3", 2, ",", """) will print the string: 'quoted, text'
GETIP$(address$)	Translates a web address such as www.thebackshed.com into the IP address nnn.nnn.nnn.nnn. The IP address is returned as a string which can be used as an input to UDP CLIENT
JSON$(array%(),string$)	Returns a string representing a specific item out of the JSON input stored in the longstring array%() e.g. JSON$(a%(), “name”) JSON$(a%(), “coord.lat”) JSON$(a%(), “weather[0].description”) JSON$(a%(),”list[4].weather[0].description Examples taken from api.openweathermap.org
LCOMPARE(array1%(), array2%())	Compare the contents of two long string variables array1%() and array2%(). The returned is an integer and will be -1 if array1%() is less than array2%(). It will be zero if they are

	equal in length and content and +1 if array1%() is greater than array2%(). The comparison uses the ASCII character set and is case sensitive.
LGETSTR$(array%(), start, length)	Returns part of a long string stored in array%() as a normal MMBasic string. The parameters start and length define the part of the string to be returned.
LINSTR(array%(), search$ [,start])	Returns the position of a search string in a long string. The returned value is an integer and will be zero if the substring cannot be found. array%() is the string to be searched and must be a long string variable. Search$ is the substring to look for and it must be a normal MMBasic string or expression (not a long string). The search is case sensitive. Normally the search will start at the first character in 'str' but the optional third parameter allows the start position of the search to be specified.
LLEN(array%())	Returns the length of a long string stored in array%()
MM.DEVICE$	Returns "Pi-cromite running on H/W version: hhhhhh" where hhhhhh is the version ID of the Raspberry Pi
MM.DISPLAY(X)	Gives the current X cursor position on the console (0<width in chars)
MM.DISPLAY(Y)	Gives the current Y cursor position on the console (0<height in chars)
MM.DISPLAY(XMAX)	Gives the width in chars of the console
MM.DISPLAY(YMAX)	Gives the height in chars of the console
MM.FORK	Returns the process id (PID) of a forked process. Returns 0 if the process has terminated
SPRITE(…)	The function is available on VGA, SSD1963 (pin RD must be connected and specified) and ILI9341 (SPI-IN must be connected) displays. In addition there is an optimised version for 16-bit parallel ILI9341 displays (code ILI9341P16)
SPRITE(C, [#]n)	Returns the number of currently active collisions for sprite n. If n=0 then returns the number of sprites that have a currently active collision following a SPRITE SCROLL command
SPRITE(C, [#]n, m)	Returns the number of the sprite which caused the “m”th collision of sprite n. If n=0 then returns the sprite number of “m”th sprite that has a currently active collision following a SPRITE SCROLL command
SPRITE(H,[#]n)	Returns the height of sprite n. This function is active whether or not the sprite is currently displayed (active).
SPRITE(L, [#]n)	Returns the layer number of active sprites number n
SPRITE(N)	Returns the number of displayed (active) sprites
SPRITE(N,n)	Returns the number of displayed (active) sprites on layer n
SPRITE(S)	Returns the number of the sprite which last caused a collision. NB if the number returned is Zero then the collision is the result of a SPRITE SCROLL command and the SPRITE(C…) function should be used to find how many and which sprites collided.

SPRITE(W, [#]n)	Returns the width of sprite n. This function is active whether or not the sprite is currently displayed (active).
SPRITE(X, [#]n)	Returns the X-coordinate of sprite n. This function is only active when the sprite is currently displayed (active). Returns 10000 otherwise.
SPRITE(Y, [#]n)	Returns the Y-coordinate of sprite n. This function is only active when the sprite is currently displayed (active). Returns 10000 otherwise

Appendix A
Sensor Fusion

The Pi-cromite supports the calculation of pitch, roll and yaw angles from accelerometer and magnetometer inputs.

For information on this technology see https://github.com/kriswiner/MPU-6050/wiki/Affordable-9-DoF-Sensor-Fusion

The SENSORFUSION command supports both the MADGWICK and MAHONY fusion algorithms. The format of the command is:
SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]
Type can be MAHONY or MADGWICK
Ax, ay, and az are the accelerations in the three directions and should be specified in units of standard gravitational acceleration.
Gx, gy, and gz are the instantaneous values of rotational speed which should be specified in radians per second.
Mx, my, and mz are the magnetic fields in the three directions and should be specified in nano-Tesla (nT)

Care must be taken to ensure that the x, y and z components are consistent between the three inputs. So , for example, using the MPU-9250 the correct input will be ax, ay,az, gx, gy, gz, **my, mx, -mz** based on the reading from the sensor.

