Colour Maximite 2

User Manual

MMBasic Ver 5.07.01

Geoff Graham

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

http://geoffg.net/maximite.html

and http://mmbasic.com

About

The Colour Maximite 2 was conceived and developed by Peter Mather (matherp on the Back Shed Forum) who also led the development project.

It is based on the Colour Maximite developed by Geoff Graham and uses the MMBasic interpreter written by Geoff Graham (http://geoffg.net).

A team of people from around the world assisted with testing, advice and developing some initial games and programs. These are Phil Boyce, Jim Hiley, Graeme Rixon, Robert Severson and Mauro Xavier.

Support

Support questions should be raised on the Back Shed forum (http://www.thebackshed.com/forum/Microcontrollers) where there are many enthusiastic Maximite and Micromite users who would be only too happy to help. The developers of both the Colour Maximite 2 and MMBasic are also regulars on this forum.

Copyright and Acknowledgments

The Maximite firmware and MMBasic is copyright 2011-2021 by Geoff Graham and Peter Mather 2016-2021.

1-Wire Support is copyright 1999-2006 Dallas Semiconductor Corporation and 2012 Gerard Sexton.

FatFs (SD Card) driver is copyright 2014, ChaN.

MOD file support was written by Jean François DEL NERO (hxcmod.c).

WAV, MP3, and FLAC file support are copyright 2019 David Reid.

PNG support is copyright 2005-2010 Lode Vandevenne and 2010 Sean Middleditch.

The editor and file manager are based on code copyright 2016 Salvatore Sanfilippo and documentation from paileyq@gmail.com

The turtle graphics support including the polygon fill algorithm are copyright Mike Lam 2015.

Marcel Rodrigues wrote the GIF decoder.

Maury Quijada wrote the image resize and image rotate code.

The compiled object code (the .bin file) for the Colour Maximite 2 is free software: you can use or redistribute it as you please. The source code is available via subscription (free of charge) to individuals for personal use or under a negotiated license for commercial use. In both cases go to http://mmbasic.com for details.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

This Manual

The author of this manual is Geoff Graham. It is distributed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Australia license (CC BY-NC-SA 3.0)

Contents

Introduction 4
Quick Start Tutorial 5
Colour Maximite 2 Generation 1 7
Colour Maximite 2 Generation 2 9
Hardware Features 11
Using MMBasic 16
File Manager 20
Full Screen Editor 22
Variables and Expressions 25
Subroutines and Functions 30
Basic Graphics 33
SD Card Support 38
Audio Output 42
Using the I/O pins 44
Special Device Support 47
Game Playing Features 50
Porting Programs 52
MMBasic Implementation Characteristics 55
Predefined Read Only Variables 56
Operators 59
Options 60
Commands 66
Functions 115
Obsolete Commands and Functions 131
Appendix A – Serial Communications 132
Appendix B – I2C Communications 135
Appendix C – 1-Wire Communications 137
Appendix D – SPI Communications 138
Appendix E – Sprites 140
Appendix F – Special Keyboard Keys 142
Appendix G – Loading the Firmware 144

Introduction

The Colour Maximite 2 is a small self contained computer inspired by the home computers of the early 80's such as the Tandy TRS-80, Commodore 64 and Apple II. It includes its own BASIC interpreter and powers up in under a second into the BASIC interpreter (there is no operating system to boot). The emphasis is on ease of use and, as a result, a first time user could enter a small program and have it running within minutes.

While the concept is borrowed from computers of the 80's the technology used is very much up to date. The CPU that powers the Colour Maximite 2 is an ARM Cortex-M7 32-bit RISC processor running at up to 480MHz with 2MB flash memory and 1MB RAM. This processor includes its own video controller and generates a VGA output at resolutions up to 1920x1080 pixels and with up to 24-bit colour.

The Colour Maximite is designed to be simple and fun. The assembly instructions and the firmware are completely free and the parts can be found from multiple sources. It can be assembled in an hour or two and will provide endless hours of fun.

The basic features of the Colour Maximite 2 are:

- **Low cost affordable fun**. The firmware (including the BASIC interpreter) is completely free. The main PCB is easy to assemble with thru hole components. The CPU and support circuits are contained on a fully assembled plug in board costing US$30. The firmware can be loaded using free software so a programmer or special equipment is not required to get started.
- **Instant startup** into the BASIC interpreter. Program space is 516KB, enough for huge and complex programs (typically 25,000 lines or more) while general RAM used for arrays and buffers is over 5MB (enough for enormous arrays).
- **Full featured BASIC interpreter** with double precision floating point, 64-bit integers and string variables, long variable names, arrays of floats, integers or strings with multiple dimensions, extensive string handling and user defined subroutines and functions. Typically it will execute a program at 270,000 lines per second.
- **Rock solid VGA output** (or HDMI with an inexpensive converter). With 15 program selectable video resolutions from 1920x1080 pixels to 240x216 pixels and up to 24-bit colour (16 million colours).
- **USB Keyboard and Mouse support.** The keyboard can be wireless (with a USB dongle) or wired and have US, FR, DE, or UK key mappings. Mouse can be USB or dual USB/PS2 types.
- **Stereo audio output** can play WAV, FLAC and MP3 files, computer generated music (MOD format) and robot speech and sound effects as well as generate precise sine wave tones.
- **A full screen editor** is built into the firmware. It includes advanced features such as colour coded syntax, search and copy, cut and paste to and from a clipboard with full mouse support. With one key press the program can be saved and run. If an error occurs another key press will return to the editor with the cursor placed on the line that caused the error.
- **Full support for SD cards** including editing and running programs on the SD card as well as opening files for reading, writing or random access. Cards up to 128GB formatted in FAT16, FAT32 or exFAT are supported and the files can also be read and written on personal computers running Windows, Linux or the Mac operating system. A graphical file manager is included in MMBasic.
- **Programs can be easily transferred** from another computer (Windows, Mac or Linux) using the SD card, XModem protocol or by streaming the program over the serial console input.
- **Extensive features for creating computer games.** These include multiple video planes, support for Blits and Sprites, the ability to create computer generated music, sound effects and computer generated speech. The Colour Maximite 2 includes full support for the Wii Nunchuk and Wii Classic games controllers.
- **Battery backed clock** will keep the correct time, even with the power disconnected.
- **Twenty eight input/output pins** with 12 capable of analog input. Built in support for an IR remote control and temperature and humidity sensors. Communications protocols include I2C, asynchronous serial, RS232, IEEE 485, SPI and 1-Wire. These can be used to communicate with many sensors (temperature, humidity, acceleration, etc) as well as for sending data to test equipment.
- **Power is 5 volts at 220mA** typical from an USB port or charger.

Quick Start Tutorial

The following assumes that you have built or purchased the Colour Maximite 2, loaded the firmware, attached a keyboard and VGA monitor, powered it up and tested that it runs. At that point you should have the command prompt (a greater than symbol ">") displayed on the monitor.

Command Prompt

Most interaction with MMBasic is done via the console at the command prompt (ie, the greater than symbol (>) on the console). On startup MMBasic will issue the command prompt and wait for some command to be entered. It will also return to the command prompt if your program ends or if it generated an error message.

When the command prompt is displayed you have a range of commands that you can execute. Typically these would list a program (LIST) or edit it (EDIT) or set some options (the OPTION command). Most times the command is just RUN which instructs MMBasic to run the program.

When entering a line at the command prompt the line can be edited using the arrow keys to move along the line, the Delete key to delete a character and the Insert key to switch between insert and overwrite. The up and down arrow keys will move through a list of previously entered commands which you can edit and reuse.

Finally the “Enter” key will cause MMBasic to execute whatever is showing at the command prompt.

Almost any command can be entered at the command prompt and this is often used to test a command to see how it works. A simple example is the PRINT command, which you can test by entering the following at the command prompt:

```
PRINT 2 + 2
```

and not surprisingly MMBasic will print out the number 4 before returning to the command prompt.

Here are a few more things that you can try out. What you type is shown in bold and the Colour Maximite's output is shown in normal text.

Try a simple calculation:

```
> PRINT 1/7
0.1428571429
```

See how much memory you have:

```
> MEMORY
   Flash:
      0K ( 0%) Program (0 lines)
    566K (100%) Free

   RAM:
      0K ( 0%) 0 Variables
      0K ( 0%) General
   24800K (100%) Free
```

What is the current time?

```
> PRINT TIME$
10:04:01
```

What is the current date?

```
> PRINT DATE$
25/04/2020
```

Count to 20:

```
> FOR a = 1 to 20 : PRINT a; : NEXT a
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Your First Program

To enter a program you can use the EDIT command which is described later in this manual. To get a quick feel for how it works, try this sequence:

- Make sure that you have a properly formatted SD card inserted into the SD card slot.
- At the command prompt type `EDIT "hello.bas"` followed by the ENTER key.
- The editor should start up and you can enter this line: `PRINT "Hello World"`
- Press the F1 key in your keyboard. This tells the editor to save your program and exit to the command prompt.
- At the command prompt type `RUN "hello.bas"` followed by the ENTER key.
- You should see the message: `Hello World`

Congratulations. You have just written and run your first program on the Colour Maximite 2. If you type EDIT again you will be back in the editor where you can change or add to your program.

Something More Complicated

A more interesting program would be to fill the screen with coloured bubbles.

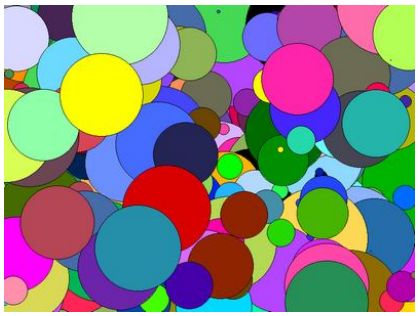
For this you need to know a little more about the editor. If you have used any full screen text editor in the past you will find the operation of this editor familiar. The arrow keys will move your cursor around in the text while the home and end keys will take you to the beginning or end of the line. The delete key will delete the character at the cursor and backspace will delete the character before the cursor.

To enter the bubbles program you should use the command `EDIT "bubbles.bas"` at the command prompt.

Then type in this short program:

```
DO
  r = RND * 255
  g = RND * 255
  b = RND * 255
  CIRCLE RND * 800, RND * 600, RND * 100,,, 0, RGB(r,g,b)
  PAUSE 5
LOOP
```

Press the F2 key which will save your program and automatically run it. You should see the screen continuously fill with hundreds of coloured bubbles as shown on the right.



If there was an error you will get a message with the line number and a description of the error. If you then re-enter the command EDIT you will be taken back into the editor with the cursor positioned on the line that caused the error. Correct the error and then save/run the program by pressing F2 again.

In this program we first set three variables (r, g and b) to random numbers in the range of zero to 255. The random number generator is called RND and it returns a random number in the range of zero to 0.999999. We multiply it by 255 to give us a random number from 0 to 255.

Then we draw a circle at a random position (again using the random number generator) with a random radius using the three colours previously calculated (ie, r, g and b). This code is contained within a DO…LOOP which instructs MMBasic to keep repeating this code (and drawing bubbles) forever.

You will notice that while this program is running you will not get the command prompt back. This is because MMBasic is now busy executing your program and drawing coloured bubbles. You can stop the program whenever you want by entering CTRL-C at the console and you should get the command prompt back again.

The purpose of the `PAUSE 5` command in the program is to slow down the program so that you have time to see the bubbles. To see how fast the Colour Maximite 2 can really go you could go back into the editor and change that line to `PAUSE 0` and then rerun the program.

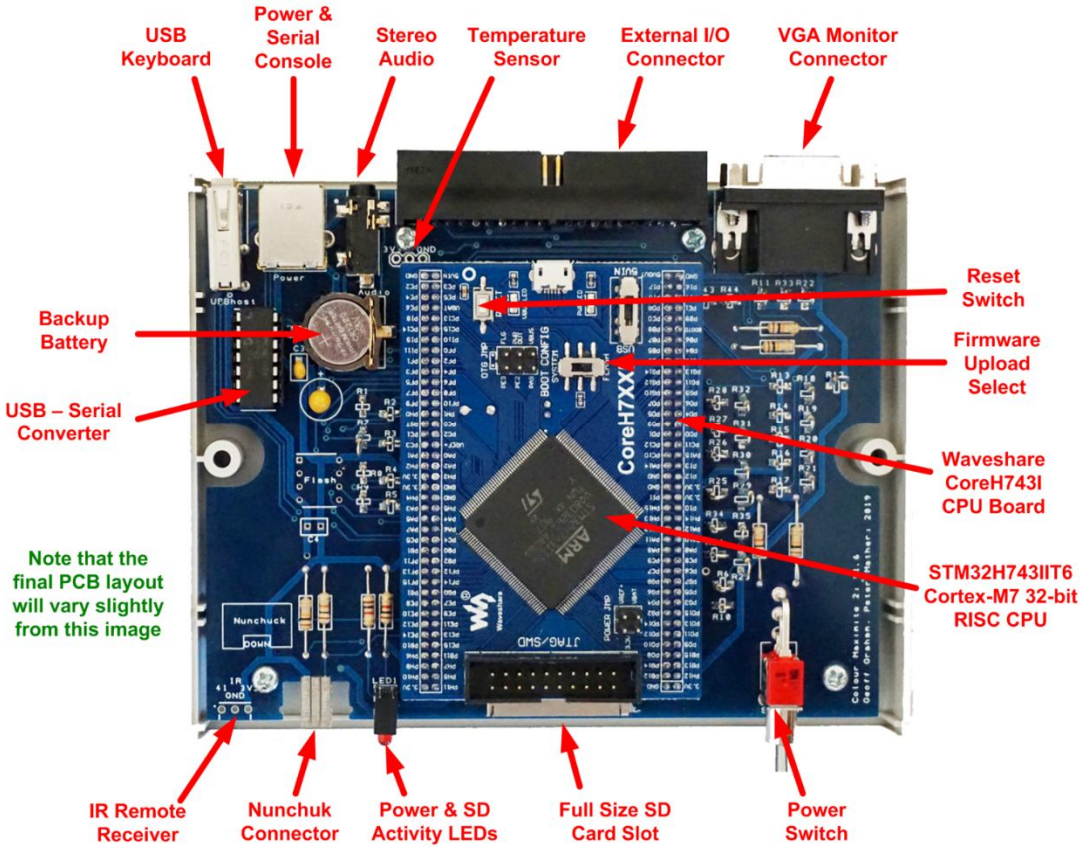
For a more in-depth tutorial and a description of programming in BASIC you should download and read "*Programming with the Colour Maximite 2*" which can be found at http://geoffg.net/maximite.html (scroll to the bottom of the page).

Colour Maximite 2 Generation 1

This version uses a plug in CPU module made by Waveshare. The module carries the ARM Cortex-M7 32-bit RISC processor, 8MB SDRAM and various support components. The plug in board concept eliminates the need to solder the ARM CPU with its 176 pins and 0.2mm gap between pins and is therefore suitable for home construction.

The CPU module has two connectors on its top, one a micro USB the other a multi pin header. Both of these are only needed for firmware development and are not used in normal operation as a BASIC computer. For normal operation all the jumpers on this board should be removed, the power switch should be set to "5VIN" and the BOOT CONFIG switch should be set to "Flash".

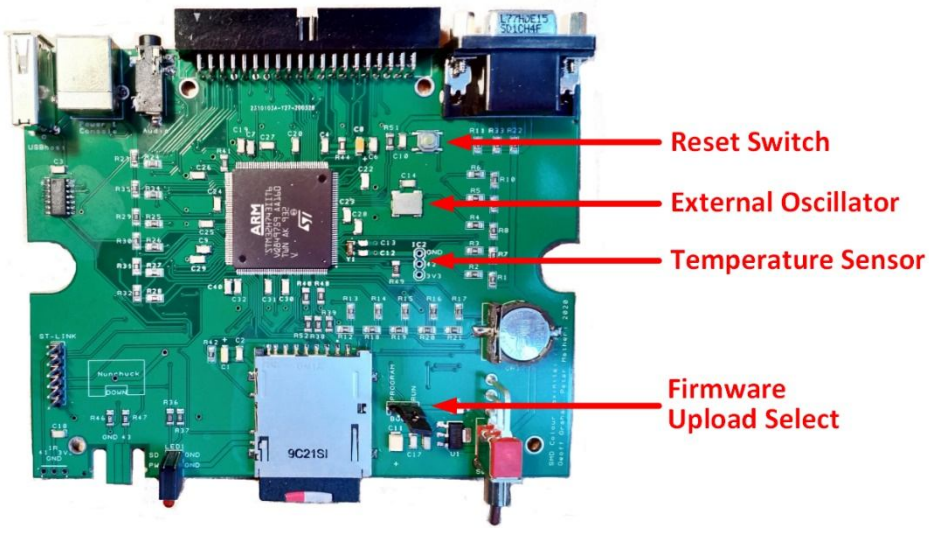
The following image illustrates the major connectors and features of this design:



Many suppliers are offering a fully assembled version of this design. These do not use the plug in module concept, instead the ARM Cortex-M7 processor and its supporting components are soldered directly to the main PCB. This makes sense for a machine assembled board and is cheaper to supply.

The layout of this board is essentially the same except that there are no jumpers as on the Waveshare board.

Some items are positioned differently as noted in the image to the right.



External Oscillator

The main processor clock comes from an 8 MHz crystal on the Waveshare board which is driven by the on chip oscillator in the ARM processor. This arrangement can have some jitter on the clock which in turn may cause jitter or instability in the VGA image. However, in most cases, this design works well with the maximum 800x600 pixel VGA video resolution provided by the early firmware releases.

However, with firmware upgrades, it is now possible to generate a VGA image at much higher resolutions and these may display an instability caused by this jitter. The solution is to fit an external 8MHz oscillator to replace the crystal and oscillator used by the Waveshare board. If the higher resolutions are not required then, in most cases, this modification will not be required. *This issue does not affect the single board versions (ie, fully assembled without the Waveshare module) as they use an external oscillator module.*

Note that the following will require a level of skill in soldering surface mount parts.

The following parts are required:

1 SMD 1206 100nF ceramic capacitor

1 8MHz oscillator in 5.0 x 7.0 mm SMD package (QX7 XO 25ppm). Eg, RS Stock Nbr 813-6194 .

The solder pads for these are located under the Waveshare board near the left hand 80 pin connector. The oscillator has a dot identifying pin 1 and this needs to be aligned with a small dot on the PCB silkscreen. Take care to not damage the 80-pin connector nearby and use the minimum of solder so that the solder does not short to the case of the oscillator. The capacitor should then be installed near the oscillator.

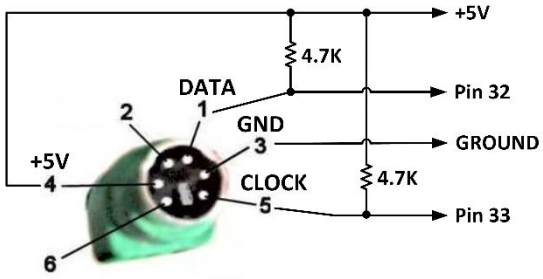
Nothing more is required and it is not necessary to remove the 8 MHz crystal from the Waveshare board. On power up the Colour Maximite 2 should operate as normal.

Mouse Interface

A PS2 mouse or a dual USB/PS2 mouse with a PS2 adapter can be connected via the rear I/O connector.

The mouse CLOCK (pin 5) connects to pin 33 of the I/O connector and mouse DATA (pin 1) connects to pin 32. Both must have 4.7KΩ pullup resistors to +5V.

The mouse GND and +5V pins should be connected to the ground and 5V pins on the I/O connector.



MMBasic has built in support for the mouse via the MOUSE() function. The mouse can also be used in the file manager and editor to select text, etc. A full description is provided in the next section of this manual.

The other features of this board are common with the Colour Maximite 2 Generation 2 and are described in the later section (*Hardware Features*).

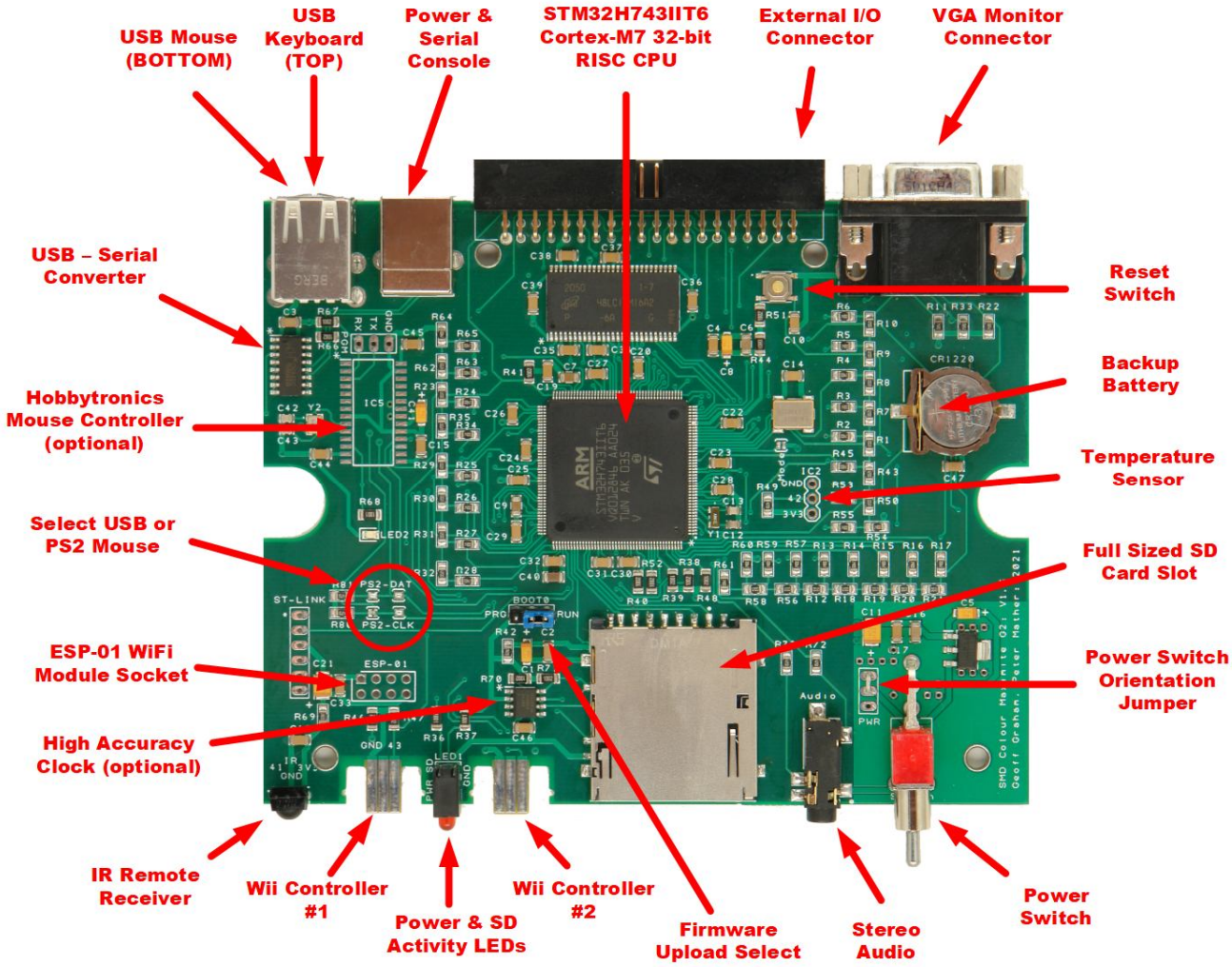
Colour Maximite 2 Generation 2

This is an upgraded design with three main improvements over the first generation.

1. It uses a larger RAM chip 32MB vs 8MB) which is used to support the top VGA resolution of 1920x1080 pixels, 24-bit colour depth and provide more RAM for BASIC programs.
2. An 8-bit digital to analog converter is used for each colour channel allowing the Generation 2 computer to display true colour 24-bit images.
3. The PCB is a 4 layers design with all components mount on the top side. This makes it more suitable for machine assembly.

This version also has connectors for a second Wii controller and a PS2 mouse and the rear I/O connector is inverted to make it more convenient to use connectors and cables designed to suit the Raspberry Pi Hats.

This diagram lists the key hardware features:



Mouse Interface

The Generation 2 design supports two types of mouse interface. The first is a full USB mouse and for this to be implemented a Hobbytronics mouse controller chip (https://www.hobbytronics.co.uk/usb-host-soic) needs to be installed at the position for IC5 along with the associated supporting components. The mouse then connects to the USB mouse socket (the lower socket) on the rear of the Colour Maximite 2.

The second type of support is for a dual USB/PS2 mouse. Most wired mice will automatically switch between USB and PS2 modes and many come with a USB to PS2 adapter. This adapter simply signals the mouse to switch to PS2 mode via a pull-up resistor and provides the physical PS2 connector. A typical example that has been tested on the Colour Maximite 2 is the Microsoft Basic Optical Mouse which is low cost and widely available.

To use a dual USB/PS2 mouse IC5 must **not** be populated, and all four solder jumpers marked "select USB or PS2 mouse" (diagram above) must be joined with a solder blob. The mouse then plugs into the USB mouse socket (the lower socket) on the rear of the Colour Maximite 2. A USB to PS2 adapter is **not** required (the Colour Maximite 2 will automatically switch the mouse into PS2 mode).

MMBasic has built in support for a mouse via the MOUSE() function. The mouse can also be used in the file manager and editor to select text, etc. A full description is provided in the next section of this manual.

Power Switch Orientation

Near the power switch there is a three pin jumper marked PWR which is used to select the orientation of the power switch. If the two pins towards the back of the Colour Maximite 2 are connected the power switch will operate as expected in the UK, Australia and New Zealand (down is on). If the two pins towards the front of the computer are connected the power switch will operate as expected in North America (up is on).

High Accuracy Real Time Clock

The basic design uses a miniature 32KHz crystal and the real time clock hardware built onto the ARM Cortex-M7 processor to keep the date and time. Optionally you can add an external DS3231MZ+ real time clock to the PCB (illustrated in the above photo) and it will be kept alive by the coin battery on the PCB. This clock will typically lose or gain only a couple of seconds in a week.

To enable this time source the command is: OPTION DS3231 ON

Once enabled the time and date can be set and retrieved from the DS3231MZ+ using the normal MMBasic DATE/TIME commands and functions.

ESP-01 Module

There is a footprint on the PCB for connecting an ESP-01 WiFi module to COM2. Version 5.07.00 and later of the firmware allows COM2 to be used as a serial console so if the correct USB/UART code is loaded into the ESP-01 it will be possible to access the console of the CMM2 remotely.

At this time MMBasic does not include any Internet functions using this device in but communications can be established using the normal serial communications protocol included in MMBasic.

The other features of the Generation 2 board are common with the first generation Colour Maximite 2 and are described in the following section (*Hardware Features*).

Hardware Features

The Colour Maximite 2 is based on a ARM Cortex-M7 32-bit processor which provides nearly all of the services for the user. This includes the flash memory (where the BASIC interpreter is installed), RAM and a sophisticated video display controller which generates the high quality VGA image with many advanced features useful to games programmers.

STM32H743IIT6 Cortex-M7 32-bit RISC CPU

This is a high performance 32-bit ARM processor with 2MB of flash and 1MB of RAM. It includes a sophisticated video display controller which is used to generate a stable VGA output. The MMBasic firmware is loaded onto the flash memory of this chip and provides the Colour Maximite 2 with its personality.

There are two versions of this chip, Rev Y with a maximum speed of 400MHz and Rev V with a speed of 480MHz. Most vendors do not allow you to specify the version in advance so, depending on the supply pipeline, you may get either. MMBasic will automatically accommodate either revision and you can tell which one you have with the MM.INFO(CPUSPEED) function which will return 400000000 for the Rev Y chip and 480000000 for the Rev V chip. Other than this there is no difference when used in the Colour Maximite 2.

VGA Monitor Connector

This is the main video output and it generates standard VGA signals in a variety of resolutions and number of colours as determined by the MODE command. These range from 800x600 pixels (the default at power up) up to 1280x1024 and down to 240x216 pixels with 256, 4096 or 65536 colours. Most modes work perfectly with monitors that have an aspect ratio of 4:3 or widescreen monitors that can switch to that ratio (most widescreen monitors will do this automatically). The Generation 2 Colour Maximite 2 can go to 1920x1080 pixels and 24-bit colour depth at some resolutions.

Note that on first setup some monitors may truncate the text on the margins or show an image that seems to shimmer or flicker. In most cases this can be fixed by pressing the auto setup button on the monitor or, failing that, using the monitor's image setup mode to adjust parameters such as the clock, phase and position.

If an HDMI output is required it is recommended that an inexpensive VGA to HDMI converter be used. These cost about US$10+ on eBay and will also encode the audio from the computer so that it can play through the monitor's speakers. As an example, the Colour Maximite 2 was successfully tested with this converter: https://tinyurl.com/w5woobe

USB Keyboard Connector

This is a Type A USB connector. It will accept a standard USB keyboard including most that have a wireless dongle for the USB connection. Note that you cannot use a USB hub on this port or keyboards that have both a keyboard and mouse function. The latter will not work because they have a built-in USB hub to support the two different functions.

Typical keyboards that have been tested and work include Logitech K120 or K270 or K400+ or K800, HP SK2885, Lenovo KU-0225 and Microsoft 600

When the keyboard is connected or on startup the Colour Maximite 2 will enumerate the keyboard and if this is successful the SD card activity LED will be illuminated. On the first access of the SD card the LED will revert to its normal action (illuminated during SD card access).

The firmware can accommodate a number of different language layouts including US, French (FR), Spanish (ES), Belgium(BE) and German (DE). This detail is requested by MMBasic on the first startup of the computer and can be later changed using the OPTION USBKEYBOARD command.

Power & Serial Console Connector

This USB B connector is for power and the serial console over USB. The power requirement is 5V at 160mA to 250mA (typical). This is within the capabilities of most USB chargers however some PCs (especially older laptops) may have trouble supplying this If your Colour Maximite 2 is suffering from intermittent issues such as reboots, errors reading the SD card, etc then it would be worth changing the power supply to one a with a much higher capacity (for example, 2 amps or more).

The serial console is available if the Colour Maximite 2 is connected via this port to a personal computer. The console is used to write and debug BASIC programs and configure the computer. Normally the VGA screen and USB keyboard are used as the console but the serial console works just as well including using the File Manager and Full Screen Editor. The only thing that the serial console does not support is graphics.

When connected to your desktop computer via USB the Colour Maximite 2 will be setup as a virtual serial port over USB and appear to your computer as a normal serial port.

Windows 10 includes the required USB device driver. For other operating systems (ie, Linux, Mac, Windows 8 and earlier) go to the Microchip website https://www.microchip.com/wwwproducts/en/MCP2221A for instructions (select the Documents tab). The second generation design uses a CH340 USB/UART chip and support for this is included in most operating systems,

In Windows the Colour Maximite 2 will appear in Device Manager as "USB Serial Port" as illustrated on the right (the COM number will probably be different).

You also need a terminal emulator program on your desktop computer. This program acts like an old fashioned computer terminal where it will display text received from a remote computer and any key presses will be sent to the remote computer over the serial link.

The terminal emulator that you use should support VT100 emulation as that is what the editor built into the MMBasic expects. For Windows users it is recommended that you use Tera Term as this has a good VT100 emulator and is known to work with the XModem protocol which you can use to transfer programs to and from the Colour Maximite (Tera Term can be downloaded from: http://tera-term.en.lo4d.com/).

The terminal emulator and the serial port that it is using should be set to the Colour Maximite 2 standard of 115200 baud, 8 data bits and one stop bit.

The Apple Macintosh (OS X) is somewhat easier as it has the device driver and terminal emulator built in. First start the application 'Terminal' and at the prompt list the connected serial devices by typing in:

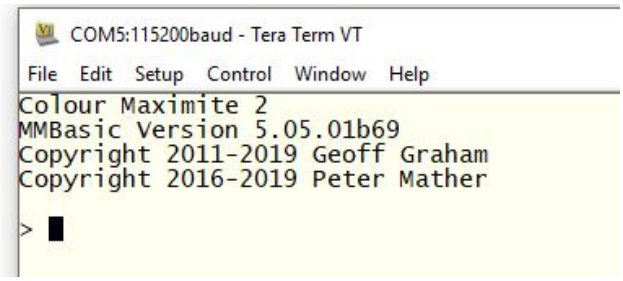
```
ls /dev/tty.*.
```

The USB to serial converter will be listed as something like `/dev/tty.usbmodem12345`. While still at the Terminal prompt you can run the terminal emulator at 115200 baud by using the command:

```
Screen /dev/tty.usbmodem12345 115200
```

Instructions for Linux are here: http://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=12171

When you have the serial port and terminal emulator setup you can reset the Colour Maximite 2 and you should see the MMBasic banner and prompt on the terminal emulator as illustrated on the right.



The USB to Serial converter chip used in the original Colour Maximite 2 can be a Microchip MCP2221A. By default this chip tells the host that it only needs 100mA on the 5V pin and in some rare cases this can cause trouble (ie, failure to power up, random restarts, etc). The fix is to go to https://www.microchip.com/wwwproducts/en/MCP2221A and download the "MCP2221 Utility" (it is under the Documents tab) and use that to change the MCP2221A to request 500mA by default.

Mouse Support

To use the mouse in the file manager and editor it needs to be configured as follows (use channel 0 for a PS2 mouse): `OPTION MOUSE channel [,sensitivity]`

The optional sensitivity parameter allows you to tune the sensitivity of the Hobbytronic interface by specifying a number between 0 and 10. Zero indicates that the default automatic sensitivity should be used while other values will explicitly set the sensitivity as per the Hobbytronic documentation.

For a PS2 mouse the sensitivity can be set between 0 and 8. Values of 1-4 set the resolution of the mouse to 1,2, 4 or 8 counts/mm. Values 5-8 do the same but in addition enable mouse scaling which gives a non-linear relationship between speed and count (note that not all PS2 mice implement these settings).

To disable the mouse in the editor and file manager use: OPTION MOUSE OFF

Within the file manager and editor the mouse works as you would expect. For example, you can use the mouse to point and select a file or line, you can select text with the mouse, scroll using the scroll wheel, double click to open/run a file and so on. For the details see the chapters on the file manager and editor.

Within a MMBasic program the mouse can be enabled for access with various sensitivity settings and interrupts on left and/or right clicks. Specialised built in functions can be used to determine the current mouse position in pixels and detect single or double clicks.

Audio Connector

This is a 3.5mm stereo phono socket. The tip is the left channel, the ring is the right channel while the sleeve is ground. The signal level at full volume is about 1V RMS (approx 3V peak to peak). MMBasic can generate audio in many formats ranging from simple sine wave tones through to playing WAV, FLAC, and MP3 files.

Note:

- The output is high impedance suitable for feeding into an amplifier. It cannot directly drive a loudspeaker, headphones or any low impedance load **and might be damaged if that was attempted.**
- If the audio is garbled and/or lacks normal bass it might indicate that your amplifier has a low input impedance. Try adding a 4.7K resistor in series with the output (the Generation 2 board has these permanently installed).
- There is a DC offset on the output. Most amplifiers have an input capacitor so this has no effect but, if not, it may be necessary to add a 4.7μF capacitor in series with the output (positive leg to the CMM2).

Power & SD Card Activity LEDs

The power indicator (green) is illuminated whenever power is applied.

The SD Card activity indicator (red) will illuminate briefly when reading or writing from/to the SD card. The SD card should **not** be removed when this LED shows activity. The red LED will also illuminate when a USB keyboard is plugged in and successfully communicates with MMBasic – it then reverts to normal activity on the first read/write to the SD card.

SD Card Connector

The SD card connector on the front panel is the main storage for the Colour Maximite 2. All programs reside on the SD card so it must be present for most operations. This is different from the original Colour Maximite where the SD card was not necessarily required.

The Colour Maximite 2 will support cards up to 128GB. Cards larger than 32GB should be formatted as exFAT and cards 32GB or less formatted as FAT32. Note that very small cards may not be reliable and exFAT is much slower than FAT32. BASIC programs are quite small so an 8GB or 16GB card formatted as FAT32 is ample and also readily available at a cheap price.

External I/O connector

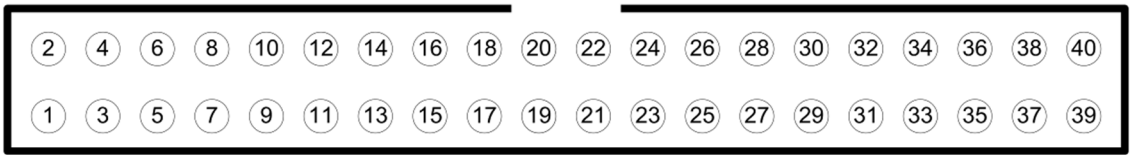
This 40-pin ribbon connector provides 28 input/output pins which can be controlled from within the BASIC program plus 3.3V and 5.0V outputs for powering external circuitry plus a number of ground pins.

The total current drawn from all 3.3V pins should be limited to 100mA. Exceeding this may cause the voltage regulator on the CPU module to enter thermal shutdown. The capacity of the 5V supply is limited only by your USB power supply.

If more input/output pins are required an I/O expander such as the Microchip MCP23017 can be used.

Colour Maximite 2 Generation 1

This is the image of the connector as seen from the rear of the <u>first generation Colour Maximite 2</u>:



These numbers are used as reference in the table below and by MMBasic to refer to an I/O pin.
They are not the connector manufacturer's standard pin numbers.

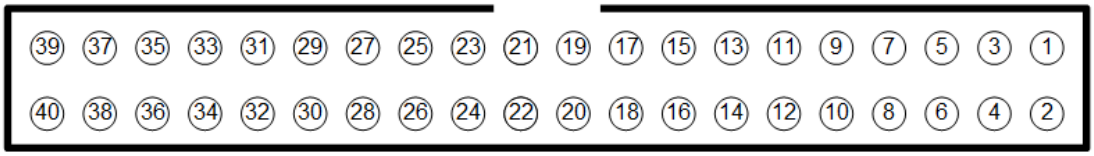
The pin layout and positioning of special functions on the external I/O connector is compatible with the Raspberry Pi allowing Pi HATs to be connected if required.

Note that most Raspberry Pi cables for use with solderless breadboards need to be inverted when plugged into the Colour Maximite 2's rear I/O connector (ie, the key tab on the connector needs to be on the bottom). If this is not done the pin legends on the breadboard connecter will be wrong. This is not a problem if the Maximite connector is unshrouded otherwise you may need to cut a new keyway on the bottom of the shroud with side cutters (not hard to do).

Colour Maximite 2 Generation 2

The second generation Colour Maximite 2 has these pins reversed so that the issue described above (inverted connector) is avoided. On this version Raspberry Pi cables can plug directly in – there is no need to cut a second keyway.

This is the image of the connector as seen from the rear of the second generation Colour Maximite 2:



These are the connector manufacturer's standard pin numbers.
These numbers are used as reference in the table below and by MMBasic to refer to an I/O pin.

I/O Pin Capabilities

The capabilities of each pin are:

Pin	Features	
1	3.3 Volt Power	
3	I^2C SDA	5V
5	I^2C SCK	5V
7	Analog Input or COUNT 1	
9	Ground	
11	COM2: RX	5V
13	Analog Input or COUNT 2	
15	Analog Input or COUNT 3	
17	3.3 Volt Power	
19	SPI MOSI	5V
21	SPI MISO	5V
23	SPI CLOCK	5V
25	Ground	
27	I^2C2 SDA	5V
29	Analog Input or PWM-1C	
31	PWM 2B	5V
33	General I/O	5V
35	SPI2 MISO	5V
37	Analog Input or COM1 DE	
39	Ground	

Pin	Features	
2	5.0 Volt Power	
4	5.0 Volt Power	
6	Ground	
8	Analog Input or COM1: TX	
10	Analog Input or COM1: RX	
12	Analog Input or PWM 1A	
14	Ground	
16	Analog Input or COM2: TX	
18	FAST COUNT	5V
20	Ground	
22	Analog Input or PWM 1B	
24	Analog Input or COUNT 4	
26	Analog Input	
28	I^2C2 SCK	5V
30	Ground	
32	General I/O	5V
34	Ground	
36	PWM 2A	5V
38	SPI2 MOSI	5V
40	SPI2 CLOCK	5V

Except for the power and ground pins (shown as shaded grey) all pins can be used for digital I/O using the PIN() function and command and using the pin number as the reference. For example pin 3 can be set to an output using SETPIN 3, DOUT and then the pin set high (ie, to 3.3V) using the command `PIN(3) = 1`.

All pins marked 5V can tolerate inputs up to 5.3V all other pins can tolerate up to 3.6V.

Pin 40 can be used to completely reset the Colour Maximite 2 to its "factory default" condition. If that pin is connected to ground (ie, pin 39) on power up all options will be reset to their defaults and any program in program memory erased. Note that external circuitry connected to this pin (eg, a capacitor) must not look like a short circuit at power up as this might trigger a reset.

There are 12 pins marked as supporting analog input and these can be used for measuring voltages. The other special capabilities (eg, COM2: RX, etc) are described in the relevant section of this manual.

Both the data line (SDA) and clock (SCL) for both I^2C ports have 10K pullup resistors (to 3.3V) installed on the motherboard so external pullup resistors are not required. Note that these may interfere with the operation of these pins if they are used as general purpose inputs.

Wii Nunchuk and Wii Classic Connector

The Colour Maximite 2 includes connectors on the front panel for either the Wii Nunchuk (illustrated on the right) or the Wii Classic game controllers and MMBasic includes commands and functions to work with both of them. Many games written for the Colour Maximite 2 will use either controller to control the game play so they are a useful addition if you plan on playing some games.

For more details on the Nunchuk and Classic search the Internet for "Nintendo Nunchuk" or "Nintendo Classic". Clones of both of these can be purchased cheaply for US$10 to US$40.

The connector used by the Wii controllers has a large plastic clip on top and this must be uppermost when the plugged in. Take care as it is possible to insert the connector upside down and that may damage the controller or the Colour Maximite.

Temperature Sensor

These solder pads are for an optional Dallas DS18B20 temperature sensor.

Support for the DS18B20 is built into MMBasic – see the section *Special Device Support* in this manual for the details. The signal line for this sensor is pin 42. If this sensor is installed the associated pullup resistor (4.7KΩ) on the motherboard must also be installed.

IR Receiver

These solder pads are for an IR remote control receiver which will allow the Colour Maximite 2 to be controlled via a standard NEC or Sony infra red remote control transmitter. See the section *Special Device Support* for details of the receivers that can be fitted and how to use them in your program. The signal pin for the receiver can also be used as a digital I/O pin as pin 41.

Firmware Upload Select

This is either a switch or a jumper. Normally the switch should be set in the "Flash" position or, if a jumper is used, it should be in the “RUN” position.

When uploading new firmware the switch is set in the "System" position and the jumper should be moved to the “PRG” position. See *Appendix G - Loading the Firmware* at the end this manual for the details.

Backup Battery

The CR1220 coin cell battery on the motherboard keeps the internal ARM Cortex-M7 real time clock running while the power is off and also keeps a bank of 4KB RAM alive at the same time. The real time clock is used to provide the correct time to MMBasic on startup and the battery backed RAM is used to store saved variables and options. The life of this battery life will be about 3 to 4 years of normal use.

Reset Switch

This is used to reset the Colour Maximite 2 and start the bootup sequence as if the power had been cycled.

Using MMBasic

Commands and Program Input

At the command prompt you can enter a command and it will be immediately run. Most of the time you will do this to run a program or set an option. But this feature also allows you to test out commands at the command prompt.

To enter a program the easiest method is to use the EDIT command. This will invoke the full screen editor which is built into the Maximite and is described later in this manual. It includes advanced features such as search and copy, cut and paste to and from a clipboard.

You could also compose the program on your desktop computer using something like Notepad and then transfer it via the XModem protocol (see the XMODEM command) or by streaming it up the console serial link (see the AUTOSAVE command) or by saving it to an SD card and transferring that to the Colour Maximite 2.

A fourth and convenient method of writing and debugging a program is to use MMEdit. This is a program running on your Windows computer which allows you to edit your program on your computer then transfer it via the serial console with a single click of the mouse. MMEdit was written by Jim Hiley and can be downloaded for free from https://www.c-com.com.au/MMedit.htm

One thing that you cannot do is use the old BASIC method of entering a program which was to prefix each line with a line number. Line numbers are optional in MMBasic so you can still use them if you wish but if you enter a line with a line number at the prompt MMBasic will simply execute it immediately.

Editing the Command Line

When entering a line at the command prompt the line can be edited using the arrow keys to move along the line, the Delete key to delete a character and the Insert key to switch between insert and overwrite. The up and down arrow keys will move through a history of previously entered commands which can be edited and reused.

Console Keyboard/Display

Input can come from either the USB keyboard or from a computer using a terminal emulator via the USB serial interface. Both the keyboard and the serial over USB can be used simultaneously and can be detached or attached at any time without affecting a running program.

The text output from MMBasic will be simultaneously sent to the VGA monitor and the serial over USB interface however the graphics commands operate on the video output only.

This behaviour can be changed with the OPTION CONSOLE command. Using this it is possible to turn either console off or on and save the setting so that it will be automatically applied on reboot.

Keyboard Shortcuts

The function keys on the keyboard or the serial console can be used at the command prompt to automatically enter common commands. The first four function keys (F1 to F4) will insert the text followed by the Enter key so that the command is immediately executed:

F1	FILES
F2	RUN
F3	LIST
F4	EDIT

Function keys F5 to F10 will insert the text then position the cursor between the quote marks at the end so that the file name can be directly entered. Pressing Enter will then execute the command:

F5	AUTOSAVE ""
F6	XMODEM RECEIVE ""
F7	XMODEM SEND ""
F8	EDIT ""
F9	LIST FILE ""
F10	RUN ""

Function keys F11 and F12 can be programmed with custom text:

F11	*User specified string – See the* OPTION F11 *Command.*
F12	*User specified string – See the* OPTION F12 *Command.*

Line Numbers and Program Structure

The structure of a program line is:

```
[line-number]  [label:]  command arguments [: command arguments] …
```

A label or line number can be used to mark a line of code.

A label has the same specifications (length, character set, etc) as a variable name but it cannot be the same as a command name. When used to label a line, the label must appear at the beginning of a line but after a line number (if used) and be terminated with a colon character (:).

Commands such as GOTO can use labels or line numbers to identify the destination (in that case the label does not need to be followed by the colon character). For example:

```
GOTO xxxx
  - - -
xxxx:  PRINT "We have jumped to here"
```

Multiple commands separated by a colon can be entered on the one line (as in `INPUT A : PRINT B`).

All Programs Are Run From the SD Card

On the Colour Maximite 2 all programs reside on the SD card which acts as the "disk drive" for the computer. As a result the SD card must be present for most operations. This is different from the original Maximite where the SD card was not necessarily required.

When you edit a program you are editing the program on the SD card, when you run a program you will run it from the SD card, etc. The reason for this arrangement is that when a program is loaded into memory for execution MMBasic will do a lot of pre-processing to speed up execution. This includes inserting any include files specified in the source, stripping out all comments, removing unnecessary spaces and so on. The resultant program is then saved in program memory but it cannot be edited or listed because after pre-processing the executable program is not easily human readable.

The benefit of this arrangement is a marked improvement in the speed of execution and that much larger programs can fit into program memory and be run.

The main commands used to manage a program are:

RUN "*prog*"	Run the program called *prog* located on the SD card.
LIST "*prog*"	List the program called *prog* on the console screen. This will pause every screen full and any key press will continue the listing.
EDIT "*prog*"	Edit the program called *prog* located on the SD card.

For example: `RUN "hello.bas"`

Note that the file name must be surrounded by double quotes as shown above. This is because the file name is a string and in MMBasic all string constants (ie, not a variable) must be quoted. In all cases the file extension ".BAS" will be automatically added if an extension was not specified in the command line.

When RUN or EDIT are used they set what is known as the *current program name*. This is the file name that will be used if the commands RUN, EDIT and LIST are used without specifying a file name. For example, you could use the command EDIT "MyProg.bas" and that will set the current program name to "MyProg.bas". From then on you could use RUN, EDIT and LIST without a file name and they will refer to "MyProg.bas" on the SD card.

To clear the current program name and erase the processed program held in program memory you can use the command NEW. This also clears all variables, closes all files, etc (ie, resets MMBasic).

There are three other commands that operate on program files. These are AUTOSAVE, LIST ALL and XMODEM. These are used for sending/receiving programs via the serial console to or from the SDcard. AUTOSAVE will also update the current file name so that after a file has been transferred the RUN command without a file name will run that program. Note that the Colour Maximite 2 does not have the commands LOAD or SAVE as they are not required.

Finally, all commands referred to above (with the exception of RUN) can be used with a different file extension to operate on files that are not programs. For example, EDIT "data.txt" will edit the text file on the SD card called "data.txt". In this case the current program name will not be changed.

Status Line

When at the command prompt MMBasic will display a status line at the bottom of the VGA screen. On the left side this will show the current directory on the SD card while the text in the centre is the current program name (ie, the file that will be used if the commands RUN, EDIT and LIST are used without specifying a file name). On the right side it shows the current time and date.

The status line can be turned off with the OPTION STATUS OFF command.

Running Programs

A program is set running by the RUN command. You can interrupt MMBasic and the running program at any time by typing CTRL-C on the console input and MMBasic will return to the command prompt.

The running program is normally held in non volatile flash memory. This means that it will not be lost if the power is removed and, if you have the AUTORUN feature turned on, the program will automatically run when power is restored (use the OPTION command to turn AUTORUN on). Normally an SD card holding the original program must be present in order to run a program but this is one of the exceptions and allows you to change the SD card for recording data, etc.

Expressions

In most cases where a number or string is required you can also use an expression. For example:

```
FNAME$ = "TEST"
LIST FNAME$ + ".BAS"
```

The RUN command is the only exception, in this case the filename argument must be a string constant surrounded by double quotes (ie, not an expression).

Standards and Compatibility

MMBasic implements a large subset of Microsoft's GW-BASIC. There are numerous small differences due to physical and practical considerations but most ordinary BASIC commands and functions are essentially the same. An online manual for GW-BASIC is available at http://www.antonis.de/qbebooks/gwbasman/index.html and this provides a more detailed description of the commands and functions.

MMBasic also implements a number of modern programming structures documented in the ANSI Standard for Full BASIC (X3.113-1987) or ISO/IEC 10279:1991. These include SELECT CASE, SUB/END SUB, the DO WHILE … LOOP and structured IF .. THEN … ELSE … ENDIF statements.

The SELECT CASE commands allow the programmer to create a clear and structured decision tree that is more flexible and easier to understand when multiple decisions must be made. The DO WHILE … LOOP command make it easy to build loops without using the GOTO statement. User defined subroutines and functions make it easy to add your own commands to MMBasic.

The IF… THEN command can span many lines with ELSEIF … THEN, ELSE and ENDIF statements as required and also spaced over many lines.

Saved Variables

Data is normally saved by the program to the SD card but sometimes there is a need to save a small amount of data which is independent of the SD card and will survive a power failure, reboot, etc. This data might include menu choices, calibration data and configuration information.

This can be done with the VAR SAVE command which will save the variables listed on its command line in non volatile memory. These variables can be restored with the VAR RESTORE command which will add all the saved variables to the variable table of the running program. Normally VAR RESTORE is placed near the start of a program so that the variables are ready for use by the program.

The space reserved for saved variables is 4KB. This is a RAM memory which is kept alive by the battery on the motherboard. Writing to this memory is near instantaneous and data can be written an unlimited number of times without degradation (unlike with the Micromite).

Timing

You can get the current date and time using the DATE$ and TIME$ functions and you can set them by assigning the new date and time to them. The Colour Maximite 2 has a battery backed clock so it will not lose the time even when powered off. If you find that the time drifts while the power is off you can use the OPTION RTC CALIBRATE command to correct for any inaccuracies.

You can freeze program execution for a number of milliseconds using PAUSE. MMBasic also maintains an internal stopwatch function (the TIMER function) which counts up in microseconds. You can reset this timer to zero or any other number by assigning a value to the TIMER.

Using SETTICK you can setup up to four "ticks" which will generate regular interrupts with a period from one millisecond to over a month.

Watchdog Timer

It is possible to use the Colour Maximite 2 without a VGA monitor and also have nothing connected to the serial console. With OPTION AUTORUN ON set the program will run automatically on power up without human intervention.

However there is the possibility that a fault in the program could cause MMBasic to generate an error and return to the command prompt. This would be of little use in this situation as there would be nothing connected to the console. Another possibility is that the BASIC program could get itself stuck in an endless loop for some reason. In both cases the visible effect would be the same, the program would stop running until the power was cycled.

To guard against this the watchdog timer can be used. This is a timer that counts down to zero and when it reaches zero the processor will be automatically restarted (the same as when power was first applied), this will occur even if MMBasic is sitting at the command prompt. Following the restart the automatic variable MM.WATCHDOG will be set to true to indicate that the restart was caused by a watchdog timeout.

The WATCHDOG command should be placed in strategic locations in the program to keep resetting the timer and therefore preventing it from counting down to zero. Then, if a fault occurs, the timer will not be reset, it will count down to zero and the program will be restarted (assuming the AUTORUN option is set).

PIN Security

Sometimes it is important to keep the data and program confidential. In the Colour Maximite 2 this can be done by using the OPTION PIN command. This command will set a pin number (which is stored in non volatile memory) and whenever MMBasic returns to the command prompt (for whatever reason) the user at the console will be prompted to enter the PIN number. Without the correct PIN the user cannot get to the command prompt and their only option is to enter the correct PIN or reboot the computer. When it is rebooted the user will still need the correct PIN to access the command prompt.

Because an intruder cannot reach the command prompt they cannot list or copy the program held in the program memory, they cannot change the program or change any aspect of MMBasic. Once set the PIN can only be removed by providing the correct PIN as set in the first place. If the number is lost the only method of recovery is to reset MMBasic as described below (which will erase the program).

Note that this is not a complete protection as it is possible to connect a debugger to the ARM Cortex-M7 CPU and access all areas of memory using that facility but it does deter casual access.

The Serial Console

The default settings for the serial over USB console are 115200 baud, 8 bits, no parity and one stop bit. Using the OPTION BAUDRATE command the baud rate of the serial console can be changed to any other speed. Changing the console baud rate to a higher speed makes the full screen editor faster in redrawing the screen.

Once changed the console baud rate will be permanently remembered unless another OPTION BAUDRATE command is used to change it. Using this command it is possible to accidently set the baud rate to an invalid speed and in that case the only recovery is to connect a VGA monitor and USB keyboard to change the baudrate or reset MMBasic as described below.

If the serial console is not required it can be disabled with the command OPTION CONSOLE SCREEN. This is reset when the program ends. If you want to permanently disable the serial console then the above command should be followed with OPTION CONSOLE SAVE. Disabling the serial console has two advantages:

- The built in editor will operate faster as it does not need to echo the edited text on the slow serial console.
- The serial port used by the serial console can be opened as COM3:.

Resetting MMBasic

MMBasic can be reset to its original configuration by placing a link between pins 39 and 40 on the rear I/O connector while power is applied. This will result in the program memory and saved variables being completely erased and all options (security PIN, console baud rate, etc) reset to their initial defaults.

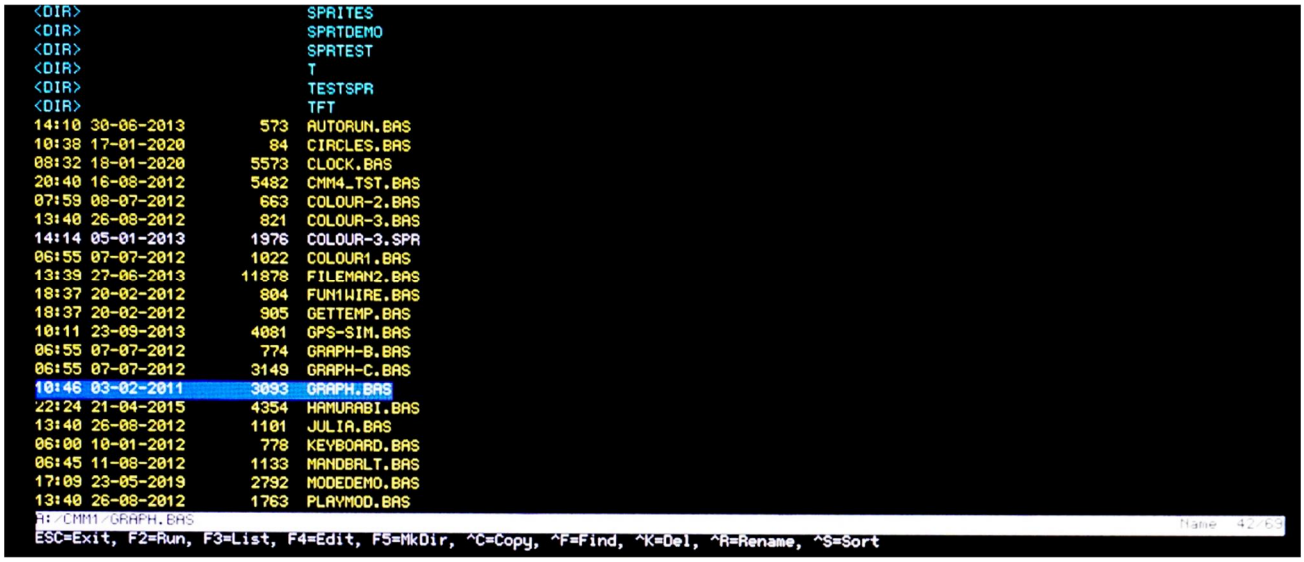
When using pin 40 for general I/O care must be taken so that it does not look like it is shorted to ground on power up. The symptom of this is unexplained sudden losses of all options that have been changed and the erasure of the program held in flash.

File Manager

The file manager is an easy way to manage files and directories on the SD card. Using this you can search, delete, rename, run, etc. The file manager will work with both the VGA video output and the serial console.

To run the file manager you use the command FILES at the command prompt or use the F1 key on your keyboard. This will start up in the current directory and list the files and directories there.

On the VGA monitor it will look like this:



To move around the list of files you use the arrow keys, Page Up or Page Down keys and the Home or End keys. Pressing Enter when positioned on a directory will take you into that directory and if it is positioned on a program it will run that program. If a mouse is fitted it can also be used to select and run files.

The Escape key (ESC) will exit the file manager.

At the bottom of the screen the status line lists details such as the current cursor position and the functions that are available. All these operate on the file currently selected by the cursor which is also displayed to the left of the status line:

Enter	This is the action key. If the file is a program this will RUN the program. If the file is an audio file this will PLAY the file on the sound output. If the file is a picture file the picture will be displayed – press any key to return to the file manager. If the cursor is positioned on a directory that directory will be entered. If the directory has the name ".." this will take you up one level in the directory hierarchy.
F3	This will LIST the program or text file selected.
F4	This will EDIT the program or text file selected.
F5	Will prompt for a directory name and create that directory.
CTRL-C	Will prompt for a file name and copy the selected file to that new name.
CTRL-F	Will enter the **search mode**. You will be prompted for the search text and as you type this in the cursor will automatically be positioned at the first matching file found. You can then use the down arrow key to search for the next occurrence or the up arrow key for the previous occurrence. The Enter key will leave the cursor where it is and return to normal mode. Escape will abort the search.
CTRL-K	Will delete a file or directory. A directory must be empty otherwise it will be ignored.
CTRL-R	Will rename a file or directory.
CTRL-S	This will toggle the sort order between name, size, type and date. The current sort order is displayed on the right hand side of the status line.

Mouse Support

To use the mouse in the file manager it needs to be configured using the OPTION MOUSE command (see the previous *Hardware Features* section. When configured the mouse works as you would expect, you can:

- Left click on a line to position the cursor.
- You can use the wheel to scroll up and down the file list.
- Right click in the top half of the screen to page up.
- Right click in the bottom half of the screen to page down.
- Double click to open/run a file (same as the Enter key).
- If you open an image you can return to the file manager by left clicking the mouse.

Full Screen Editor

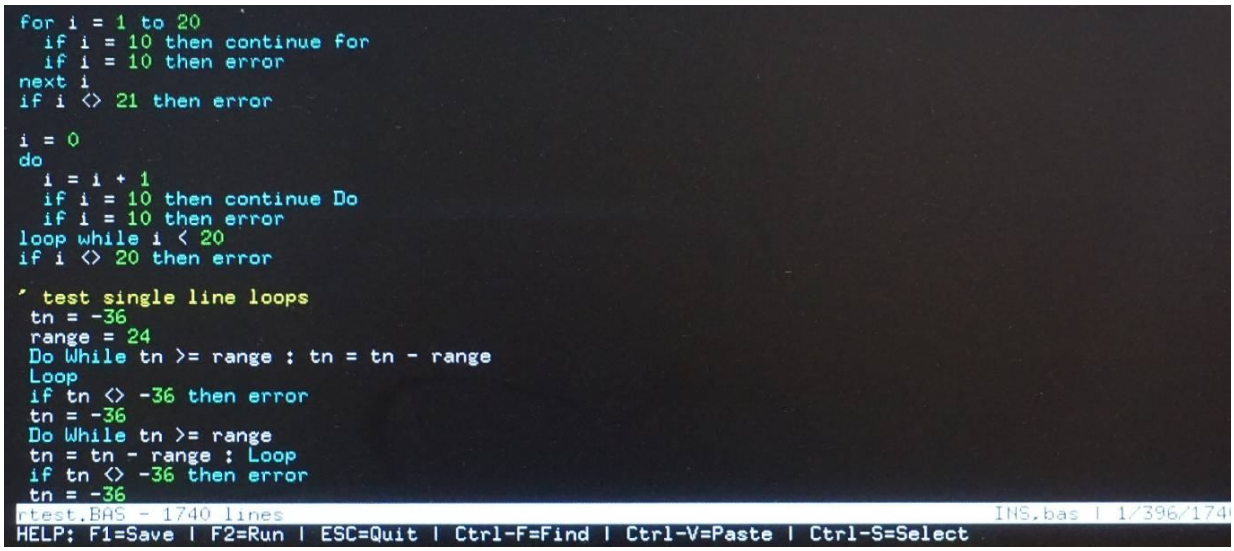
An important productivity feature is the built-in full screen editor. This will work with both the VGA video output and the serial console. To run the editor you use the command EDIT at the command prompt. For example:

```
EDIT "filename"
```

Note that double quote marks must be used around the file name. If the file's extension is not specified MMBasic will automatically add the extension ".BAS". If the file does not exist it will be created when you save and exit the editor. Non program files can be edited by specifying an extension other than ".BAS".

You can also use EDIT without a file name and in that case the last program that was edited or RUN will be edited. After editing the file it can be run using the RUN command without specifying a file name or you could use the F2 function key within the editor to save and run the program.

On the VGA screen the editor looks like this:



When the editor starts up the cursor will be automatically positioned at the last place that you were editing or, if your program had just been stopped by an error, the cursor will be positioned at the line that caused the error.

If you have used an editor like Windows Notepad previously you will find that the operation of this editor is familiar. The arrow keys will move the cursor around in the text, home and end will take you to the beginning or end of the line. Page up and page down will do what their titles suggest. The delete key will delete the character at the cursor and backspace will delete the character before the cursor. The insert key will toggle between insert and overtype modes. If a mouse is fitted it can also be used to position the cursor, select text, etc,

About the only unusual key combination is that two home key presses will take you to the start of the program and two end key presses will take you to the end. At the bottom of the screen the status line lists details such as the current cursor position and the common functions supported by the editor. The full selection of commands is:

ESC	This will cause the editor to abandon all changes and return to the command prompt with the file unchanged. If you have changed the text you will be asked to press ESC twice more to confirm this action.
F1	This will save the file and return to the command prompt.
F2	This will save the file and immediately run it.
F3 or CTRL-F	This will enter the **find mode**. You will be prompted for the search text and as you type this in the editor will automatically position the cursor at the first text found. You can then use the down arrow key to search for the next occurrence or the up arrow key for the previous occurrence. The Enter key will leave the cursor at this position and return to normal editing mode. F5 or CTRL-V will replace the searched text with whatever is in the clipboard (see below). Escape will abort the search.

F4 or CTRL-S	This will enter the **select mode**. In this mode you can use the arrow keys, HOME or END to select text and copy it to the clipboard. It will be highlighted on the screen as you select it. Then F5 or CTRL-C will copy the selection to the clipboard, F4 or CTRL-X will copy and delete the selection. DELETE will simply delete the selection and ESCAPE will return to the normal editing mode without changing anything. Note: you can also enter selection mode when using a USB keyboard by holding the shift key and pressing right-arrow or down-arrow.
F5 or CTRL-V	This will insert (at the current cursor position) the text that had been previously cut or copied in the select mode (see above).
F6	This will save the edited text and exit the editor similar to the F1 key. The difference is that F6 will not update the "current program name" which is used when the RUN, LIST and EDIT commands are entered without specifying a filename.
CTRL-K	Will delete all text from the current cursor position to the end of the line.
CTRL-W	Will allow you to save a backup copy of the edited file to a different file. The editor will continue to edit the original file.
F7	Will prompt for a file name and will insert the text from that file into the editor at the current cursor position.
F8 or CTRL-B	Will prompt for a file name and will write to the file the contents of the text that had been previously cut or copied in the select mode (see above). This together with F7 is an easy mechanism for moving blocks of text beween files.
F11	Will paste at the current cursor position the top command last viewed in the help dialogue. It is inactive until the help facility has been used.
F12	Enters the help dialogue and automatically sets any text under the cursor as the match string for help. Use F12 or ESC to exit the help dialogue. On exiting help the top line of the help dialogue will be available to paste into the current program using F11. See the HELP command for more details.
TAB	Will move the cursor to the next tab position as defined by OPTION TAB.
SHIFT TAB	Deletes a number of spaces (defined by OPTION TAB) if the cursor is on a space character. Useful for changing the indenting of a line. USB keyboard only.
SHIFT DELETE	if used at the beginning of a line deletes all leading spaces. Anywhere else in the line and it acts like DELETE. USB keyboard only.

For security all save commands will create a backup file by appending ".bak" to the filename and renaming the original file before saving the file. This ensures that in the event of any sort of error writing to SD card the worst case is that only the edited version is lost.

The best way to learn how to use the editor is to simply fire it up and experiment.

The editor is a very productive method of writing a program. With the command EDIT you can enter your program then, by pressing the F2 key, you can save and run the program. If your program stops with an error pressing the function key F4 at the command prompt will run the command EDIT and place you back in the editor with the cursor positioned at the line that caused the error. This edit/run/edit cycle is very fast.

Mouse Support

To use the mouse in the editor it needs to be configured using the OPTION MOUSE command (see the previous *Hardware Features* section. When configured the mouse works as you would expect, you can:

- Left click to position the edit cursor.
- Scroll wheel to move up and down.
- Left click and hold down then move cursor to a new position and release the left button. The enclosed area will be selected for cut and paste.
- Within cut and paste mode you can use the cursor and wheel to change the selection.
- To exit cut and paste without doing anything right click.
- Right click in top area of screen to scroll up (except in select mode).
- Right click in bottom of screen to scroll down (except in select mode).
- Left click at far left to scroll horizontally one character if the screen is scrolled horizontally.

- Left click at far right to scroll horizontally one character if the line is too long for the display.
- Double left click at the far left to scroll to the beginning of line if the screen is scrolled horizontally.
- Double left click at the far right to scroll horizontally to the end of line if the line is too long to display.

Colour Coded Editor Display

The editor will automatically colour code the edited program with keywords, numbers and comments displayed in different colours. If necessary this feature can be disabled with the command:

```
OPTION COLOURCODE OFF or OPTION COLOURCODE REVERSE
```

and re enabled with:

```
OPTION COLOURCODE ON
```

This setting is saved in non-volatile memory and automatically applied on startup. It applies to both the VGA output and the serial console.

Serial Console

The editor will also automatically work with the serial console (including colour coding).

To operate properly this feature requires a terminal emulator that can interpret the appropriate VT100 escape codes and respond correctly. It works correctly with Tera Term however Putty needs its default background colour to be changed to white (Settings >> Colours >> Default Background >> Modify).

Colour coding the editor's output requires many extra characters to be sent to the terminal emulator and this can slow down the screen update. If necessary colour coding can be turned off or the baud rate can be set to a higher speed for faster updates (see OPTION BAUDRATE).

Memory Usage

To provide long line editing and sideways scrolling the editor needs more RAM than would be implied by the number of characters being edited. This only has an effect on very large programs (ie, greater than approx 9,000 lines) on the original 1st Generation design and can cause the editor to generate an out of memory error when loading. This is generally not a problem with the 2nd Generation design.

The solution is to break the program into chunks with the relatively static portions included in the main program using the #INCLUDE command. These included files are inserted when the program is loaded via the RUN command and have no influence on performance or the overall amount of program memory used.

Variables and Expressions

In MMBasic command names, function names, labels, variable names, file names, etc are not case sensitive, so that "Run" and "RUN" are equivalent and "dOO" and "Doo" refer to the same variable.

Variables

Variables can start with an alphabetic character or underscore and can contain any alphabetic or numeric character, the period (.) and the underscore (_). They may be up to 32 characters long.

A variable name or a label must not be the same as a function or one of the following keywords: THEN, ELSE, GOTO, GOSUB, TO, STEP, FOR, WHILE, UNTIL, LOAD, MOD, NOT, AND, OR, XOR, AS.
Eg, step = 5 is illegal as STEP is a keyword.

MMBasic supports three different types of variables:

1. Double Precision Floating Point.
   These can store a number with a decimal point and fraction (eg, 45.386) however they will lose accuracy when more than 14 digits of precision are used. Floating point variables are specified by adding the suffix '!' to a variable's name (eg, i!, nbr!, etc). They are also the default when a variable is created without a suffix (eg, i, nbr, etc).
2. 64-bit Signed Integer.
   These can store positive or negative numbers with up to 19 decimal digits without losing accuracy but they cannot store fractions (ie, the part following the decimal point). These are specified by adding the suffix '%' to a variable's name. For example, i%, nbr%, etc.
3. A String.
   A string will store a sequence of characters (eg, "Tom"). Each character in the string is stored as an eight bit number and can therefore have a decimal value of 0 to 255. String variable names are terminated with a '$' symbol (eg, name$, s$, etc). Strings can be up to 255 characters long.

Note that it is illegal to use the same variable name with different types. Eg, using `nbr!` and `nbr%` in the same program would cause an error. This is different from the original Colour Maximite which allowed this.

Most programs use floating point variables for arithmetic as these can deal with the numbers used in typical situations and are more intuitive than integers when dealing with division and fractions. So, if you are not bothered with the details, always use floating point.

Constants

Numeric constants may begin with a numeric digit (0-9) for a decimal constant, &H for a hexadecimal constant, &O for an octal constant or &B for a binary constant. For example &B1000 is the same as the decimal constant 8. Constants that start with &H, &O or &B are always treated as 64-bit unsigned integer constants.

Decimal constants may be preceded with a minus (-) or plus (+) and may be terminated with 'E' followed by an exponent number to denote exponential notation. For example 1.6E+4 is the same as 16000.

When a constant number is used it will be assumed that it is an integer if a decimal point or exponent is not used. For example, 1234 will be interpreted as an integer while 1234.0 will be interpreted as a floating point number.

String constants are surrounded by double quote marks ("). Eg, "Hello World".

OPTION DEFAULT

A variable can be used without a suffix (ie, !, % or $) and in that case MMBasic will use the default type of floating point. For example, the following will create a floating point variable:

```
Nbr = 1234
```

However the default can be changed with the OPTION DEFAULT command. For example, OPTION DEFAULT INTEGER will specify that all variables without a specific type will be integer. So, the following will create an integer variable:

```
OPTION DEFAULT INTEGER
Nbr = 1234
```

The default can be set to FLOAT (which is the default when a program is run), INTEGER, STRING or NONE. In the latter all variables must be specifically typed otherwise an error will occur.

The OPTION DEFAULT command can be placed anywhere in the program and changed at any time but good practice dictates that if it is used it should be placed at the start of the program and left unchanged.

OPTION EXPLICIT

By default MMBasic will automatically create a variable when it is first referenced. So, `Nbr = 1234` will create the variable and set it to the number 1234 at the same time. This is convenient for short and quick programs but it can lead to subtle and difficult to find bugs in large programs. For example, in the third line of this fragment the variable `Nbr` has been misspelt as `Nbrs`. As a consequence the variable `Nbrs` would be created with a value of zero and the value of `Total` would be wrong.

```
Nbr = 1234
Incr = 2
Total = Nbrs + Incr
```

The OPTION EXPLICIT command tells MMBasic to not automatically create variables. Instead they must be explicitly defined using the DIM, LOCAL or STATIC commands (see below) before they are used. The use of this command is recommended to support good programming practice. If it is used it should be placed at the start of the program before any variables are used.

DIM and LOCAL

The DIM and LOCAL commands can be used to define a variable and set its type and are mandatory when the OPTION EXPLICIT command is used.

The DIM command will create a global variable that can be seen and used throughout the program including inside subroutines and functions. However, if you require the definition to be visible only within a subroutine or function, you should use the LOCAL command at the start of the subroutine or function. LOCAL has exactly the same syntax as DIM.

If LOCAL is used to specify a variable with the same name as a global variable then the global variable will be hidden to the subroutine or function and any references to the variable will only refer to the variable defined by the LOCAL command. Any variable created by LOCAL will vanish when the program leaves the subroutine.

At its simplest level DIM and LOCAL can be used to define one or more variables based on their type suffix or the OPTION DEFAULT in force at the time. For example:

```
DIM nbr%, s$
```

But it can also be used to define one or more variables with a specific type when the type suffix is not used:

```
DIM INTEGER nbr, nbr2, nbr3, etc
```

In this case nbr, nbr2, nbr3, etc are all created as integers. When you use the variable within a program you do not need to specify the type suffix. For example, `MyStr` in the following works perfectly as a string variable:

```
DIM STRING MyStr
MyStr = "Hello"
```

The DIM and LOCAL commands will also accept the Microsoft practice of specifying the variable's type after the variable with the keyword "AS". For example:

```
DIM nbr AS INTEGER, s AS STRING
```

In this case the type of each variable is set individually (not as a group as when the type is placed before the list of variables).

The variables can also be initialised while being defined. For example:

```
DIM INTEGER a = 5, b = 4, c = 3
DIM s$ = "World", i% = &H8FF8F
DIM msg AS STRING = "Hello" + " " + s$
```

The value used to initialise the variable can be an expression including user defined functions.

The DIM or LOCAL commands are also used to define an array and all the rules listed above apply when defining an array. For example, you can use:

```
DIM INTEGER nbr(10), nbr2, nbr3(5,8)
```

When initialising an array the values are listed as comma separated values with the whole list surrounded by brackets. For example:

```
DIM INTEGER nbr(5) = (11, 12, 13, 14, 15, 16)
```

or

```
DIM days(7) AS STRING = ("","Sun","Mon","Tue","Wed","Thu","Fri","Sat")
```

STATIC

Inside a subroutine or function it is sometimes useful to create a variable which is only visible within the subroutine or function (like a LOCAL variable) but retains its value between calls to the subroutine or function.

You can do this by using the STATIC command. STATIC can only be used inside a subroutine or function and uses the same syntax as LOCAL and DIM. The difference is that its value will be retained between calls to the subroutine or function (ie, it will not be initialised on the second and subsequent calls).

For example, if you had the following subroutine and repeatedly called it, the first call would print 5, the second 6, the third 7 and so on.

```
SUB Foo
  STATIC var = 5
  PRINT var
  var = var + 1
END SUB
```

Note that the initialisation of the static variable to 5 (as in the above example) will only take effect on the first call to the subroutine. On subsequent calls the initialisation will be ignored as the variable had already been created on the first call.

As with DIM and LOCAL the variables created with STATIC can be float, integers or strings and arrays of these with or without initialisation. The length of the variable name created by STATIC and the length of the subroutine or function name added together cannot exceed 32 characters.

CONST

Often it is useful to define an identifier that represents a value without the risk of the value being accidently changed - which can happen if variables were used for this purpose (this practice encourages another class of difficult to find bugs).

Using the CONST command you can create an identifier that acts like a variable but is set to a value that cannot be changed. For example:

```
CONST InputVoltagePin = 26
CONST MaxValue = 2.4
```

The identifiers can then be used in a program where they make more sense to the casual reader than simple numbers. For example:

```
IF PIN(InputVoltagePin) > MaxValue THEN SoundAlarm
```

A number of constants can be created on the one line:

```
CONST InputVoltagePin = 26, MaxValue = 2.4, MinValue = 1.5
```

The value used to initialise the constant is evaluated when the constant is created and can be an expression including user defined functions.

The type of the constant is derived from the value assigned to it; so for example, `MaxValue` above will be a floating point constant because 2.4 is a floating point number. The type of a constant can also be explicitly set by using a type suffix (ie, !, % or $) but it must agree with its assigned value.

Expressions and Operators

MMBasic will evaluate a mathematical expression using the standard mathematical rules. For example, multiplication and division are performed first followed by addition and subtraction. These are called the rules of precedence and are detailed below.

This means that 2 + 3 * 6 will resolve to 20, so will 5 * 4 and also 10 + 4 * 3 – 2.

If you want to force the interpreter to evaluate parts of the expression first you can surround that part of the expression with brackets. For example, (10 + 4) * (3 – 2) will resolve to 14 not 20 as would have been the case if the brackets were not used. Using brackets does not appreciably slow down the program so you should use them liberally if there is a chance that MMBasic will misinterpret your intension.

The following operators, in order of precedence, are implemented in MMBasic. Operators that are on the same level (for example + and -) are processed with a left to right precedence as they occur on the program line.

Arithmetic operators:

^	Exponentiation (eg, `b^n` means b^n)
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction

Shift operators:

x << y x >> y	These operate in a special way. << means that the value returned will be the value of x shifted by y bits to the left while >> means the same only right shifted. They are integer functions and any bits shifted off are discarded. For a right shift any bits introduced are set to the value of the top bit (bit 63). For a left shift any bits introduced are set to zero.

Logical operators:

NOT INV	invert the logical value on the right (eg, `NOT a=b` is `a<>b`) or bitwise inversion of the value on the right (eg, `a = INV b`)
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	equality
AND OR XOR	Conjunction, disjunction, exclusive or

For Microsoft compatibility the operators AND, OR and XOR are integer bitwise operators. For example, PRINT (3 AND 6) will output the number 2. Because these operators can act as both logical operators (for example, IF a=5 AND b=8 THEN …) and as a bitwise operators (eg, y% = x% AND &B1010) the interpreter will be confused if they are mixed in the same expression. So, always evaluate logical and bitwise expressions in separate expressions.

The other logical operations result in the integer 0 (zero) for false and 1 for true. For example the statement PRINT 4 >= 5 will print the number zero on the output and the expression A = 3 > 2 will store +1 in A.

The NOT operator will invert the logical value on its right (it is not a bitwise invert) while the INV operator will perform a bitwise invert. Both of these have the highest precedence so they will bind tightly to the next value. For normal use of NOT or INV the expression to be operated on should be placed in brackets. Eg:

```
IF NOT (A = 3 OR A = 8) THEN …
```

String operators:

+	Join two strings
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	Equality

String comparisons respect case. For example "A" is greater than "a".

Mixing Floating Point and Integers

MMBasic automatically handles conversion of numbers between floating point and integers. If an operation mixes both floating point and integers (eg, `PRINT A% + B!`) the integer will be converted to a floating point number first, then the operation performed and a floating point number returned. If both sides of the operator are integers then an integer operation will be performed and an integer returned.

The one exception is the normal division ("/") which will always convert both sides of the expression to a floating point number and then returns a floating point number. For integer division you should use the integer division operator "\".

MMBasic functions will return a float or integer depending on their characteristics. For example, PIN() will return an integer when the pin is configured as a digital input but a float when configured as an analog input.

If necessary you can convert a float to an integer with the INT() function. It is not necessary to specifically convert an integer to a float but if it was needed the integer value could be assigned to a floating point variable and it will be automatically converted in the assignment.

64-bit Unsigned Integers

MMBasic on the Colour Maximite 2 supports 64-bit signed integers. This means that there are 63 bits for holding the number and one bit (the most significant bit) which is used to indicate the sign (positive or negative). However it is possible to use full 64-bit unsigned numbers as long as you do not do any arithmetic on the numbers.

64-bit unsigned numbers can be created using the &H, &O or &B prefixes to a number and these numbers can be stored in an integer variable. You then have a limited range of operations that you can perform on these. They are << (shift left), >> (shift right), AND (bitwise and), OR (bitwise or), XOR (bitwise exclusive or), INV (bitwise inversion), = (equal to) and <> (not equal to). Arithmetic operators such as division or addition may be confused by a 64-bit unsigned number and could return nonsense results.

Note that shift right is a signed operation. This means that if the top bit is a one (a negative signed number) and you shift right then it will shift in ones to maintain the sign.

To display 64-bit unsigned numbers you should use the HEX$(), OCT$() or BIN$() functions.

For example, the following 64-bit unsigned operation will return the expected results:

```
X% = &HFFFF0000FFFF0044
Y% = &H800FFFFFFFFFFFFF
X% = X% AND Y%
PRINT HEX$(X%, 16)
```

Will display "800F0000FFFF0044"

Subroutines and Functions

A program defined subroutine or function is simply a block of programming code that is contained within a module and can be called from anywhere within your program. It is the same as if you have added your own command or function to the language.

Subroutines

A subroutine acts like a command and it can have arguments (sometimes called a parameter list). In the definition of the subroutine they look like this:

```
SUB MYSUB arg1, arg2$, arg3
  <statements>
  <statements>
END SUB
```

And when you call the subroutine you can assign values to the arguments. For example:

```
MYSUB  23, "Cat", 55
```

Inside the subroutine `arg1` will have the value `23`, `arg2$` the value of `"Cat"`, and so on. The arguments act like ordinary variables but they exist only within the subroutine and will vanish when the subroutine ends. You can have variables with the same name in the main program and they will be hidden by the arguments defined for the subroutine.

When calling a subroutine you can supply less than the required number of values and in that case the missing values will be assumed to be either zero or an empty string. You can also leave out a value in the middle of the list and the same will happen. For example:

```
MYSUB  23, , 55
```

Will result in `arg2$` being set to the empty string `""`.

Rather than using the type suffix (eg, the $ in arg2$) you can use the suffix AS <type> in the definition of the subroutine argument and then the argument will be known as the specified type, even when the suffix is not used. For example:

```
SUB MYSUB arg1, arg2 AS STRING, arg3
  IF arg2 = "Cat" THEN …
END SUB
```

Local Variables

Inside a subroutine you can define a variable using LOCAL (which has the same syntax as DIM). This variable will only exist within the subroutine and will vanish when the subroutine exits. You can have a variable in your main program with the same name but it will be hidden and the local variable used while the subroutine is executed.

If you do not declare the variable as LOCAL within the subroutine and OPTION EXPLICIT is not in force it will be created as a global variable and be visible in your main program and subroutines, just like a normal variable declared outside a subroutine or function.

Functions

Functions are similar to subroutines with the main difference being that the function is used to return a value in an expression. The rules for the argument list in a function are similar to subroutines. The only difference is that brackets are required around the argument list when you are calling a function, even if there are no arguments (they are optional when calling a subroutine).

To return a value from the function you assign a value to the function's name within the function. If the function's name is terminated with a $, a % or a ! the function will return that type, otherwise it will return whatever the OPTION DEFAULT is set to. You can also specify the type of the function by adding AS <type> to the end of the function definition.

For example:

```
FUNCTION Fahrenheit(C) AS FLOAT
  Fahrenheit = C * 1.8 + 32
END FUNCTION
```

Passing Arguments by Reference

If you use an ordinary variable (ie, not an expression) as the value when calling a subroutine or a function, the argument within the subroutine/function will point back to the variable used in the call and any changes to the argument will also be made to the supplied variable. This is called passing arguments by reference.

For example, you might define a subroutine to swap two values, as follows:

```
SUB Swap a, b
  LOCAL t
  t = a
  a = b
  b = t
END SUB
```

In your calling program you would use variables for both arguments:

```
Swap nbr1, nbr2
```

And the result will be that the values of `nbr1` and `nbr2` will be swapped.

For this to work the type of the variable passed (eg, `nbr1`) and the defined argument (eg, `a`) must be the same (in the above example both default to float).

Unless you need to return a value via the argument you should not use an argument as a general purpose variable inside a subroutine or function. This is because another user of your routine may unwittingly use a variable in their call and that variable could be "magically" changed by your routine. It is much safer to assign the argument to a local variable and manipulate that instead.

Passing Arrays

Single elements of an array can be passed to a subroutine or function and they will be treated the same as a normal variable. For example, this is a valid way of calling the Swap subroutine (discussed above):

```
Swap dat(i), dat(i + 1)
```

This type of construct is often used in sorting arrays.

You can also pass one or more complete arrays to a subroutine or function by specifying the array with empty brackets instead of the normal dimensions. For example, `a()`. In the subroutine or function definition the associated parameter must also be specified with empty brackets. The type (ie, float, integer or string) of the argument supplied and the parameter in the definition must be the same.

In the subroutine or function the array will inherit the dimensions of the array passed and these must be respected when indexing into the array. If required the dimensions of the array could be passed as additional arguments to the subroutine or function so it could correctly manipulate the array. The array is passed by reference which means that any changes made to the array within the subroutine or function will also apply to the supplied array.

For example, when the following is run the words "Hello World" will be printed out:

```
DIM MyStr$(5, 5)
MyStr$(4, 4) = "Hello" : MyStr$(4, 5) = "World"
Concat MyStr$()
PRINT MyStr$(0, 0)

SUB Concat arg$()
  arg$(0,0) = arg$(4, 4) + " " + arg$(4, 5)
END SUB
```

Early Exit

There can be only one END SUB or END FUNCTION for each definition of a subroutine or function. To exit early from a subroutine (ie, before the END SUB command has been reached) you can use the EXIT SUB command. This has the same effect as if the program reached the END SUB statement. Similarly you can use EXIT FUNCTION to exit early from a function.

Recursion

Recursion is where a subroutine or function calls itself. You can do recursion in MMBasic but there are a number of issues (these are a direct consequence of the limitations of microcontrollers and the BASIC language):

- There is a fixed limit to the depth of recursion. In the Colour Maximite 2 this is 50 levels.

- If you have many arguments to the subroutine or function and many LOCAL variables (especially strings) you could easily run out of memory before reaching the 50 level limit.
- Any FOR…NEXT loops and DO…LOOPs will be corrupted if the subroutine or function is recursively called from within these loops.

Examples

There is often the need for a special command or function to be implemented in MMBasic but in many cases these can be constructed using an ordinary subroutine or function which will then act exactly the same as a built in command or function.

For example, sometimes there is a requirement for a TRIM function which will trim specified characters from the start and end of a string. The following provides an example of how to construct such a simple function in MMBasic.

The first argument to the function is the string to be trimmed and the second is a string containing the characters to trim from the first string. RTrim$() will trim the specified characters from the end of the string, LTrim$() from the beginning and Trim$() from both ends.

```
' trim any characters in c$ from the start and end of s$
Function Trim$(s$, c$)
  Trim$ = RTrim$(LTrim$(s$, c$), c$)
End Function

' trim any characters in c$ from the end of s$
Function RTrim$(s$, c$)
  RTrim$ = s$
  Do While Instr(c$, Right$(RTrim$, 1))
    RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)
  Loop
End Function

' trim any characters in c$ from the start of s$
Function LTrim$(s$, c$)
  LTrim$ = s$
  Do While Instr(c$, Left$(LTrim$, 1))
    LTrim$ = Mid$(LTrim$, 2)
  Loop
End Function
```

As an example of using these functions:

```
S$ = "   ****23.56700  "
PRINT Trim$(s$, " ")
```

Will give "****23.56700"

```
PRINT Trim$(s$, " *0")
```

Will give "23.567"

```
PRINT LTrim$(s$, " *0")
```

Will give "23.56700"

Basic Graphics

There are ten basic drawing commands that you can use within MMBasic to draw images on the VGA monitor (none of these apply to the serial console).

There are also a number of advanced commands designed for programmers writing games (such as MODE, BLIT and SPRITE) however this section will focus on the standard commands used by most programmers.

Screen Coordinates

All screen coordinates and measurements on the screen are done in terms of pixels with the X coordinate being the horizontal position and Y the vertical position. The top left corner of the screen has the coordinates X = 0 and Y = 0 and the values increase as you move down and to the right of the screen.

By default on startup the VGA output will be set to 800x600 pixels with each pixel supporting 256 different colours. At this resolution the bottom right pixel will be at X = 799 and Y = 599.

Read Only Variables

There are six read only variables which provide useful information about the VGA video output:

- MM. HRES
  Returns the width of the display (the X axis) in pixels.
- MM. VRES
  Returns the height of the display (the Y axis) in pixels.
- MM.INFO(FONTHEIGHT)
  Returns the height of the current font (in pixels). All characters in a font have the same height.
- MM.INFO(FONTWIDTH)
  Returns the width of a character in the current font (in pixels). All characters in a font have the same width.
- MM.INFO(HPOS)
  Returns the X coordinate of the text cursor (ie, the horizontal location (in pixels) of where the next character will be printed on the VGA monitor)
- MM.INFO(VPOS)
  Returns the Y coordinate of the text cursor (ie, the vertical location (in pixels) of where the next character will be printed on the VGA monitor)

Colours

Colour is specified as a true colour 24 bit number where the top eight bits represent the intensity of the red colour, the middle eight bits the green intensity and the bottom eight bits the blue. For example the colour red is &HFF0000 and yellow is &HFFFF00.

An easier way to generate a colour value is to use the RGB() function which has the form:

```
RGB(red, green, blue)
```

A value of zero for a colour represents black and 255 represents full intensity.

The RGB() function also supports a shortcut where you can specify common colours by naming them. For example, RGB(red) or RGB(cyan). The colours that can be named using the shortcut form are black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey and grey (or USA spelling gray/lightgray).

In addition there is a special colour NOTBLACK. For any mode this will be the darkest colour that can be displayed that will not act as transparent when manipulated by graphics commands that support transparency.

Because the Colour Maximite 2 uses double precision floating point it can store the 24 bit number representing colour (i.e. returned by the RGB() function) in either a floating point variable or an integer variable.

MMBasic will automatically convert colours to the format required for the current colour depth set by the MODE command. So, for example, at startup the Colour Maximite 2's VGA output defaults to 8-bit colour and in this case the 16777215 colours that can be represented by a 24-bit colour specification will be translated as best as possible to the 256 colours supported by the 8-bit colour mode. Other VGA display modes can support up to 24-bit colour (in the Generation 2 version) loss of information.

The default colour for commands that require a colour parameter can be set with the COLOUR command. This is handy if your program uses a consistent colour scheme, you can then set the defaults and use the short version of the drawing commands throughout your program (the USA spelling COLOR is also accepted).

The COLOUR command takes the format:

```
COLOUR foreground-colour, background-colour
```

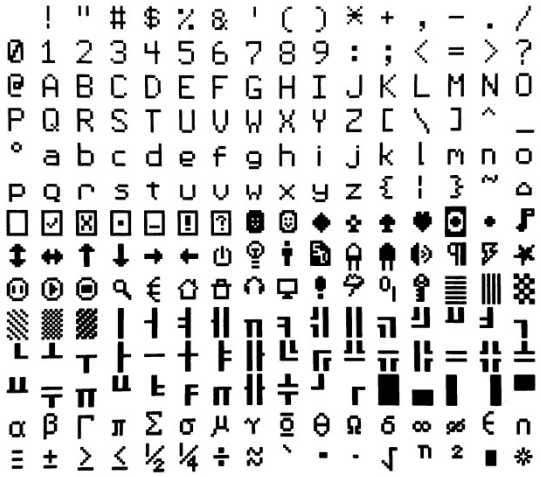
Fonts

There are seven built in fonts. These are:

Font Number	Size (width x height)	Character Set	Description
1	8 x 12	All 95 ASCII characters plus 7F to FF (hex)	Standard font (default on startup). Default font for the editor
2	12 x 20	All 95 ASCII characters	Medium sized font
3	16 x 24	All 95 ASCII characters	A larger font useful for the 800 x 600 display mode
4	10x16	All 95 ASCII characters plus 7F to FF (hex)	A useful font for improved clarity in high resolution modes
5	24 x 32	All 95 ASCII characters	Large font, very clear
6	32 x 50	0 to 9 plus some symbols	Numbers plus decimal point, positive, negative, equals, degree and colon symbols. Very clear.
7	6 x 8	All 95 ASCII characters	A small font useful when low resolutions are used.

In all fonts (including font #6) the back quote character (60 hex or 96 decimal) has been replaced with the degree symbol (º).

Font #1 (the default font) and font #4 have an extended character set covering all characters from CHR$(32) to CHR$(255) or 20 to FF (hex) as illustrated on the right.



If required, additional fonts can be embedded in a BASIC program. These fonts work exactly same as the built in font (ie, selected using the FONT command or specified in the TEXT command).

The format of an embedded font is:

```
DefineFont #Nbr
    hex [[ hex[…]
    hex [[ hex[…]
END DefineFont
```

It must start with the keyword "DefineFont" followed by the font number (which may be preceded by an optional # character). Any font number in the range of 2 to 5 and 8 to 16 can be specified and if it is the same as a built in font it will replace that font. The body of the font is a sequence of 8-digit hex words with each word separated by one or more spaces or a new line. The font definition is terminated by an "End DefineFont " keyword. These can be placed anywhere in a program and MMBasic will skip over it. This format is the same as that used by the Micromite.

Additional fonts and information can be found in the Embedded Fonts folder in the Colour Maximite 2 firmware download. These fonts cover a wide range of character sets including a symbol font (Dingbats) which is handy for creating on screen icons, etc.

In addition to using embedded fonts a program can dynamically load one font from the SD card using the LOAD FONT command. A program can load many fonts using this method during the course of its execution but each new font will overwrite the previously loaded font.

The format of fonts loaded using LOAD FONT have a similar format as the embedded fonts described above except that no comments or blank lines are allowed, the font number must always be #8, the first word must be on a line on its own and the following lines (except the last) must have exactly eight words per line.

As an example, the following is a tiny (6x4 pixel) font that is useful in the 320x200 display mode. This can be either loaded using LOAD FONT or embedded in the BASIC program:

```
DefineFont #8
 60200604
 44000000 00A04040 A0AEAE00 82406C6C EACC2048 00004460 84204424 E4A48044
 00E404A0 00800400 040000E0 00480240 4CE0AAEA 48C24044 C062C2E0 E820E2AA
 EA68E0E2 8048E2E0 EAE0EAEA 0404C0E2 80040400 0E208424 2484000E 4040E280
 4A60E84A CACAA0EA 608868C0 E8C0AACA E8E8E0E8 60EA6880 E4A0EAAA 2A22E044
 A0CAAA40 AEE08888 EEAEA0EA 40AA4AA0 4A80C8CA ECCA60AE C04268A0 AA4044E4
 A4AA60AA A0EEAA40 AAA04AAA 48E24044 E088E8E0 E2004208 004AE022 F0000000
 0C000084 AA8CE06A 608806C0 0660AA26 E42460AC 24AE0640 40A0CA88 22204044
 A0CC8AA4 0EE044C4 AA0CA0EE 40AA04A0 06C8AA0C 880662AA C0C60680 0A60444E
 AE0A60AA E0AE0A40 0AA0440A 6C0E24A6 608464E0 C4400444 006CC024 E0EEEE00
End DefineFont
```

You can convert the original Colour Maximite's font files to this format using the program FontTweak from: https://www.c-com.com.au/MMedit.htm

Drawing Commands

The drawing commands have optional parameters. You can completely leave these off the end of a command or you can use two commas in sequence to indicate a missing parameter. For example, the fifth parameter of the LINE command is optional so you can use this format:

```
LINE 0, 0, 100, 100, , rgb(red)
```

Optional parameters are indicated below by italics, for example: *font*.

In the following commands C is the drawing colour and defaults to the current foreground colour. FILL is the fill colour which defaults to -1 which indicates that no fill is to be used.

The drawing commands are:

- CLS *C*
  Clears the screen to the colour C. If C is not specified the current default background colour will be used.

- PIXEL X, Y, *C*
  Illuminates a pixel. If C is not specified the current default foreground colour will be used.

- LINE X1, Y1, X2, Y2, *LW, C*
  Draws a line starting at X1 and Y1 and ending at X2 and Y2.
  LW is the line's width and is only valid for horizontal or vertical lines. It defaults to 1 if not specified or is changed to 1 if the line is a diagonal.

- BOX X, Y1, W, H, *LW, C, FILL*
  Draws a box starting at X and Y1 which is W pixels wide and H pixels high.
  LW is the width of the sides of the box and can be zero. It defaults to 1.

- RBOX X, Y1, W, H, *R, C, FILL*
  Draws a box with rounded corners starting at X and Y1 which is W pixels wide and H pixels high.
  R is the radius of the corners of the box. It defaults to 10.

- TRIANGLE X1, Y1, X2, Y2, X3, Y3, *C, FILL*
  Draws a triangle with the corners at X1, Y1 and X2, Y2 and X3, Y3. C is the colour of the triangle and FILL is the fill colour. FILL can omitted or be -1 for no fill.

- CIRCLE X, Y, R, *LW, A, C, FILL*
  Draws a circle with X and Y as the centre and a radius R. LW is the width of the line used for the circumference and can be zero (defaults to 1). A is the aspect ratio which is a floating point number and defaults to 1. For example, an aspect of 0.5 will draw an oval where the width is half the height.

- ARC x, y, r1, r2, a1, a2, c
  Draws an arc with the centre at x and y, r1 and r2 are the inner and outer radius defining the thickness of the arc (if they are the same the arc will be one pixel thick), a1 and a2 are the start and end angles in degrees and c is the colour.

- POLYGON n, xarray%(), yarray%(), *C , FILL*
  Draws a outline or filled polygon defined by the x, y coordinate pairs in xarray%() and yarray%(). 'n' is the

number of points to use in drawing the polygon. If the last xy-coordinate pair is not the same as the first the firmware will automatically create an additional xy-coordinate pair to complete the polygon.

- `TEXT X, Y, STRING, ALIGNMENT, FONT, SCALE, C, BC`
  Displays a string starting at X and Y. ALIGNMENT is 0, 1 or 2 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 or RIGHT aligned text and the second letter is the vertical alignment around Y and can be T, M or B for TOP, MIDDLE or BOTTOM aligned text. The default alignment is left/top. FONT and SCALE are optional and default to that set by the FONT command. C is the drawing colour and BC is the background colour. They are optional and default to that set by the COLOUR command.

Most graphics commands allow the use of arrays as parameters so that you can draw multiple graphic objects with the one command. In this case the array is passed as the array name followed by empty brackets (eg `arr()`). Drawing multiple graphic elements this way is *much* faster than drawing them one by one using separate commands.

For example, the PIXEL command allows arrays to be specified for the x and y coordinates (in this case both must be arrays). The firmware will then plot the number of pixels as determined by the dimensions of the smallest array. For the PIXEL command 'c' can also be an array or a single variable/constant.

This is demonstrated with the following example which will draw three pixels in different colours:

```
DIM xx(2) = (10, 20, 30)
DIM yy(2) = (100, 150, 200)
DIM cc(2) = (RGB(red), RGB(green), RGB(blue))
PIXEL xx(), yy(), cc()
```

Example of Basic Graphics

As an example, the following program will draw a simple digital clock on the VGA monitor.

```
CLS
CONST DBlue = RGB(0, 0, 128)                ' A dark blue colour
COLOUR RGB(GREEN), RGB(BLACK)               ' Set the default colours
FONT 6                                      ' Set the default font

BOX 0, 0, MM.HRes-1, MM.VRes/2, 3, RGB(RED), DBlue

DO
  TEXT MM.HRes/2, MM.VRes/4, TIME$, "CM", 6, 1, RGB(CYAN), DBlue
  TEXT MM.HRes/2, MM.VRes*3/4, DATE$, "CM"
LOOP
```

The program starts by defining a constant with a value corresponding to a dark blue colour and then sets the defaults for the colours and the font. It then draws a box with red walls and a dark blue interior. Following this the program enters a continuous loop where it performs two functions:

1. Displays the current time inside the previously drawn box. The string is drawn centred both horizontally and vertically in the middle of the box. Note that the TEXT command overrides both the default font and colours to set its own parameters.

2. Draws the date centred in the lower half of the screen. In this case the TEXT command uses the default font and colours previously set.

The screenshot on the right shows the result.

Rotated Text

As described above the alignment of the text in the TEXT command can be specified by using one or two characters. In addition a third character can be used to indicate the rotation of the text. This character can be one of:

- 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 (ie, upside down)
- U the text will be rotated counter clockwise by 90º
- D the text will be rotated clockwise by 90º

As an example, the following will display the words "Vertical Text" vertically down the left hand margin of the monitor and centred vertically:

```
TEXT 0, 250, "Vertical Text", "LMV", 5
```

Positioning is relative to the top left corner of the character when viewed normally so inverted 100,100 will have the top left pixel of the first character at 100,100 and the text will then be above y=101 and to the left of x=101. Similarly “R” in the alignment string is viewed from the perspective of the character in whatever orientation it is in.

Transparent Text

The TEXT command 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.

Displaying Images

Using the LOAD command you can load an image from the SD card and display it on the VGA monitor. Supported formats are BMP, GIF, JPG and PNG and the image can be positioned anywhere on the screen.

There are some limitations on the format of the images and these are detailed in the commands later in this manual. The most flexible is the LOAD BMP command which supports all types of the BMP format including black and white and true colour 24-bit images. The image can be positioned anywhere on the screen and be of any size (pixels that end up being positioned off the screen and will be ignored).

Advanced Graphics Tutorial

The graphics subsystem in the Colour Maximite 2 has many advanced features such as multiple video pages, layered images, sprites and sophisticated methods of manipulating images. Peter Mather, who developed the graphics subsystem, has written a tutorial for games developers and advanced users.

This is presented as a series of posts on the Back Shed Forum and is recommended reading for users who need to get into the details: https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=12125 . A PDF version of this is included in the Colour Maximite 2 firmware zip file.

SD Card Support

The Colour Maximite 2 has full support for programs, files and directories on the SD card. This includes opening files for reading, writing or random access and editing and running programs.

MMBasic will work with cards up to 128GB in capacity. Cards larger than 32GB should be formatted as exFAT and cards 32GB or less formatted as FAT32. The recommended size is 8GB formatted as FAT32.

In the following note that:

- The filename can be a string expression, variable or constant. If it is a constant the string must be quoted (eg, KILL "MYPROG.BAS"). The exception is the RUN command where only a constant is allowed.
- Long file/directory names are supported in addition to the old 8.3 format.
- The maximum file/path length is 127 characters.
- Upper/lowercase characters and spaces are allowed although the file system is not case sensitive.
- Directory paths are allowed in file/directory strings. (ie, OPEN "/dir1/dir2/file.txt" FOR …).
- Forward slashes or back slashes are valid in paths between directories. Eg /dir/file.txt or \dir\file.txt.
- The current MMBasic time is used for file create and last access times.
- Up to ten files can be simultaneously open.
- Input and output commands and functions can also use file #0 which refers to the console.

There are 34 commands and functions related to the SD card. See the Commands/Functions section later in this manual for their full description):

Program Management Commands

All programs reside on the SD card and so it must be present when running, editing and listing programs. This is different from the original Maximite where the SD card was not necessarily required.

□ RUN “prog”
Run a program. ‘prog’ must be a string constant (ie, not a variable).

□ EDIT fname$
Edit a program or text file.

□ LIST fname$
List on the console a program or text file.

□ AUTOSAVE fname$
Receive a file streamed by a computer connected to the serial console.

□ XMODEM RECEIVE fname$
Receive a file from a computer connected to the serial console using the XModem protocol.

□ XMODEM SEND fname$
Send a file to a computer connected to the serial console using the XModem protocol.

File Access Within a Program

Except for INPUT, LINE INPUT and PRINT the # in #fnbr is optional and may be omitted.

□ OPEN fname$ FOR mode AS #fnbr
Opens a file for reading or writing. 'fname$' is the file name, 'mode' can be INPUT, OUTPUT, APPEND or RANDOM, ‘fnbr’ is the file number (1 to 10).

□ PRINT #fnbr, expression [[,;]expression] … etc
Outputs text to the file opened as #fnbr.

□ INPUT #fnbr, list of variables
Read a list of comma separated data into the variables specified from the file previously opened as #fnbr.

□ SEEK #fnbr, pos
Will position the read/write pointer in a file that has been opened for RANDOM access to the 'pos' byte.

- LINE INPUT #fnbr, variable$
  Read a complete line into the string variable specified from the file previously opened as #fnbr.
- CLOSE #fnbr [,#fnbr] …
  Close the file(s) previously opened with the file number '#fnbr'.

Also there are a number of functions that support the above commands.

- INPUT$(nbr, #fnbr)
  Will return a string composed of a number of characters read from a file previously opened for INPUT.
- EOF(#fnbr)
  Will return true if the file previously opened for INPUT with the file number '#fnbr' is positioned at the end of the file.
- LOC(#fnbr)
  For a file opened as RANDOM this will return the current position of the read/write pointer in the file.
- LOF(#fnbr)
  Will return the current length of the file in bytes

File and Directory Management

- LIST FILES [wildcard] [,sortorder]
  Search the current directory and list the files/directories found.
- KILL fname$
  Delete a file.
- COPY oldfile$ TO newfile$
  Copy a file.
- RENAME oldfile$ AS newfile$
  Rename a file.
- MKDIR dname$
  Make a sub directory.
- CHDIR dname$
  Change into to the directory $dname. $dname can also be ".." (dot dot) for up one directory or "/" for the root directory.
- RMDIR dir$
  Remove, or delete, the directory 'dir$'.

And there are two functions that are handy for searching and managing files/directories.

- DIR$(fspec, type)
  Will search an SD card for files and return the names of the entries found.
- MM.INFO(function)
  Returns many types of information related to the SD card (size, free space, file size, etc).

Play Audio Files

- PLAY WAV | FLAC | MP3 file$ [, interrupt]
  Play a WAV, FLAC or MP3 audio file on the stereo audio output.
- PLAY MODFILE file$
  Play a MOD file on the stereo audio output.
- PLAY EFFECT filename$ [,interrupt]
  Play a WAV file at the same time as a MOD file is playing.

Load and Save Images

- LOAD BMP | GIF | JPG | PNG fname$
  Load a BMP, GIF, JPG or PNG image and display it on the VGA monitor.

- SAVE IMAGE fname$
  Save the current VGA monitor's screen image as a BMP file.

Sequential File Access

Sequential input/output is the standard method of reading or writing to a file and the easiest to understand. When a file is opened it is read from the beginning character by character or line by line. Similarly, when a file is opened for writing the output is sequentially added to the end of the file. This method is often used for recording data or saving temporary information.

In the example below a file is created and two lines are written to the file (using the PRINT command). The file is then closed.

```
OPEN "fox.txt" FOR OUTPUT AS #1
PRINT #1, "The quick brown fox"
PRINT #1, "jumps over the lazy dog"
CLOSE #1
```

You can read the contents of the file using the LINE INPUT command. For example:

```
OPEN "fox.txt" FOR INPUT AS #1
LINE INPUT #1,a$
LINE INPUT #1,b$
CLOSE #1
```

LINE INPUT reads one line at a time so the variable `a$` will contain the text "The quick brown fox" and `b$` will contain "jumps over the lazy dog".

Another way of reading from a file is to use the INPUT$() function. This will read a specified number of characters. For example:

```
OPEN "fox.txt" FOR INPUT AS #1
ta$ = INPUT$(12, #1)
tb$ = INPUT$(3, #1)
CLOSE #1
```

The first INPUT$() will read 12 characters and the second 3 characters. So the variable `ta$` will contain "The quick br" and the variable `tb$` will contain "own".

Files normally contain just text and the print command will convert numbers to text. So in the following example the first line will contain the line "123" and the second "56789".

```
nbr1 = 123 : nbr2 = 56789
OPEN "numbers.txt" FOR OUTPUT AS #1
PRINT #1, nbr1
PRINT #1, nbr2
CLOSE #1
```

Again you can read the contents of the file using the LINE INPUT command but then you would need to convert the text to a number using VAL(). For example:

```
OPEN "numbers.txt" FOR INPUT AS #1
LINE INPUT #1, a$
LINE INPUT #1, b$
CLOSE #1
x = VAL(a$) : y = VAL(b$)
```

Following this the variable `x` would have the value 123 and `y` the value 56789.

Random File Access

Random access allows the program to jump around within a file so that sections in the middle (ie, not at the end) can be read or written. This method is often used for database type applications where the file consists of many records which have the same fixed length.

For random access the file should be opened with the keyword RANDOM. For example:

```
OPEN "filename" FOR RANDOM AS #1
```

To seek to a record within the file you would use the SEEK command which will position the read/write pointer to a specific byte. The first byte in a file is numbered one so, for example, the fifth record in a file that uses 64 byte records would start at byte 257. In that case you would use the following to point to it:

```
SEEK #1, 257
```

When reading from a random access file the INPUT$() function should be used as this will read a fixed number of bytes (ie, a complete record) from the file. For example, to read a record of 64 bytes you would use:

```
dat$ = INPUT$(64, #1)
```

When writing to the file a fixed record size should be used and this can be easily accomplished by adding sufficient padding characters (normally spaces) to the data to be written. For example:

```
PRINT #1, dat$ + SPACE$(64 – LEN(dat$);
```

The SPACE$() function is used to add enough spaces to ensure that the data written is an exact length (64 bytes in this example). The semicolon at the end of the print command suppresses the addition of the carriage return and line feed characters which would make the record longer than intended.

Two other functions can help when using random file access. The LOC() function will return the current byte position of the read/write pointer and the LOF() function will return the total length of the file in bytes.

The following program demonstrates random file access. Using it you can append to the file (to add some data in the first place) then read/write records using random record numbers. The first record in the file is record number 1, the second is 2, etc.