Pitch, roll and yaw should be floating point variables and will contain the outputs from the sensor fusion.

The SENSORFUSION routine will automatically measure the time between consecutive calls and will use this in its internal calculations.

The Madwick algorithm takes an optional parameter p1. This is used as beta in the calculation. It defaults to 0.5 if not specified

The Mahony algorithm takes two optional parameters p1, and p2. These are used as Kp and Ki in the calculation. If not specified these default to 10.0 and 0.0 respectively.

A fully worked example of using the code is given on the BackShed forum at

http://www.thebackshed.com/forum/forum_posts.asp?TID=9321&PN=1&TPN=1

Appendix B
Socket Use

This Appendix contains a fully worked example of code to implement a simple interactive web site in MMBasic. The code is for a remote thermostat which requires a security code to be entered before the thermostat setting can be changed. The code buffers incoming HTTP requests in a long string and then uses a generic parsing routine to interpret the request. The HTML file shows how to include MMBasic variables which will have current values substituted into the HTML when the page is transmitted

See http://www.thebackshed.com/forum/forum_posts.asp?TID=9601 for more information and examples of the use of CSS files and embedded pictures

```
option explicit
option default none
Const starttemp = 18
Const maxargs = 32
Const relaypin=7
Const off=0
Const on=1
Dim a%(1000),i%
Dim integer sp(11)
Dim s$,pg$
Dim security$="123456"
Dim string cp(11)
Dim float tcurrent,tnew
Dim float tmax=-1000
Dim float tmin=1000
Dim arg$(1,maxargs-1)
Dim integer checked=0
Const check$="checked='checked'" 'string to set a readio button pressed
Dim string heating="off"
Const hon$="#ff0000" 'red
Const hoff$="#00ff00" 'green
Dim string hcol=hoff$
'
SetPin relaypin,dout
For i%=0 To 10
  sp(i%)=i%+starttemp-1 'set up the temperature values
  cp(i%)="" 'set up the radio buttons as not pressed
Next i%
tcurrent=TEMPR(11) 'get the current temperature
cp(checked)=check$
Open "socket,myint,100" As #2
Do
  TEMPR START 11
  Pause 2000
  tnew=TEMPR(11)
  If Abs(tnew-tcurrent)<10 Then tcurrent=tnew
  updateheater
  If tcurrent>tmax Then tmax=tcurrent
  If tcurrent<tmin Then tmin=tcurrent
Loop
'
Sub myint
  Local p%=0, t%=0
  Local g$
  Do While Not Eof(2)
    LongString append a%(),Input$(10,2)
  Loop
  p%=LInStr(a%(),"GET",1)
  t%=LInStr(a%(),"HTTP",1)
```

```
  If p%<>0 And t%<>0 Then 'full request received
    s$=LGetStr$(a%(),p%,t%-p%+4)
    LongString trim a%(),t%+4
    pg$= parserequest$(s$,i%)
    If i% Then
      If arg$(0,0)="Security" And arg$(1,0)=security$ Then 'valid update
        If arg$(0,1)="R" Or arg$(0,2)="R" Then
          cp(checked)=""
          If arg$(0,1)="R" Then checked=Asc(arg$(1,1))-Asc("A")
          If arg$(0,2)="R" Then checked=Asc(arg$(1,2))-Asc("A")
          cp(checked)=check$
          updateheater
        EndIf
        If arg$(0,1)="Check" Or arg$(0,2)="Check" Then
          tmin=tcurrent
          tmax=tcurrent
        EndIf
      EndIf
    EndIf
'
    If pg$="index.html" Then
      Transmit page "index.html"
    Else
      Transmit code 404
    EndIf
  EndIf
End Sub

'Function to parse an HTML GET request'
' Assumes that the request starts with "GET /"
' and ends with "HTTP"
'
Function parserequest$(req$, paramcount As integer)
  Local a$,b$
  Local integer inpos,startparam,processargs
  For inpos=0 To maxargs-1
    arg$(0,inpos)=""
    arg$(1,inpos)=""
  Next inpos
  paramcount=0
  a$=Mid$(req$,6,Len(req$)-10)
  inpos=Instr(a$,"?")
  If inpos<>0 Then 'parameters found
    processargs=1
    parserequest$=Left$(a$,inpos-1)
    a$=Mid$(a$,inpos+1)
    Do
       arg$(0,paramcount)=""
       arg$(1,paramcount)=""
       inpos=Instr(a$,"=")
       startparam=1
       arg$(0,paramcount)=Mid$(a$,startparam,inpos-startparam)
       startparam=inpos+1
       inpos=Instr(a$,"&")
       If inpos<>0 Then
                arg$(1,paramcount)=Mid$(a$,startparam,inpos-startparam)
                a$=Mid$(a$,inpos+1)
                paramcount=paramcount+1
       Else
          arg$(1,paramcount)=Mid$(a$,startparam)
          paramcount=paramcount+1
          processargs=0
       EndIf
    Loop While processargs
  Else
    parserequest$=a$
```

```
  EndIf
  If a$="" Then
    parserequest$="index"
  EndIf
  If Instr(parserequest$,".html")=0 And Instr(parserequest$,".HTML")=0 Then
  parserequest$=parserequest$+".html"
End Function

Sub updateheater
  Local float hcalc
  If checked=11 Then
    Pin(relaypin)=1
    heating="On"
    hcol=hon$
  EndIf
  If checked=0 Then
    Pin(relaypin)=0
    heating="Off"
    hcol=hoff$
  EndIf
  If checked>=1 And checked<=10 Then
    hcalc=checked+starttemp-1 'setpoint temperature
    If tcurrent>hcalc Then 'turn heating off
      Pin(relaypin)=0
      heating="Off"
      hcol=hoff$
    EndIf
    If tcurrent<hcalc Then 'turn heating on
      Pin(relaypin)=1
      heating="On"
      hcol=hon$
    EndIf
  EndIf
End Sub

<html>
<head>
    <title>Remote Thermostat</title>
</head>
<body>
    <form name='f1' method='get' action='index'>
        <h2 align='left'>Remote Thermostat V4.0</h2>
        Update Code: <input type='text' name='Security' size='6' value='000000'>
        <br>
        <p>
        <TABLE BORDER='1' CELLSPACING='0' CELLPADDING='5'>
            <TR>
                <TD>Heating</TD>
                <TD BGCOLOR='{HCOL}'>{heating}</TD>
            </TR>
        </TABLE>
        </p>
        <p>
        <TABLE BORDER='1' CELLSPACING='0' CELLPADDING='5'>
            <TR>
                <TD></TD>
                <TD>Temperature</TD>
            </TR>
            <TR>
                <TD>Current</TD>
                <TD>{TCURRENT}°C</TD>
            </TR>
            <TR>
                <TD>Max</TD>
                <TD>{TMAX}°C</TD>
            </TR>
```

```
            <TR>
                <TD>Min</TD>
                <TD>{TMIN}°C</TD>
            </TR>
        </TABLE>
        </p>
        <input name='Check' type='checkbox' value='Reset' onClick='this.form.submit()'>
        Reset Max/Min<p>
        <TABLE BORDER='1' CELLSPACING='0' CELLPADDING='5'>
            <TR>
                <TD>Thermostat</TD>
                <TD><input name='R' type='radio' {CP(0)} value='A' onClick='this.form.submit()'> Off<br></TD>
                <TD><input name='R' type='radio' {CP(1)} value='B' onClick='this.form.submit()'> {SP(1)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(2)} value='C' onClick='this.form.submit()'> {SP(2)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(3)} value='D' onClick='this.form.submit()'> {SP(3)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(4)} value='E' onClick='this.form.submit()'> {SP(4)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(5)} value='F' onClick='this.form.submit()'> {SP(5)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(6)} value='G' onClick='this.form.submit()'> {SP(6)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(7)} value='H' onClick='this.form.submit()'> {SP(7)}°C<br></
TD>
                <TD><input name='R' type='radio' {CP(8)} value='I' onClick='this.form.submit()'> {SP(8)}°C<br></T
D>
                <TD><input name='R' type='radio' {CP(9)} value='J' onClick='this.form.submit()'> {SP(9)}°C<br></T
D>
                <TD><input name='R' type='radio' {CP(10)} value='K' onClick='this.form.submit()'> {SP(10)}°C<br>
</TD>
                <TD><input name='R' type='radio' {CP(11)} value='L' onClick='this.form.submit()'> On<br></TD>
            </TR>
        </TABLE>
        </p>
    </form>
</body>
</html>
```

Appendix C
Code Examples

Capturing system output to a file

```
System  "ls -al >> myfile"
Open "myfile"   For input As #1
Do
  Line Input #1,a$
  Print a$
Loop While Not Eof(#1)
Close #1
Kill "myfile"
```

NB The >> will append to the file (and create it if required).
Instead you can use > and it will throw away any existing data then write the file.

Capturing system output to a long string

```
DIM INTEGER a(1000)
SYSTEM "ls -al",a()
FOR i=1 to LLEN(a())
  print LGETSTR$(a(),i,1);
NEXT i
```

Accessing a USB/UART from MMBasic

Thanks to TassyJim for the code snippet

```
  ' serial test program
  System "stty -F /dev/ttyUSB0 38400"
  OPEN "/dev/ttyUSB0" FOR random AS #5 ' open the serial port
  PRINT #5, "HELLO!!! Is anyone out there?"
  DO ' preferred way to receive serial data which might not be there!
    k$ = INPUT$(1,#5)
    IF k$="" THEN
      nodata=nodata+1
    ELSE
      result$=result$+k$
      PRINT ASC(k$),result$
      nodata=0
    ENDIF
    PAUSE 20
  LOOP UNTIL k$=CHR$(10)OR nodata=50 ' 50*20ms = 1 second timeout
  DO
    PRINT #5, "U";
  LOOP

  CLOSE #5
```

Appendix D
Using an I2S DAC with MMBasic

The pinout for I2S is:

Bit-clock : pin 12
LR-clock : pin 35
i2s-data-in : pin 38
i2s data-out : pin 40

I2S can be enabled with

```
sudo curl -sS https://get.pimoroni.com/phatdac | bash
```

Set the MMBasic to use the PWM clock with

```
OPTION CLOCK PWM
```

Reserve the I2S pins so that MMBasic can't use them with

```
OPTION PINS &HA400000800
```

Then an audio file can be played over i2s either external to mmbasic (and unaffected by it), or from mmbasic using any appropriate Linux command

```
SYSTEM "aplay wavfile"
```

To play mp3 or flac files over i2s I found mpv worked, omxplayer does not work

```
sudo apt-get install mpv
```

Appendix E
Using an HDMI display with MMBasic

Set up the display using OPTION LCDPANEL HDMI.

The code automatically reads in the display size and MM.HRES and MM.VRES are set automatically for you. The display must be configured in full 32-bit colour (RGBA).

The code works best on a Pi running Raspbian Lite as updating elements of the Pixel GUI will break through the Pi-cromite display (e.g. the clock and CPU meter) unless stopped

You can start and stop the PIXEL desktop with:

```
sudo systemctl stop lightdm
sudo systemctl start lightdm
```

And then disable the flashing cursor with

```
sudo su
setterm -cursor off > /dev/tty0
```

On a Lite install it works best if you disable the login prompt and the cursor

To disable the login prompt use

```
sudo systemctl disable getty@tty1.service
```

to disable the flashing cursor use:

```
sudo su
setterm -cursor off > /dev/tty1
```

As normally configured Raspbian turns off the display after a period without keyboard or mouse input (screen blanking)

To stop screen blanking on Lite

```
sudo nano /boot/cmdline.txt
```

Append " consoleblank=0" to the end of the line

To stop screen blanking on Pixel

```
sudo nano /etc/lightdm/lightdm.conf
```

in the "[Seat:*]" section insert the line

"xserver-command=X -s 0 dpms"

In both cases reboot for the change to take effect

Appendix F
Sprites

See the SPRITE commands and functions for syntax details.
The concept of the sprite implementation is as follows:

1. Sprites are full colour and of any size. The collision boundary is the enclosing rectangle.
2. Sprites are loaded to a specific number (1-50)
3. Sprites are displayed using the SPRITE SHOW command
4. For each SHOW command the user must select a "layer". This can be between 0 and 10.
5. Sprites collide with sprites on the same layer, layer 0, or the screen edge
6. Layer 0 is a special case and sprites on all other layers will collide with it
7. The SCROLL commands leave sprites on all layers except layer 0 unmoved
8. Layer 0 sprites scroll with the background and this can cause collisions
9. There is no practical limit on the number of collisions caused by SHOW or SCROLL commands
10. The sprite function allows the user to fully interrogate the details of a collision
11. A SHOW command will overwrite the details of any previous collisions for that sprite
12. A SCROLL command will overwrite details of previous collisions for ALL sprites
13. To restore a screen to a previous state sprites should be removed in the opposite order to which they were written "LIFO"

Because moving a sprite or, particularly, scrolling the background can cause multiple sprite collisions it is important to understand how they can be interrogated.

The best way to deal with a sprite collision is using the interrupt facility. A collision interrupt routine is set up using the SPRITE INTERRUPT command.

e.g. SPRITE INTERRUPT collision

The following is a pro-forma for identifying all collisions that have resulted from either a SPRITE SHOW command or a SCROLL command

```
'
' This routine demonstrates a complete interrogation of collisions
'
sub collision
  local integer i
' First use the SPRITE(S) function to see what caused the interrupt
  if sprite(S) <> 0 then 'collision of specific individual sprite
    'sprite(S) returns the sprite that moved to cause the collision
     print "Collision on sprite ",sprite(S)
     process_collision(sprite(S))
     print ""
'
  else '0 means collision of one or more sprites caused by background move
    ' SPRITE(C, 0) will tell us how many sprites had a collision
    print "Scroll caused a total of ",sprite(C,0)," sprites to have
collisions"
```

```
    for i=1 to sprite(C,0)
      ' SPRITE(C, 0, i) will tell us the sprite number of the "I"th sprite
      print "Sprite ",sprite(C,0,i)
      process_collision(sprite(C,0,i))
    next i
    print ""
  endif
end sub
' get details of the specific collisions for a given sprite
sub process_collision(S as integer)
  local integer i ,j
  'sprite(C, #n) returns the number of current collisions for sprite n
  print "Total of ",sprite(C,S)," collisions"
   for i=1 to sprite(C,S)
     ' SPRITE(C, S, i) will tell us the sprite number of the "I"th sprite
    j=sprite(C,S,i)
    if j=100 then
      print "collision with left of screen"
    else if j=101 then
      print "collision with top of screen"
    else if j=102 then
      print "collision with right of screen"
    else if j=103 then
      print "collision with bottom of screen"
    else
      print "Collision with sprite ",sprite(C,S,i) 'sprite(C, #n, #m)
returns details of the mth collision
    endif
  next i
end sub
```

Appendix G
Running a program at boot

Thanks to disco4now and google there is now a way of running a MMBasic program fully automatically at boot on a Pi - the steps are pretty. There may be better ways but this way works very reliably. The following assumes that the mmbasic executable is in the directory /home/pi. This is the default login directory for the username "pi".

First install the application "screen" on the pi

```
sudo apt-get install screen
```

Next create a file "start.bash" in the same directory as mmbasic. Put the following into the file

```
cd /home/pi
sudo ./mmbasic
```

Make the script executable with the command

```
chmod +x start.bash
```

Now we need to tell Raspbian to execute screen and start.bash on boot. Edit the file "/etc/rc.local" (sudo nano /etc/rc.local) and append the following line to the file

```
/usr/bin/sudo -u pi /usr/bin/screen -dmS screen /home/pi/start.bash
```

This tells the pi to run screen under the user "pi" and to execute "start.bash" under screen.

Next, we assume we want to run a specific MMBasic program so once it is developed save it as AUTORUN.BAS in the same directory as the mmbasic executable.

Run MMBasic as normal (sudo ./mmbasic) and set the autorun option

```
OPTION AUTORUN ON
```

exit mmbasic (ctrl-Z)

Finally reboot the pi (sudo shutdown -r now") and when it reboots your MMBasic program will be running - you can check this with the command "ps -aux"

To connect to your running program execute the command

```
screen -D -r
```

and the terminal will connect so you can see any output and stop the program and edit it if required.

The use of "screen" ensures that console output from MMBasic is properly handled and your program can run happily without a console connected.

One minor issue with screen is that if you paste your code into mmbasic with putty it sometimes will miss a character when pasting if hosted under screen. (especially if a large program). You can drop out of screen from the linux prompt using

```
exit
```

and run mmbasic without it during code development if pasting the code becomes a problem during development.