```
RecLen = 64
OPEN "test.dat" FOR RANDOM AS #1
DO
   abort: PRINT
   PRINT "Number of records in the file =" LOF(#1)/RecLen
   INPUT "Command (r = read,w = write, a = append, q = quit): ", cmd$
   IF cmd$ = "q" THEN CLOSE #1 : END
   IF cmd$ = "a" THEN
      SEEK #1, LOF(#1) + 1
   ELSE
      INPUT "Record Number: ", nbr
      IF nbr < 1 or nbr > LOF(#1)/RecLen THEN PRINT "Invalid record" : GOTO abort
      SEEK #1, RecLen * (nbr - 1) + 1
   ENDIF
   IF cmd$ = "r" THEN
      PRINT "The record = " INPUT$(RecLen, #1)
   ELSE
      LINE INPUT "Enter the data to be written: ", dat$
      PRINT #1,dat$ + SPACE$(RecLen - LEN(dat$));
   ENDIF
LOOP
```

Random access can also be used on a normal text file. For example, this will print out a file backwards:

```
OPEN "file.txt" FOR RANDOM AS #1
FOR i = LOF(#1) TO 1 STEP -1
   SEEK #1, i
   PRINT INPUT$(1, #1);
NEXT i
CLOSE #1
```

Audio Output

The Colour Maximite 2 can play WAV, FLAC and MP3 files from the SD card, generate synthesised music in the MOD format, create robot speech and sound effects as well as generate precise sine wave tones. All these are outputted on the audio socket. The ARM Cortex-M7 chip includes its own DAC (digital to analog converter) so an output filter network is not needed (as on the original Colour Maximite).

Playing WAV, MP3 and FLAC Files

The PLAY command will play an audio file residing on an SD card to the sound output. It can be used to provide background music, add sound effects to programs and provide informative announcements.

The syntax of the command is one of the following depending of the format of the file:

```
   PLAY WAV file$, interrupt
or PLAY MP3 file$, interrupt
or PLAY FLAC file$, interrupt
```

file$ is the name of the audio file to play. It must be on the SD card and the appropriate extension (eg .WAV) will be appended if missing. The audio will play in the background (ie, the program will continue without pause). *interrupt* is optional and is the name of a subroutine which will be called when the file has finished playing. Most variations in encoding are supported (see the PLAY command in the command listing for the details).

Background Music

If *fname$* in the PLAY WAV/MP3/FLAC command is a directory then the firmware will list all the files of the relevant type in that directory and start playing them one-by-one. To play files in the current directory use an empty string (ie, ""). Each file listed will play in turn and the optional interrupt will fire when all files have been played. The filenames are stored with full path so you can use CHDIR while tracks are playing without causing problems. All files in the directory are listed if the command is executed at the command prompt but the listing is suppressed in a program.

While playing in this background mode the user can edit programs, run programs, etc without interrupting the playing of the music. Amongst other things this allows the Colour Maximite 2 to be used as a music player while programming or doing other tasks.

Generating Sine Waves

The PLAY TONE command also uses the audio output and will generate sine waves with selectable frequencies for the left and right channels. This feature is intended for generating attention catching sounds but, because the frequency is very accurate, it can be used for many other applications. For example, signalling DTMF tones down a telephone line or testing the frequency response of loudspeakers.

The syntax of the command is:

```
PLAY TONE left, right, duration, interrupt
```

left and *right* are the frequencies in Hz to use for the left and right channels. The tone plays in the background (the program will continue running after this command) and 'dur' specifies the number of milliseconds that the tone will sound for.

duration is optional and if not specified the tone will continue until explicitly stopped or the program terminates. *interrupt* (if specified) will be triggered when the duration has finished.

The frequency can be from 1 Hz to 20 KHz and is very accurate (it is based on a crystal oscillator). The frequency can be changed at any time by issuing a new PLAY TONE command. Note that the sine wave is generated by stepping through a lookup table so to reduce the distortion the audio output should be passed through a low pass filter.

Specialised Audio Output

There are a number of specialised audio commands that are mostly used in computer games.

These are:

- PLAY MODFILE which will play synthesised music using the MOD format.
- PLAY TTS command which will generate robotic speech.
- PLAY SOUND which will generate an output based on a mixture of sine, square, noise, etc waveforms.
- PLAY EFFECT command which will play a WAV at the same time as a MOD file is playing.

Using PLAY

It is important to realise that the PLAY command will generate the audio in the background. This allows a program (for example) to play the sound of an explosion while still animating the visual of the explosion on the screen. Without the background facility the whole computer would freeze while the sound was heard.

However, generating the audio in the background has some subtle inferences which can trip up newcomers. For example, take the following program:

```
PLAY TONE 500, 500, 2000
END
```

You may expect the 500Hz tone to sound for 2 seconds but in practice it will not make any sound at all. This is because MMBasic will execute the PLAY TONE command (which will start generating the sound in the background) and then it will immediately continue and execute the END command which will terminate the program and the background sound. This happens so fast that nothing is heard.

Similarly the following program will not work either:

```
PLAY TONE 500, 500, 2000
PLAY TONE 300, 300, 5000
```

This is because the first command will set a 500Hz the tone playing but then the second PLAY command will immediately replace that with a 300Hz tone and following that the program will run off the end terminating the program and the background audio resulting in nothing being heard.

If you want MMBasic to wait while the PLAY command is doing its thing you should use suitable PAUSE commands. For example:

```
PLAY TONE 500, 500
PAUSE 2000
PLAY TONE 300, 300
PAUSE 5000
```

This applies to all versions of the PLAY command (eg, PLAY WAV/MP3/FLAC, etc).

Utility Commands

There are a number of commands that can be used to manage the sound output:

`PLAY PAUSE`	Temporarily halt (pause) the currently playing file or tone.
`PLAY RESUME`	Resume playing a file or tone that was previously paused.
`PLAY STOP`	Terminate the playing of the file or tone. The sound output will also be automatically stopped when the program ends.
`PLAY VOLUME L, R`	Set the volume to between 0 and 100 with 100 being the maximum volume. The volume will reset to the maximum level when a program is run.

The following commands can be used when playing a sequence of files (ie, "background music"):

`PLAY NEXT`	Skip to the next file.
`PLAY PREVIOUS`	Skip to the previous file.

Using the I/O pins

The 40-pin ribbon connector on the rear panel provides 28 input/output pins which can be controlled from within the BASIC program with 12 of these supporting the measurement of voltage. An I/O pin is referred to by its pin number and these, and their capabilities, are listed towards the start of this manual in the section *External I/O Connector*.

Digital Inputs

A digital input is the simplest type of input configuration. If the input voltage is higher than 2.5V the logic level will be true (numeric value of 1) and anything below 0.65V will be false (numeric value of 0). The inputs use a Schmitt trigger input so anything in between these levels will retain the previous logic level. Pins marked as 5V are 5V tolerant and can be directly connected to a circuit that generates up to 5.5V without the need for voltage dropping resistors.

In your BASIC program you would set the input as a digital input and use the PIN() function to get its level. For example:

```
SETPIN 4, DIN
IF PIN(4) = 1 THEN PRINT "High"
```

The SETPIN command configures pin 4 as a digital input and the PIN() function will return the value of that pin (the number 1 if the pin is high). The IF command will then execute the command after the THEN statement if the input was high. If the input pin was low the program would just continue with the next line in the program.

The SETPIN command also recognises a couple of options that will connect an internal resistor from the input to either the supply or ground. This is called a "pullup" or "pulldown" resistor and is handy when connecting to a switch as it saves having to install an external resistor to place a voltage across the contacts.

Analog Inputs

Pins marked as ANALOG can be configured to measure the voltage on the pin. The input range is from zero to 3.3V and the PIN() function will return the voltage. For example:

```
> SETPIN 23, AIN
> PRINT PIN(23)
 2.345
>
```

You will need a voltage divider if you want to measure voltages greater than 3.3V. For small voltages you may need an amplifier to bring the input voltage into a reasonable range for measurement.

The measurement uses 3.3V power supply to the CPU as its reference and it is assumed that this is exactly 3.3V. This value can be changed with the OPTION command.

Counting Inputs

The pins marked as COUNT can be configured as counting inputs to measure frequency, period or just count pulses on the input.

For example, the following will print the frequency of the signal on pin 7:

```
> SETPIN 7, FIN
> PRINT PIN(7)
110374
>
```

In this case the frequency is 110.374 kHz.

By default the gate time is one second which is the length of time that MMBasic will use to count the number of cycles on the input and this means that the reading is updated once a second with a resolution of 1 Hz. By specifying a third argument to the SETPIN command it is possible to specify an alternative gate time between 10 ms and 100000 ms. Shorter times will result in the readings being updated more frequently but the value returned will have a lower resolution. The PIN() function will always return the frequency in Hz regardless of the gate time used.

For example, the following will set the gate time to 10ms with a corresponding loss of resolution:

```
> SETPIN 7, FIN, 10
> PRINT PIN(7)
110300
>
```

For accurate measurement of signals less than 10 Hz it is generally better to measure the period of the signal. When set to this mode the Colour Maximite 2 will measure the number of milliseconds between sequential rising edges of the input signal. The value is updated on the low to high transition so if your signal has a period of (say) 100 seconds you should be prepared to wait that amount of time before the PIN() function will return an updated value.

The COUNTING pins can also count the number of pulses on their input. When a pin is configured as a counter (for example, `SETPIN 7,CIN`) the counter will be reset to zero and Colour Maximite 2 will then count every transition from a low to high voltage. The counter can be reset to zero again by executing the SETPIN command a second time (even though the input was already configured as a counter).

There is also a fast counting input (pin 18) which has been tested up to 40MHz for frequency measurement and counting. Note that period measurement is not available on this pin

Digital Outputs

All I/O pins can be configured as a standard digital output. This means that when an output pin is set to logic low it will pull its output to zero and when set high it will pull its output to 3.3V. In MMBasic this is done with the PIN command. For example `PIN(15) = 0` will set pin 15 to low while `PIN(15) = 1` will set it high.

The "OC" option on the SETPIN command makes the output pin open collector. This means that the output driver will pull the output low (to zero volts) when the output is set to a logic low but will go to a high impedance state when set to logic high. If you then connect a pull-up resistor to 5V (on pins that are 5V tolerant) the logic high level will be 5V (instead of 3.3V using the standard output mode). The maximum pull-up voltage in this mode is 5.5V.

Pulse Width Modulation

The PWM (Pulse Width Modulation) command allows the Colour Maximite 2 to generate square waves with a program controlled duty cycle. By varying the duty cycle you can generate a program controlled voltage output for use in controlling external devices that require an analog input (power supplies, motor controllers, etc). The PWM outputs are also useful for driving servos and for generating a sound output via a small transducer.

There are two PWM controllers; the first has three outputs and the second two to give a total of five PWM outputs.

When the Colour Maximite 2 is powered up or the PWM OFF command is used the PWM outputs will be set to high impedance (they are neither off nor on). So, if you want the PWM output to be low by default (zero power in most applications) you should use a resistor to pull the output to ground when it is set to high impedance. Similarly, if you want the default to be high (full power) you should connect the resistor to 3.3V.

Interrupts

Interrupts are a handy way of dealing with an event that can occur at an unpredictable time. An example is when the user presses a button. In your program you could insert code after each statement to check to see if the button has been pressed but an interrupt makes for a cleaner and more readable program.

When an interrupt occurs MMBasic will execute a special section of code and when finished return to the main program. The main program is completely unaware of the interrupt and will carry on as normal.

Any I/O pin that can be used as a digital input can be configured to generate an interrupt using the SETPIN command with up to ten interrupts active at any one time. Interrupts can be set up to occur on a rising or falling digital input signal (or both) and will cause an immediate branch to the specified user defined subroutine. The target can be the same or different for each interrupt. Return from an interrupt is via the END SUB or EXIT SUB commands. Note that no parameters can be passed to the subroutine however within the interrupt subroutine calls to other subroutines are allowed.

If two or more interrupts occur at the same time they will be processed in order of the interrupts as defined with SETPIN. During the processing of an interrupt all other interrupts are disabled until the interrupt subroutine

returns. During an interrupt (and at all times) the value of the interrupt pin can be accessed using the PIN() function.

Interrupts can occur at any time but they are disabled during INPUT statements. Also interrupts are not recognised during some long hardware related operations (eg, the TEMPR() function) although they will be recognised if they are still present when the operation has finished. When using interrupts the main program is completely unaffected by the interrupt activity unless a variable used by the main program is changed during the interrupt.

Because interrupts run in the background they can cause difficult to diagnose bugs. Keep in mind the following factors when using interrupts:

- For most programs MMBasic will respond to an interrupt in under 30µs however some commands (such as the TEMPR() function) can block interrupts for up to 200ms and it is possible for an interrupt (eg, a button press) to occur and vanish within this window and in that case it will never be recognised.
- When inside an interrupt all other interrupts are blocked so your interrupts should be short and exit as soon as possible. For example, never use PAUSE inside an interrupt. If you have some lengthy processing to do you should simply set a flag and immediately exit the interrupt, then your main program loop can detect the flag and do whatever is required.
- The subroutine that the interrupt calls (and any other subroutines called by it) should always be exclusive to the interrupt. If you must call a subroutine that is also used by an interrupt you must disable the interrupt first (you can reinstate it after you have finished with the subroutine).
- Remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non-intuitive bugs.

In addition to interrupts generated by the change in state of an I/O pin, an interrupt can also be generated by other sections of MMBasic including timers and communications ports and the above notes also apply to them.

Special Device Support

To make it easier for a program to interact with the external world the Colour Maximite 2 includes drivers for a number of common peripheral devices.

Infrared Remote Control Decoder

You can easily add a remote control to the Colour Maximite 2. The solder pads for the IR receiver are on the motherboard (near the Wii connector) and it is only a matter of soldering in a suitable receiver. With this done you can use the IR command to handle the device and key codes within your program.

It will work with any NEC or Sony compatible remote controls including ones that generate extended messages. Most cheap programmable remote controls will generate either protocol and using one of these you can add a sophisticated flair to your programs. The NEC protocol is also used by many other manufacturers including Apple, Pioneer, Sanyo, Akai and Toshiba so their branded remotes can be used.

NEC remotes use a 38kHz modulation of the IR signal and suitable receivers tuned to this frequency include the Vishay TSOP4838, Jaycar ZD1952 and Altronics Z1611A .

Sony remotes use a 40 kHz modulation and receivers for this frequency can be hard to find. Generally 38 kHz receivers will work but maximum sensitivity will be achieved with a 40 kHz receiver.

To setup the decoder you use the command:

```
IR  dev, key, interrupt
```

Where dev is a variable that will be updated with the device code and key is the variable to be updated with the key code. Interrupt is the interrupt subroutine to call when a new key press has been detected. The IR decoding is done in the background and the program will continue after this command without interruption.

This is an example of using the IR decoder:

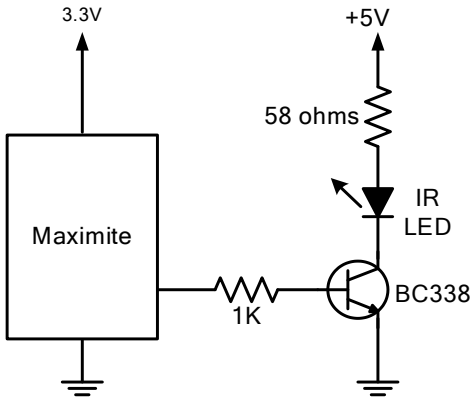
```
IR DevCode, KeyCode, IR_Int         ' start the IR decoder
DO
  ' < body of the program >
LOOP

SUB IR_Int                          ' a key press has been detected
  PRINT "Received  device = " DevCode "  key = " KeyCode
END SUB
```

IR remote controls can address many different devices (VCR, TV, etc) so the program would normally examine the device code first to determine if the signal was intended for the program and, if it was, then take action based on the key pressed. There are many different devices and key codes so the best method of determining what codes your remote generates is to use the above program to discover the codes.

Infrared Remote Control Transmitter

Using the IR SEND command you can transmit a 12 bit Sony infrared remote control signal. This is intended for communications with another Colour Maximite 2 or a Micromite but it will also work with Sony equipment that uses 12 bit codes. Note that all Sony products require that the message be sent three times with a 26mS delay between each message.

The circuit on the right illustrates what is required. The transistor is used to drive the infrared LED because the output of the I/O pin is limited to about 10mA. This circuit provides about 50mA to the LED.



To send a signal you use the command:

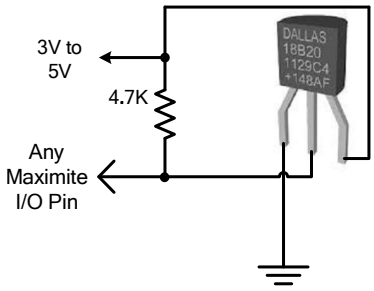
```
IR SEND pin, dev, cmd
```

Where pin is the I/O pin used, dev is the device code to send and key is the key code. Any I/O pin can be used and you do not have to set it up beforehand (the IR SEND command will automatically do that).

Note that the modulation frequency used is 38KHz and this matches the common IR receivers (described above) for maximum sensitivity when communicating to another Maximite or a Micromite.

Measuring Temperature

The TEMPR() function will get the temperature from a DS18B20 temperature sensor. This device can be purchased on eBay for about $5 in a variety of packages including a waterproof probe version.



There is a position on the Colour Maximite 2's mother board for a TO-92 version of this sensor and this can be mounted so that the body of the sensor pokes out of a hole in the back panel to sense the ambient temperature. If this is installed the associated pullup resistor (4.7KΩ) on the motherboard must also be installed and pin 42 used in the function and command described below..

Sensors can also be connected to any of the pins on the eternal I/O connector. The DS18B20 should be powered separately by a 3V to 5V supply as shown on the right. Multiple sensors can be used but a separate I/O pin and a 4.7K pullup resistor is required for each one.

To get the current temperature you just use the TEMPR() function in an expression. For example:

```
PRINT "Temperature: " TEMPR(pin)
```

Where 'pin' is the I/O pin to which the sensor is connected. You do not have to configure the I/O pin, that is handled by MMBasic.

The returned value is in degrees C with a resolution of 0.25 ºC and is accurate to ±0.5 ºC. If there is an error during the measurement the returned value will be 1000.

The time required for the overall measurement is 200ms and the running program will halt for this period while the measurement is being made. This also means that interrupts will be disabled for this period. If you do not want this you can separately trigger the conversion using the TEMPR START command then later use the TEMPR() function to retrieve the temperature reading. The TEMPR() function will always wait if the sensor is still making the measurement.

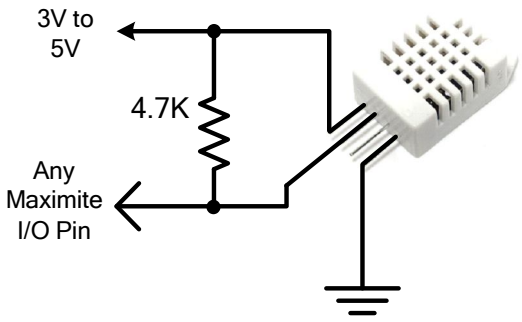
For example:

```
TEMPR START 15
< do other tasks >
PRINT "Temperature: " TEMPR(15)
```

Measuring Humidity and Temperature

The HUMID command will read the humidity and temperature from a DHT22 humidity/temperature sensor. This device is also sold as the RHT03 or AM2302 but all are compatible and can be purchased on eBay for under $5. The DHT11 sensor is also supported.

The DHT22 can be powered from 3.3V or 5V (5V is recommended) and it should have a pullup resistor on the data line as shown. This is suitable for long cable runs (up to 20 meters).



To get the temperature or humidity you use the HUMID command with three or four arguments as follows:

```
DHD22 pin, tVar, hVar [,DHT11]
```

Where 'pin' is the I/O pin to which the sensor is connected. You can use any I/O pin but if the DHT22 is powered from 5V it must be 5V capable (ie, NOT a pin that supports analog input). The I/O pin will be automatically configured by MMBasic.

'tVar' is a floating point variable in which the temperature is returned and 'hVar' is a second variable for the humidity. Both of these variables must be declared first as floats (using DIM). The temperature is returned as degrees C with a resolution of one decimal place (eg, 23.4) and the humidity is returned as a percentage relative humidity (eg, 54.3).

If the optional DHT11 parameter is set to 1 then the command will use device timings suitable for that device. In this case the results will be returned with a resolution of 1 degree and 1% humidity

For example:

```
DIM FLOAT temp, humidity
HUMID pin, temp, humidity
PRINT "The temperature is" temp " and the humidity is" humidity
```

Measuring Distance

Using a HC-SR04 ultrasonic sensor (illustrated below) and the built in DISTANCE() function you can measure the distance to a target.

This device can be found on eBay for about $4 and it will measure the distance to a target from 3cm to 3m. It works by sending an ultrasonic sound pulse and measuring the time it takes for the echo to be returned.

Compatible sensors are the SRF05, SRF06, Parallax PING and the DYP-ME007 (which is waterproof and therefore good for monitoring the level of a water tank).

You use the DISTANCE function as follows:

```
d = DISTANCE(trig, echo)
```

Where trig is the I/O pin connected to the "trig" input of the sensor and echo is the pin connected the "echo" output of the sensor. You can also use 3-pin devices and in that case only one pin number is specified.

The value returned is the distance in centimetres to the target. The I/O pins are automatically configured by this function but note that they should be 5V capable as the HC SR04 is a 5V device.

WS2812 Support

The Colour Maximite 2 has built in support for the WS2812 multicolour LED chip. This chip needs a very specific timing to work properly and with the BITBANG WS8212 command it is easy to control these devices with minimal effort.

This command will output the required signals needed to drive a chain of WS2812 LED chips connected to the pin specified and set the colours of each LED in the chain. The syntax of the command is:

```
BITBANG WS2812 type, pin, colours%()
```

Note that the pin must be set to a digital output before this command is used.

The colours%() array should be sized to have exactly the same number of elements as the number of LEDs to be driven. Each element in the array should contain the colour in the normal RGB888 format (0 - &HFFFFFF). There is no limit to the size of the WS2812 string supported.

'type' is a single character specifying the type of chip being driven as follows:

O = original WS2812

B = WS2812B

S = SK6812

As an example:

```
DIM b%(4)=(RGB(red), Rgb(green), RGB(blue), RGB(Yellow), Rgb(cyan))
SETPIN 5, DOUT
BITBANG WS2812 O, 5, b%()
```

will output the specified colours to an array of five WS2812 LEDs daisy chained off pin 5.

Hobbytronic Mouse Support

The Colour Maximite 2 has built in support for the Hobbytronic USB mouse interface.
https://www.hobbytronics.co.uk/usb-host-mouse

This is enabled using the CONTROLLER MOUSE OPEN command and then the mouse position and buttons can be interrogated using the MOUSE function

Game Playing Features

The Colour Maximite 2 has many features designed to help programmers in writing computer games. One of the most important features of this computer is its speed (approx 10 times faster than the original Colour Maximite) and its large program memory. These alone make it possible to write large and complex programs without the issues of optimising the program for speed or space.

Other useful features include managing the video output and drawing moveable images, a selection of text drawing methods and fonts, displaying images, playing audio and getting input from a keyboard or games controller.

VGA Resolution, Colour Depth and Pages

The video output to the VGA monitor is controlled by the MODE command. With this you can select various resolutions from 1280x720, 800x600 pixels (the normal default on startup) to 240x216 and various colour depths from 16-bits to 8-bits. For the 8-bit colour mode each individual colour can be specified from a 16-bit pallet using the MAP command.

Particularly useful is the 12-bit colour mode which supports 4096 colours with an additional 4-bit alpha channel allowing 16 levels of transparency. This mode has provision for two image layers plus a background colour. Images on the top layer will cover the lower layer except where the alpha channel is set to allow transparency thereby allowing various degrees of the lower layer and/or background to show through.

In addition there are many extra video pages that are available to the programmer for building images. These pages can then be copied at high speed during the video blanking period to the main display page providing an instantaneous display update without any tearing artefacts. This is managed by the PAGE WRITE command which specifies the video page to be used for the output of subsequent graphics commands and the PAGE COPY command which will copy one page to another.

When drawing to the main page being displayed on the monitor the programmer can use the GETSCANLINE function to report on the line that is currently being drawn on the VGA monitor. Using this to time updates to the screen can avoid screen glitches caused by updates while the screen is being updated. A similar outcome can be achieved using an optional feature of the MODE command which will call a user defined subroutine at the start of the VGA frame blanking.

This and other features of the graphics subsystem are explained in detail in a tutorial presented on the Back Shed forum: https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=12125 . A PDF version is included in the Colour Maximite 2 firmware zip file.

Scrolling and Sprites

The PAGE SCROLL command will scroll a page horizontally or vertically by a specified number of pixels allowing the games programmer to create a smoothly scrolling background for platform games and the like.

The Colour Maximite 2 has extensive support for sprites which is an image that can be moved over the background without disturbing the background. This feature is far more sophisticated than that available on the original Colour Maximite. The sprite can be a PNG image or an image defined in a text file and can be of any size up to the video horizontal and vertical resolution. The transparency colour can be either black or defined in the image depending on the colour depth specified by the MODE command.

Multiple sprites can be loaded and they can be moved around the screen, hidden or displayed as a group or individually. MMBasic also includes a versatile mechanism for detecting when two sprites collide allowing (for example) the programmer to create a realistic bounce effect.

Displaying Images

The Colour Maximite 2 can load and display images stored on the SD card in a variety of formats including BMP, GIF, PNG and JPG. These are read from the SD card by the LOAD command and can be of any size and positioned anywhere on the screen. In addition on-screen images can be scaled and rotated under control of the program.

Fonts

Built in to MMBasic is support for seven fonts ranging from small to large. In addition user supported fonts can be defined by the BASIC program or dynamically loaded from the SD card. All fonts can be displayed in various colours, scaled and rotated.

Playing Audio

The Colour Maximite 2 can play audio on the stereo audio output in a variety of formats using the PLAY command. These include sound effects or music encoded as WAV, FLAC or MP3 files. Also supported is the playing of MOD files which is a standard for computer generated music. The PLAY EFFECT command will allow the simultaneously playing of an WAV file over a MOD file so that sound effects can be superimposed on the computer generated music playing in the background.

Other audio formats available to the programmer include pure sine wave tones or sound effects consisting of a mixture of sine wave, square wave, triangle wave, sawtooth wave or random noise (The PLAY TONE and PLAY SOUND commands). The programmer can also change the volume, pause any audio output, resume the playing or step forward or back to the next audio file.

To round off the list of audio features the TTS command will generate a robotic speech output based on a string which defines the words and sounds to be generated.

Keyboard Keys

In games that use the keyboard for input the user will often hold down a number of keys simultaneously. This condition can be detected using the KEYDOWN() function. This will return the number of keys held down and their values in the order that they were depressed. MMBasic will track up to six simultaneous key depressions.

Wii Nunchuk and Classic Controllers

The Wii Nunchuk (also spelt Nunchuck) and the Wii Classic are inexpensive games controller which include a joystick and various buttons. MMBasic on the Colour Maximite 2 provides full support for up to three of these controllers including querying the state of all user inputs and their calibration constants.

The Nunchuk controller is opened with the CONTROLLER NUNCHUK OPEN command and its status is queried using the NUNCHUK() function. The Classic controller is opened with the CONTROLLER CLASSIC OPEN command and its status is queried using the CLASSIC() function.

Porting Programs

This chapter covers some of the considerations involved in porting programs from the original Colour Maximite to the Colour Maximite 2. There is a high degree of backwards compatibility in the Colour Maximite 2 and most programs will run with little effort, however, as can be expected, there are some differences that need to be addressed.

Most of these differences involve the more specialised functions such as graphics, input/output and some functions like the random number generator. Note that not all differences are listed here, just the more important ones that are likely to cause problems when porting programs.

Variables

In the original Colour Maximite it was possible to define variables with the same name but using a different type. For example, it was possible to use the following: `v=3.4512` and `v$="abcdefg"` in the same program (ie, the variable was defined as both a float and a string).

This is not allowed in the Colour Maximite 2, variables must be unique regardless of their type.

Floating Point

In the Colour Maximite 2 floating point is double precision. This means that if a number is printed without formatting it will contain more significant digits than the same number on the original Colour Maximite. Generally, this will not cause an issue but it might mess up numbers that need to be printed in neat columns.

Graphic Commands

The syntax of the graphic commands PIXEL, LINE, BOX and CIRCLE have completely changed. However, using the command OPTION LEGACY ON you can change these commands back to the same syntax as used in the original Colour Maximite. This option is not saved so it needs to be placed in the program before any graphic commands are used.

This option will also cause the drawing commands to accept colours in the range of 0 to 7. However, the colour shortcuts (red, blue, etc) are not defined as they were on the original Colour Maximite so, if these are required, they will need to be defined in the program as follows:

```
CONST Black = 0
CONST Blue = 1
CONST Green = 2
CONST Cyan = 3
CONST Red = 4
CONST Purple = 5
CONST Yellow = 6
CONST White = 7
```

You can also use OPTION LEGACY OFF to switch back to the Colour Maximite 2 syntax, so it is possible to mix both forms of the graphics commands in the one program.

The VGA display mode is not changed with this option so it should be specifically set to MODE 4, 8 for emulating the 480x432 resolution of the original Colour Maximite or MODE 5, 8 for emulating its 240x216 resolution.

The syntax of the MODE command has changed.

The Scan Line Colour Override function is not supported in the Colour Maximite 2.

Fonts

The FONT command has a completely different purpose and syntax in the Colour Maximite 2.

The Colour Maximite 2 has the ability to load fonts using the LOAD FONT command and you can convert the original Colour Maximite's font files to this format using the program FontTweak from: https://www.c-com.com.au/MMedit.htm

BLIT

The BLIT command is compatible with the exception that it does not implement the optional parameter specifying which colour planes to copy.

Sprites

As with the original Colour Maximite sprites can be loaded from a Maximite sprite file using the command:

```
SPRITE LOAD filename$ [,start_sprite_number]
```

This is compatible with the original Colour Maximite however the Colour Maximite 2 has a few improvements:

- An optional third parameter is available in the first line of the sprite file in which case the first parameter becomes the width and the third parameter the height. Both width and height can be set to any size.
- If only two parameters are specified the first parameter sets both the width and height and can be a number other than 16.
- Multiple sprite files can be loaded by specifying the optional 'start_sprite_number' to ensure that the sprites from different files do not overlap. This parameter defaults to 1 if not specified.

The following example shows a red mouse pointer. The first line of the file holds the parameters for the sprite (ie, the width, the number of sprites in the file, and the height in that order).

```
13,1,19
4
44
4 4
4  4
4   4
4    4
4     4
4      4
4       4
4        4
4         4
4          4
4           4
4       444444
4   4   4
4  44   4
4 4  4   4
44   4   4
       4  4
       4444
```

The above sprite can be saved as "mouse.spr". Then the following commands will load and show it:

```
SPRITE LOAD "mouse.spr", 5
SPRITE SHOW 5, 100, 100, 1
```

Note that a space character is interpreted as BLACK in 8 and 16-bit colour modes and transparent BLACK in 12-bit colour. A "0" is interpreted as NOTBLACK in 8 and 16-bit colour modes and solid BLACK in 12-bit colour.

SOUND and TONE

The SOUND command has a different purpose and syntax in the Colour Maximite 2 and should be replaced with the PWM command. The TONE command has been replaced by PLAY TONE.

SD Card and File Related Commands

There is no drive B: in the Colour Maximite 2. The SD card is drive A:.

The command CHAIN is not implemented. The Colour Maximite 2 has an exceptionally large program memory space so this feature is not necessary. The library and the LIBRARY commands are not implemented and not needed on the Colour Maximite 2. Instead of using these commands you can use #INCLUDE to insert files into the program at run time.

The commands PLAYMOD, LOADBMP and SAVEBMP have changed names. See PLAY MODFILE, LOAD BMP and SAVE IMAGE.

The DRIVE command and the predefined read-only variable MM.DRIVE$ are not supported.

Special Devices

LCD displays, Real Time Clocks (RTC) and keypads are not supported in the Colour Maximite 2. These can all be implemented in BASIC.

CONFIG Commands

Relevant CONFIG commands in the original Colour Maximite have been converted to OPTION commands.

Error Handling

Error handling using the OPTION ERROR commands has changed (see ON ERROR).

Random Number Generator

The Colour Maximite 2 has an advanced random number generator based on an analog circuit that generates a sequence of true random numbers which will never repeat. For this reason the CMM2 does not require (or allow) the programmer to seed the random generator to get different sequences.

The original Colour Maximite generated a pseudo random number sequence that always repeated with the same seed and in some rare cases this is what the programmer requires. If you need this behaviour you can use the following to generate a repeatable set of random numbers:

```
function pseudo() as float
  static seed%=7
  static a%=1103515245, c%=12345, m%=2^31
  seed%=(a% * seed% + c%) mod m%
  pseudo = seed%/m%
end function
```

Change the assignment to seed% to change the seeding number.

MMBasic Implementation Characteristics

- Maximum program size (as plain text) is 512KB. Note that MMBasic tokenises the program when it is stored in program memory so the final size in program memory might vary from the plain text size
- Maximum length of a command line is 255 characters
- Maximum length of a variable name or a label is 32 characters
- Maximum number of variables 1024: 512 global and 512 local
- Maximum number of dimensions to an array is 5
- Maximum number of arguments to commands that accept a variable number of arguments is 32
- Maximum number of nested FOR…NEXT loops is 100
- Maximum number of nested DO…LOOP commands is 100
- Maximum number of nested GOSUBs, subroutines and functions (combined) is 50
- Maximum number of nested multiline IF…ELSE…ENDIF commands is 20
- Maximum number of SELECT CASE statements is unlimited
- Maximum number of user defined subroutines and functions (combined): 500
- Maximum number of interrupt pins that can be configured: 10
- The range of floating point numbers is 1.797693134862316e+308 to 2.225073858507201e-308
- The range of 64-bit integers (whole numbers) that can be manipulated is ± 9223372036854775807
- Maximum string length is 255 characters
- Maximum line number is 65000
- Maximum number of background pulses launched by the PULSE command is 5
- Maximum number of sprites is 64 (#1 to #64)
- Maximum number of sprite collisions is 8
- Maximum length of a line in a program is 240 characters

Predefined Read Only Variables

These variables are set by MMBasic and cannot be changed by the running program.

MM.CMDLINE$	A string representing the arguments on the command line when the program was run.
MM.DEVICE$	A string representing the device or platform that MMBasic is running on. Currently this variable will contain one of the following: "Maximite" on the standard Maximite and compatibles. "Colour Maximite" on the Colour Maximite and UBW32. "Colour Maximite 2" on the **Colour Maximite 2.** "DuinoMite" when running on one of the DuinoMite family. "DOS" when running on Windows in a DOS box. "Generic PIC32" for the generic version of MMBasic on a PIC32. "Micromite" on the PIC32MX150/250 "Micromite MkII" on the PIC32MX170/270 "Micromite Plus" on the PIC32MX470 "Micromite Extreme" on the PIC32MZ series
MM.ERRNO MM.ERRMSG$	If a statement caused an error which was ignored these variables will be set accordingly. MM.ERRNO is a number where non zero means that there was an error and MM.ERRMSG$ is a string representing the error message that would have normally been displayed on the console. They are reset to zero and an empty string by RUN, ON ERROR IGNORE or ON ERROR SKIP.
MM.HRES MM.VRES	Integers representing the horizontal and vertical resolution of the VGA display in pixels.
MM.I2C	Following an I^2C write or read command this integer variable will be set to indicate the result of the operation as follows: 0 = The command completed without error. 1 = Received a NACK response 2 = Command timed out
MM.INFO() MM.INFO$()	These two versions can be used interchangeably but good programming practice would require that you use the one corresponding to the returned datatype.
MM.INFO(BCOLOUR)	Returns the current background colour
MM.INFO$(CPUSPEED)	Returns the CPU speed as a string. This will be 400000000 for the Y version of the STM32H743II or 480000000 for the V version.
MM.INFO$(CURRENT)	Returns the name of the current program or NONE if called after a NEW command
MM.INFO$(DIRECTORY)	Returns the current working directory. This will always end with a '/' character.
MM.INFO(DISK SIZE)	Returns the capacity of the SD card in bytes
MM.INFO(EXISTS FILE filename$	Returns 1 if the file filename$ exists otherwise returns 0.
MM.INFO(EXISTS DIR dirname$)	Returns 1 if the directory dirname$ exists otherwise returns 0.

MM.INFO(FCOLOUR)	Returns the current foreground colour
MM.INFO(FILESIZE file$)	Returns the size of 'file$' in bytes. Returns -1 if the file is not found. Returns -2 if file$ is the name of a valid directory
MM.INFO(FONT ADDRESS n)	Returns the address of the memory location containing the address of FONT n
MM.INFO(FONT POINTER n)	Returns a POINTER to the start of FONT n in memory
MM.INFO(FONTHEIGHT) MM.INFO(FONTWIDTH)	Integers representing the height and width of the current font (in pixels).
MM.INFO(FRAMEBUFFER)	Returns the physical memory location of the framebuffer. This is useful if you need to POKE/PEEK the contents of the page.
MM.INFO(FRAMEH)	Returns the horizontal size of the frame buffer in pixels.
MM.INFO(FRAMEV)	Returns the vertical size of the framebuffer in pixels
MM.INFO(FREE SPACE)	Returns the number of free bytes on the SD card
MM.INFO(HPOS) MM.INFO(VPOS)	The current horizontal and vertical position (in pixels) following the last graphics or print command.
MM.INFO$(KEYBOARD)	Returns the string CONNECTED if a USB keyboard is connected and working. Otherwise returns "NOT CONNECTED"
MM.INFO(MAX PAGES)	Returns the maximum page number that can be selected in the current graphics mode.
MM.INFO(MODE)	Returns the video mode as a floating point number e.g. 1.8, 2.16, etc.
MM.INFO$(MODIFIED file$)	Returns the date/time that 'file$' was last modified. File$ can be a normal file or the name of a directory. Returns an empty string if the file or directory is not found.
MM.INFO(OPTION option)	Returns the current value of a range of options that affect how a program will run. "option" can be one of ANGLE, AUTORUN, BASE, BREAK, CONSOLE, CONSOLE PORT, DEFAULT, EXPLICIT, LEGACY, MOUSE, PROFILING, USBKEYBOARD, Y_AXIS
MM.INFO(PAGE ADDRESS n)	Returns the physical memory location of page 'n'. This is useful if you need to POKE/PEEK the contents of the page.
MM.INFO(PATH)	Returns the path of the current program or NONE if called after a NEW command
MM.INFO$(PIN pinno)	Returns the status of I/O pin 'pinno'. Valid returns are: INVALID, RESERVED, IN USE, and UNUSED
MM.INFO(PROGRAM)	Returns the address in memory of the start of the program
MM.INFO$(RESET)	Returns the cause of a firmware restart. The returned value will be one of "Switch" (i.e. pressing the reset switch), "Power-On", "Software", and "Watchdog" (NB the latter is the H/W watchdog and unrelated to the MMbasic version which causes a software reset)
MM.INFO$(SDCARD)	Returns the status of the SD card. Valid returns are: DISABLED, NOT PRESENT, READY, and UNUSED

MM.INFO$(SEARCH PATH)	Returns the string set as the search path by OPTION SEARCH PATH
MM.INFO$(SOUND)	Returns the status of the sound output device. Valid returns are: OFF, PAUSED, TONE, WAV, MP3, MODFILE, TTS, FLAC, DAC, SOUND
MM.INFO$(TRACK)	The name of the current audio track playing. This returns "OFF" if nothing is playing.
MM.INFO(VERSION)	The version number of the firmware as a floating point number in the form aa.bbcc where aa is the major version number, bb is the minor version number and cc is the revision number. For example version 5.03.00 will return 5.03 and version 5.03.01 will return 5.0301.
MM.INFO(WRITE PAGE)	Returns the address in memory of the page to which writes will take place
MM.INFO(UPTIME)	Returns the time since rebooting the CMM2 in seconds
MM.ONEWIRE	Following a 1-Wire reset function this integer variable will be set to indicate the result of the operation as follows: 0 = Device not found. 1 = Device found
MM.WATCHDOG	An integer which is true (ie, 1) if MMBasic was restarted as the result of a Watchdog timeout (see the WATCHDOG command). False (ie, 0) if MMBasic started up normally.

Operators

The following operators are listed in order of precedence. Operators that are on the same level (for example + and -) are processed with a left to right precedence as they occur on the program line.

Numeric Operators (Float or Integer)

NOT INV	NOT will invert the logical value on the right. INV will perform a bitwise inversion of the value on the right. Both of these have the highest precedence so if the value being operated on is an expression it should be surrounded by brackets. For example, `IF NOT (A = 3 OR A = 8) THEN …`
^	Exponentiation (eg, `b^n` means b^n)
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction
x << y x >> y	These operate in a special way. << means that the value returned will be the value of x shifted by y bits to the left while >> means the same only right shifted. They are integer functions and any bits shifted off are discarded. For a left shift any bits introduced are set to zero. >> is an unsigned right shift where the top bit is set to 0 >>> is a signed right shift and any bits introduced are set to the value of the top bit (bit 63).
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	Equality (also used in assignment to a variable, eg implied LET).
AND OR XOR	Conjunction, disjunction, exclusive or. These are bitwise operators and can be used on 64-bit unsigned integers.

The operators AND, OR and XOR are integer bitwise operators. For example PRINT (3 AND 6) will output 2.

The other logical operations result in the integer 0 (zero) for false and 1 for true. For example the statement PRINT 4 >= 5 will print the number zero on the output and the expression A = 3 > 2 will store +1 in A.

String Operators

+	Join two strings
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version).
=	Equality

String comparisons respect the case of the characters (ie "A" is greater than "a").

Options

This table lists the various option commands which can be used to configure MMBasic and change the way it operates. Options that are marked as permanent will be saved in non volatile memory and automatically restored when the Colour Maximite 2 is restarted. Options that are not permanent will be reset on startup.

	Permanent?	
OPTION ANGLE RADIANS | DEGREES		This command switches trig functions between degrees and radians. Acts on SIN, COS, TAN, ATN, ATAN2, MATH ATAN3, ACOS, ASIN This is a temporary option that is cleared to default (RADIANS) when programs end, after an error, or after Ctrl-C so should be set at the top of any program that requires to use angles in degrees.
OPTION AUTORUN OFF | ON	✓	Instructs MMBasic to automatically run the program in program flash memory on power up or restart (eg, by the WATCHDOG timer). This is turned off by the NEW command but other commands that might change program memory (EDIT, etc) do not change this setting. Entering the break key (default CTRL-C) at the console will interrupt the running program and return to the command prompt. If the CMM2 is set to run programs from RAM (OPTION RAM) then the firmware will look for the file AUTORUN.BAS on reset or power up and execute it if found.
OPTION Y_AXIS DOWN | UP		This command can only be used in a program and inverts the y axis for drawing commands. Ie, with UP set 0,0 is at the bottom left as in a typical graphing application. This is a temporary option that is cleared to default (DOWN) when programs end, after an error, or after Ctrl-C.
OPTION BASE 0 | 1		Set the lowest value for array subscripts to either 0 or 1. This must be used before any arrays are declared and is reset to the default of 0 on power up.
OPTION BASELINE ON/OFF	✓	This sets the CMM2 to run as a 400MHz first generation machine and is intended to allow testing of code on the lowest common denominator.
OPTION BAUDRATE nbr	✓	Set the baud rate for the serial console to 'nbr'. This can be any value between 1200 (the minimum) and 1000000 (1MHz). This change is made immediately and will be remembered when the power is cycled. Using this command it is possible to set the console to an unworkable baud rate and in this case the baudrate should be reset using a USB keyboard and VGA monitor. If that is not available then reseting the firmware will reset the baudrate to the default of 115200.
OPTION BREAK nn		Set the value of the break key to the ASCII value 'nn'. This key is used to interrupt a running program. The value of the break key is set to CTRL-C key at power up but it can be changed to any keyboard key using this command (for example, OPTION BREAK 4 will set the break key to the CTRL-D key). Setting this option to zero will disable the break function entirely.

OPTION COLOURCODE ON or OPTION COLOURCODE OFF Or OPTION COLOURCODE REVERSE	✓	Turn on or off colour coding for the editor's output. Keywords will be in cyan, comments in yellow, etc. The default is ON. OPTION COLOURCODE REVERSE is the same as OPTION COLORCODE OFF except that the text will be in reverse video black on white. This will apply to the file manager and editor. The keyword COLORCODE (USA spelling) can also be used. On the serial console colour coding requires a terminal emulator that can interpret the appropriate escape codes.
OPTION CONSOLE PORT n	✓	Allow you to select the serial port to be used as the console. 'n' can be 1, 2, or 3 (default). If this option is set all console output is directed to the specified com port. This allows easier remote operation using a wifi-uart or radio link. If the option is set to other than default then COM3 is available on the USB-B connector.
OPTION CONSOLE SCREEN or OPTION CONSOLE SERIAL or OPTION CONSOLE BOTH		OPTION CONSOLE SCREEN will disable the serial console for both input and output and direct all output to the VGA monitor. This will allow the VGA output to update much faster. With this option enabled the serial port used for the console can be opened as COM3. OPTION CONSOLE SERIAL will disable the console output to the VGA screen and send all output to the serial console. This is useful for debugging graphics applications as diagnostic PRINT statements will not corrupt the screen display. OPTION CONSOLE BOTH will enable both the serial and VGA screen for console input/output. This is the default on power up unless OPTION CONSOLE SAVE (see below) is used.
OPTION CONSOLE SAVE	✓	This will save the current console mode (see above) as the default stored mode. If you are using the CMM2 as a stand alone computer it is recommended that you execute OPTION CONSOLE SCREEN, then OPTION CONSOLE SAVE to permanently disable the serial console and thereby eliminate serial I/O overhead.
OPTION CRLF mode		Defines what the USB keyboard will send when the Enter key is pressed. 'mode' can be one of CR, LF or CRLF. The default is CRLF.
OPTION DEFAULT FLOAT | INTEGER | STRING | NONE		Used to set the default type for a variable which is not explicitly defined. If OPTION DEFAULT NONE is used then all variables must have their type explicitly defined. When a program is run the default is set to FLOAT for compatibility with previous versions of MMBasic.
OPTION DEFAULT MODE n	✓	Specifies the video mode for the command prompt, the editor and the file manager and is the default mode when a program is run. n can be: 1 = 800x600 (default) 8 = 640x480 9 = 1024x768 10 = 848x480 (widescreen) 11 = 1280x720 (widescreen) 12 = 960x540 (widescreen) 14 = 960x540 (widescreen, preferred over 12 but may not work on all monitors)

		15 = 1280x1024 16 = 1920x1080 (widescreen, G2 only) This setting is remembered even after a firmware upgrade.
OPTION DS3231 ON/OFF	✓	Generation 2 only. Enables and disables the use of the DS3231 RTC for providing the time and date. NB: if OPTION MILLISECONDS is ON the internal RTC will always be used for providing the time
OPTION EXPLICIT		Placing this command at the start of a program will require that every variable be explicitly declared using the DIM, LOCAL or STATIC commands before it can be used in the program. This option is disabled by default when a program is run. If it is used it must be specified before any variables are used.
OPTION EDIT FONT SMALL | NORMAL | MEDIUM | LARGE| VERY LARGE	✓	Sets the font to be used in the editor and file manager. The default is NORMAL which is a 8x12 pixel font.
OPTION FLASH [n]	✓	Specifies that programs are to be run from FLASH memory and will therefore be preserved when the CMM2 is powered off or reset. The optional parameter 'n' specifies the starting point in 128Kbyte pages of memory. 'n' defaults to 0. For 0<=n<=4 the maximum program size is 512KBytes for n=5 the maximum program size is 384KBytes for n=6 the maximum program size is 256KBytes for n=7 the maximum program size is 128KBytes Using the OPTION FLASH command can increase flash lifespan by a factor of 8 although it is unlikely ever to be needed for most users. See also OPTION RAM.
OPTION F11 string$	✓	Define the string that will be generated when the F11 function key is pressed at the command prompt. Example: OPTION F11 "RUN "+chr$(34)+"myprog" +chr$(34)+chr$(13)+chr$(10). The maximum string length is 64 characters.
OPTION F12 string$	✓	Define the string that will be generated when the F12 function key is pressed at the command prompt. The maximum string length is 64 characters.
OPTION KEYBOARD REPEAT firstchar, nextchar	✓	Define the repeat characteristics of the USB keyboard when a key is held down. 'firstchar' is the time in milliseconds before a new character repeats. Default is 600mSec, the valid range is 100 to 2000 mSec 'nextchars' is the time in milliseconds before subsequent character repeats. Default is 150mSec, the valid range is 25 to 2000 mSec
OPTION LEGACY ON or OPTION LEGACY OFF		This will turn on or off compatibility mode with the original Colour Maximite. Commands such as LINE, CIRCLE and PIXEL work as they originally did and all drawing commands will accept colours in the range of 0 to 7. Notes: • Keywords such as RED, BLUE, etc are not implemented so they should be defined as constants if needed. • The VGA display mode is not changed with this option so it

		should be specifically set to MODE 4 for emulating the 480x432 resolution of the original Colour Maximite or MODE 5 for emulating the 240x216 resolution. The colour depth must be set to 8-bits.
OPTION LIST		This will list the settings of any options that have been changed from their default setting and are the permanent type.
OPTION MILLISECONDS ON or OPTION MILLISECONDS OFF		Specifies that the TIME$ function will, or will not, include milliseconds as a decimal fraction of seconds in its output. The default is OFF.
OPTION MOUSE n [,sensitivity] OPTION MOUSE OFF	✓	Enables a mouse to be used in the filemanager and editor. Use channel 0 for a PS2 mouse, or 1 to 3 for a Hobbytronic connected mouse. Note if the mouse is enabled but not plugged in and working you will get an error when entering the filemanager or editor. Use OPTION MOUSE OFF to disable.
OPTION OVERCLOCK ON/OFF	✓	This overclocks the CMM2 to 504MHz which is fastest speed at which most of both the original and Generation 2 versions of the CMM2 run reliably
OPTION PIN nbr		Set 'nbr' as the PIN (Personal Identification Number) for access to the console prompt. 'nbr' can be any non zero number of up to eight digits. Whenever a running program tries to exit to the command prompt for whatever reason MMBasic will request this number before the prompt is presented. This is a security feature as without access to the command prompt an intruder cannot list or change the program in memory or modify the operation of MMBasic in any way. To disable this feature enter zero for the PIN number (ie, OPTION PIN 0). A permanent lock can be applied by using 99999999 for the PIN number. If a permanent lock is applied or the PIN number is lost the only way to recover is to reset the Colour Maximite 2 firmware (as described in the section *Resetting MMBasic*).
OPTION PROFILING ON|OFF	✓	Turns on or off profiling. With profiling on you will see a performance impact of about 5% caused by the profiling data collection which happens at the MMBasic statement level. In addition you will see a reduction in available variable RAM of 512KB. Note that profiling and OPTION RAM are mutually incompatible on the original CMM2 H/W but not the Generation 2 With profiling enabled you can run your programs as usual and you should see no change in behaviour other than as mentioned above. You can see the way the program has run using the commands See also LIST PROFILE
OPTION RAM	✓	OPTION RAM causes the program to be loaded into RAM to run rather than flash memory. This makes loading somewhat faster and avoids impacting the flash write life. Program performance is similar to running from flash (less than 1% slower). Programs are lost with power, reset, or option change. Variable storage is reduced by 512Kb on the original CMM2 H/W but not the

		Generation 2 See also OPTION FLASH.
OPTION RESET	✓	Reset all saved options to their default values.
OPTION RTC CALIBRATE ±n	✓	Used to calibrate the battery backed Real Time Clock that keeps time in the Colour Maximite 2. 'n' is a number between -511 and + 512. A change of ±1 should equate to about 0.0824 seconds per day. Negative numbers will slow the clock down, positive will speed it up (different from the Micromite). This setting is remembered even after a firmware upgrade.
OPTION SEARCH PATH pathname$	✓	This defines a path which will be searched when you use the existing RUN command or the short form RUN command (*) if the file does not exist in the current directory
OPTION SLEEP n	✓	n is the number of minutes (0-255) before the monitor is turned off when at the command line. Set to 0 to disable. Any keypress will wake the monitor
OPTION SD TIMING NORMAL or OPTION SD TIMING FAST	✓	The fast option will speed up the timing for SD card access. This results in read/write speeds being about 20% faster where cards can accommodate the higher speed access. The default is normal. This setting is remembered even after a firmware upgrade.
OPTION SERIAL PULLUP ENABLE or OPTION SERIAL PULLUP DISABLE	✓	Enable or disable pullup resistors on the receive line of all serial ports including the serial console. The default is disabled.
OPTION STATUS ON | OFF	✓	Enable or disable the status line at the bottom of the VGA screen. The status line shows the date/time and the "current program filename" used by the commands RUN, LIST and EDIT when a file name is not specified. Default is ON.
OPTION TAB 2 | 3 | 4 | 8	✓	Set the spacing for the tab key. Default is 2.
OPTION USBKEYBOARD nn [,noLED]	✓	Set the language type for the attached USB keyboard. ‘nn is a two character code defining the keyboard layout. The choices are US for the standard keyboard layout in the USA, Australia and New Zealand and UK for the United Kingdom, DE for Germany, FR for France and ES for Spain. The optional noLED parameter can be set to 1 to block sending the command to the keyboard that lights the LEDs relating to Caps Lock etc. This may be needed on some keyboards which do not process this command properly and may lock up. It defaults to 0 if not specified (i.e. the LED commands are sent). This command can only be run from the command line and will cause a reboot of the CMM2. This setting is remembered even after a firmware upgrade.

OPTION VCC voltage		Specifies the voltage (Vcc) supplied to the ARM Cortex-M7 chip. When using the analog inputs to measure voltage the ARM Cortex-M7 chip uses its supply voltage (Vcc) as its reference. This voltage can be accurately measured using a DMM and set using this command for more accurate measurement. The parameter is not saved and should be initialised either on the command line or in a program. The default if not set is 3.3.

Commands

Square brackets indicate that the parameter or characters are optional.

‘ (single quotation mark)	Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.
? (question mark)	Shortcut for the PRINT command.
* (asterix)	Synonym for the RUN command at the command prompt e.g. *myprog any test string Will run the program myprog.bas and pass it the command line “any test string” in MM.CMDLINE$
#COMMENT START #COMMENT END	Directive to allow multi-line comments. The command must be in capitals. Any lines between the two commands are completely ignored and not loaded into memory
#DEFINE “before”, “after”	This will cause all copies of the string “before” in a program to be replaced by the string “after”. Both parameters must be literal quoted strings. Matches within quoted strings in the program are ignored. DEFINEs are executed in reverse order of creation so a symbol can be redefined and from that point on in the program the new definition will be active. Case is ignored in the strings in the DEFINE directive. The program can support up to 256 #define statements.
#INCLUDE file$	This will insert the file 'file$' into the program at that point. This file must be resident on the SD card and must have the extension ".INC". Inserting the text is performed by the pre-processor when the program is loaded into program memory by the RUN command or on exiting EDIT or AUTOSAVE using F2. Because this operation is performed before the program is run it is recommended that include files are specified relative to the directory holding the program or with full pathnames. Within the program the command CHDIR will be executed at runtime so will not affect MMBasic's ability to locate include files. This command acts exactly as if the included file was manually inserted into the code using an editor but it is more convenient for loading libraries and other static code fragments. It essentially replaces the LIBRARY command on the original Maximite. Runtime errors in the included file are reported with the file name and line number in the file. The firmware will automatically check for changes in include files when a program is RUN and update the program if required.
#MMDEBUG ON #MMDEBUG OFF	These can appear anywhere in the program and are used by the program loader. If #MMDEBUG is OFF (default condition) then any lines starting with the command MMDEBUG are ignored (effectively treated as comments) and will have absolutely zero impact on program performance - they are simply not loaded into program memory. If #MMDEBUG is ON then lines starting MMDEBUG are included in the program. See the MMDEBUG command for more details

ADC	The ADC functionality can capture up to 3 channels of analog data in the background at up to 500KHz per channel (480KHz for 400MHz processors) with user selectable triggering
ADC OPEN frequency, channel1-pin [,channel2-pin] [,channel3-pin] [, interrupt]	Open the ADC channels. "frequency" is the sampling frequency in Hz. Above 160KHz the conversion is 8-bits per channel From 40KHz to 160KHz the conversion is 10-bits per channel From 20KHz to 40KHz the conversion is 12-bits per channel From 10KHz to 20KHz the conversion is 14-bits per channel Below 10KHz conversion is 16-bits per channel This is automatically applied in the firmware. 'channel1-pin' can be one of 7,10,16,22,24,37 'channel2-pin' can be one of 8,12,26,29 'channel3-pin' can be one of 13,15 'interrupt' is a normal MMBasic subroutine that will be called when the conversion completes.
ADC FREQUENCY frequency	Allows the ADC frequency to be adjusted after the ADC START command. This command is only valid if the number of bits calculated in the table above does not change.
ADC TRIGGER channel, level	Sets up triggering of the ADC. This should be specified before the ADC START command. The 'channel' can be a number between one and three depending on the number of pins specified in the ADC OPEN command. The 'level' can be between -VCC and VCC. A positive number indicates that the trigger will be on a positive going transition through the specified voltage. A negative number indicates a negative going transition through the specified voltage.
ADC START channel1array!() [,channel2array!()] [,channel3array!()]	Starts ADC conversion. The floating point arrays must be the same size and their size will determine the number of samples. Once the start command is issued the ADC(s) will start converting the input signals into the arrays at the frequency specified. If the OPEN command includes an interrupt then the command will be non-blocking. If an interrupt is not specified the command will be blocking. The samples are returned as floating point values between 0 and VCC.
ADC CLOSE	Closes the ADC and returns the pins to normal use
ARC x, y, r1, [r2], rad1, rad2, colour	Draws an arc of a circle or a given colour and width between two radials (defined in degrees). Parameters for the ARC command are: 'x' is the X coordinate of the centre of arc. 'y' is the Y coordinate of the centre of arc. 'r1' is the inner radius of the arc. 'r2' is the outer radius of the arc - can be omitted if 1 pixel wide. 'rad1' is the start radial of the arc in degrees. 'rad2' is the end radial of the arc in degrees. 'colour' is the colour of the arc.
AUTOSAVE file$	Enter automatic program entry mode. This command will take lines of text from the console serial input and save them to a file on the SD card specified as 'file$'. This mode is terminated by pressing F1 on the console keyboard which will then cause the received data to be saved to the SD card.

	Terminating the transfer by pressing F2 will cause a similar save but then the saved program will be immediately loaded into program memory and run. Both F1 and F2 update the “current program name” which is used by RUN, LIST and EDIT when a file is not specified. The transfer can also be terminated using F6 which acts the same as F1 without updating the current program name. At any time this command can be aborted by Control-C which will leave program memory untouched. This is one way of transferring a BASIC program into the Maximite. The program to be transferred can be pasted into a terminal emulator and this command will capture the text stream and store it into program memory. It can also be used for entering a small program directly at the console input.
BITBANG BITSTREAM pinno, n_transitions, array%()	This command is used to generate an extremely accurate bit sequence on the pin specified. The pin must have previously been set up as an output and set to the required starting level. Notes: • The array contains the length of each level in the bitstream in microseconds. The maximum period allowed is 65.5 mSec • The first transition will occur immediately on executing the command. • The last period in the array is ignored other than defining the time before control returns to the program or command line. • The pin is left in the starting state if the number of transitions is even and the opposite state if the number of transitions is odd.
BITBANG WS2812 type, pin, colours%()	This command outputs the required signals to drive one or more WS2812 LED chips connected to 'pin'. Note that the pin must be set to a digital output before this command is used. 'type' is a single character specifying the type of chip being driven: O = original WS2812 B = WS2812B S = SK6812 The 'colours%()' array should be an integer array sized to have exactly the same number of elements as the number of LEDs to be driven. Each element in the array should contain the colour in the normal RGB888 format (ie, 0 to &HFFFFFF).
BLIT READ [#]b, x, y, w, h [,pagenumber] or BLIT WRITE [#]b, x, y [,orientation] or BLIT CLOSE [#]b	Copy one section of the display screen to or from a memory buffer. BLIT READ will copy a portion of the display to the memory buffer '#b'. The source coordinate is 'x' and 'y' and the width of the display area to copy is 'w' and the height is 'h'. When this command is used the memory buffer is automatically created and sufficient memory allocated. The optional parameter page number specifies which page is to be read. The default is the current write page. This buffer can be freed and the memory recovered with the BLIT CLOSE command. Set the pagenumber to FRAMEBUFFER to read from the framebuffer – see the FRAMEBUFFER command BLIT WRITE will copy the memory buffer '#b' to the display. The destination coordinate is 'x' and 'y' using the width/height of the buffer. The optional 'orientation' parameter defaults to 4 and specifies how the stored image data is changed as it is written out. It is the bitwise AND of the following values: &B001 = mirrored left to right &B010 = mirrored top to bottom &B100 = don't copy transparent pixels

	BLIT CLOSE will close the memory buffer '#b' to allow it to be used for another BLIT READ operation and recover the memory used. Notes: • Sixty four buffers are available ranging from #1 to #64. • When specifying the buffer number the # symbol is optional. • All other arguments are in pixels.
BLIT x1, y1, x2, y2, w, h [, page] [,orientation]	Copy one section of the display screen to another part of the display. The source coordinate is 'x1' and 'y1'. The destination coordinate is 'x2' and 'y2'. The width of the screen area to copy is 'w' and the height is 'h'. 'page' is the page number that the image data is read from; it is then written to the current write page as specified by the PAGE WRITE n command. If 'page' is omitted the data is read from the write page. Set the page to FRAMEBUFFER to read from the framebuffer – see the FRAMEBUFFER command. All arguments are in pixels and the source and destination can overlap. The optional 'orientation' parameter specifies how the section of the screen is changed as it is copied. It is the bitwise AND of the following values: &B001 = mirrored left to right &B010 = mirrored top to bottom &B100 = don't copy transparent pixels
BOX x, y, w, h [, lw] [,c] [,fill]	Draws a box on the VGA monitor with the top left hand corner at 'x' and 'y' with a width of 'w' pixels and a height of 'h' pixels. 'lw' is the width of the sides of the box and can be zero. It defaults to 1. 'c' is the colour and defaults to the default foreground colour if not specified. 'fill' is the fill colour. It can be omitted or set to -1 in which case the box will not be filled. All parameters can be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. 'x', 'y', '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 chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.
BOX AND_PIXELS x, y, w, h, colour [,pageno] BOX OR_PIXELS x, y, w, h, colour [,pageno] BOX XOR_PIXELS x, y, w, h, colour [,pageno]	Executes the requested logical operation between the pixels in the area defined on the page specified (defaults to the write page) with the colour specified
CALL usersubname$ [,usersubparameters,....]	This is an efficient way of programmatically calling user defined subroutines (see also the CALL() function). In many case it can allow you to get rid of complex SELECT and IF THEN ELSEIF ENDIF clauses and is processed in a much more efficient way. The “usersubname$” can be any string or variable or function that resolves to the name of a normal user subroutine (not an in-built command). The “usersubparameters” are the same parameters that would be used to call the subroutine directly. A typical use could be writing any sort of emulator where one of a large number of subroutines should be called depending on some variable. It also allows a way of passing a subroutine name to another subroutine or function as a variable.

CAT S$, N$	Concatenates the strings by appending N$ to S$. This is functionally the same a S$ = S$ + N$ but operates faster.
CHDIR dir$	Change the current working directory on the SD card to ‘dir$’ The special entry “..” represents the parent of the current directory and “.” represents the current directory. "/" is the root directory.
CIRCLE x, y, r [,lw] [, a] [, c] [, fill]	Draw a circle on the video output centred at 'x' and 'y' with a radius of 'r' on the VGA monitor. ‘lw’ is optional and is the line width (defaults to 1). 'c' is the optional colour and defaults to the current foreground colour if not specified. The optional 'a' is a floating point number which will define the aspect ratio. If the aspect is not specified the default is 1.0 which gives a standard circle 'fill' is the fill colour. It can be omitted or set to -1 in which case the circle will not be filled. All parameters can be expressed as arrays and the software will plot the number of circles 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 chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.
CLEAR	Delete all variables and recover the memory used by them.
CLOSE [#]nbr [,[#]nbr] …	Close the file(s) previously opened with the file number ‘#fnbr’ Close the serial communications port(s) previously opened with the file number ‘nbr’. The # is optional. Also see the OPEN command. The text “GPS” can be substituted for [#]nbr to close a communications port used for a GPS receiver.
CLS [colour]	Clears the VGA screen and the terminal emulator's screen. Optionally 'colour' can be specified which will be used for the VGA background when clearing the screen. Note: the terminal emulator screen is only cleared if the current write page is page 0
COLOUR fore [, back] or COLOR fore [, back]	Sets the default colour for commands (PRINT, etc) that display on the on the VGA monitor. 'fore' is the foreground colour, 'back' is the background colour. The background is optional and if not specified will default to black.
CONST id = expression [, id = expression] … etc	Create a constant identifier which cannot be changed once created. 'id' is the identifier which follows the same rules as for variables. The identifier can have a type suffix (!, %, or $) but it is not required. If it is specified it must match the type of 'expression'. 'expression' is the value of the identifier and it can be a normal expression (including user defined functions) which will be evaluated when the constant is created. A constant defined outside a sub or function is global and can be seen throughout the program. A constant defined inside a sub or function is local to that routine and will hide a global constant with the same name.
CONTINUE	Resume running a program that has been stopped by an END statement, an error, or CTRL-C. The command is also used to exit a BREAK state and resume program execution following a MMDEBUG BREAK command. The program will restart with the next statement following the previous stopping point.

	Note that it is not always possible to resume the program correctly – this particularly applies to complex programs with graphics, music, nested loops and/or nested subroutines and functions.
CONTINUE DO or CONTINUE FOR	Skip to the end of a DO/LOOP or a FOR/NEXT loop. The loop condition will then be tested and if still valid the loop will continue with the next iteration.
COPY fname1$ TO fname2$	Copy a file from ‘fname1$’ to ‘fname2$’. Both are strings. A directory path can be used in both 'fname$' and 'fname$'. If the paths differ the file specified in 'fname$' will be copied to the path specified in 'fname2$' with the file name as specified.
CONTROLLER CLASSIC OPEN [n] [,interrupt [,bitmask]] or CONTROLLER CLASSIC CLOSE [n]	Opens a Classic port for communications. 'n' is the I2C channel that the Classic is connected to. If not specified channel 3 is opened (the front panel Wii connector). 'interrupt' is an optional MMbasic interrupt routine that will be triggered when buttons are pressed. The interrupts trigger once on the button press and will not re-interrupt until it is released. The bitmask specifies which of the 15 buttons on the Wii Classic will trigger an interrupt. The default is that they all will. See the CLASSIC function for details of which bits in the bitmask control which buttons. The firmware checks for the existence of a controller on open and will throw error if not found. Once open the Classic is polled continuously in the background once every 16mSec. The I2C transfers are under interrupt control and non-blocking for the rest of the firmware. The I2C channel is opened at 100KHz - other I2C devices can share the I2C port with the Classic if required in which case I2C(n) OPEN should not be called. CONTROLLER CLASSIC CLOSE will close the Wii port ‘n’. If the parameter is omitted channel 3 is closed. Also see the CLASSIC() function for accessing the state of the device.
CONTROLLER MOUSE OPEN [n] [,LBinterrupt][,RBinterrupt] [,sensitivity][,LBUpinterrupt] or CONTROLLER MOUSE CLOSE [n]	Opens a Hobbytronic Mouse port for communications. 'n' is the I2C channel that the mouse controller is connected to. If not specified channel 2 is opened (pins 27 and 28 on the I/O port). 'LBinterrupt' and 'RBinterrupt' are optional MMbasic interrupt routines that will be triggered when one or other of the mouse buttons is pressed. The interrupts trigger once on the button press and will not re-interrupt until it is released.The firmware checks for the existence of a controller on open and will throw error if not found. This error can be trapped with the ON ERROR command. Once open the mouse I/F is polled continuously in the background once every 16mSec. The I2C transfers are under interrupt control and non-blocking for the rest of the firmware. The I2C channel is opened at 100KHz - other I2C devices can share the I2C port with the mouse if required in which case I2C(n) OPEN should not be called. The sensitivity of the Hobbytronic mice can be controller by specifying a number between 0 and 10. 0 indicates that the default automatic sensitivity should be used. Other values will explicitly set the sensitivity as per the Hobbytronic documentation. LBUpinterrupt is an optional interrupt routine which is called when the left button is releasedCONTROLLER MOUSE CLOSE will close the mouse port ‘n’. If the parameter is omitted channel 2 is closed. Also see the MOUSE() function for accessing the state of the device.

CONTROLLER MOUSE OPEN 0 [,LBinterrupt [,RBinterrupt]] or CONTROLLER MOUSE CLOSE 0	Opens a PS2 port for communications. On the original CMM2 to make the connection the mouse clock line is connected to pin 33 of the CMM2 and the mouse data to pin 32. Both lines MUST be pulled up to 5V with 4K7 resistors. On the CMM2 Generation 2 additional pins are allocated to the PS2 mouse and it is connected to the lower USB port. Pullup resistors are already fitted to the CMM2 Generation 2 PCB. 'LBinterrupt' and 'RBinterrupt' are optional MMbasic interrupt routines that will be triggered when one or other of the mouse buttons is pressed. The interrupts trigger once on the button press and will not re-interrupt until it is released. The PS2 mouse sensitivity can be set between 0 and 8. Values of 1-4 set the resolution of the mouse to 1,2, 4 or 8 counts/mm. Values 5-8 do the same but in addition enable mouse scaling which gives a non-linear relationship between speed and count. See https://isdaman.com/alsos/hardware/mouse/ps2interface.htm for more details. NB: not all PS2 mice seem to implement these commands. The firmware checks for the existence of a controller on open and will throw error if not found. This error can be trapped with the ON ERROR command. Once open the mouse I/F operates under interrupt control so has no performance impact unless the mouse is movinhg. CONTROLLER MOUSE CLOSE 0 will close the PS2 mouse port. Also see the MOUSE() function for accessing the state of the device.
CONTROLLER NUNCHUK OPEN [n] [,Zinterrupt [,Cinterrupt]] or CONTROLLER NUNCHUK CLOSE [n]	Opens a Nunchuk port for communications. 'n' is the I2C channel that the Nunchuk is connected to. If not specified channel 3 is opened (the front panel Wii connector). 'Cinterrupt' and 'Zinterrupt' are optional MMbasic interrupt routines that will be triggered when one or other of the buttons is pressed. The interrupts trigger once on the button press and will not re-interrupt until it is released. The firmware checks for the existence of a controller on open and will throw error if not found. This error can be trapped with the ON ERROR command. If a classic controller is plugged in instead, an error will not be thrown, instead you can check the ID code with NUNCHUK(T, channel). Once open the Nunchuk is polled continuously in the background once every 16mSec. The I2C transfers are under interrupt control and non-blocking for the rest of the firmware. The I2C channel is opened at 100KHz - other I2C devices can share the I2C port with the Nunchuk if required in which case I2C(n) OPEN should not be called. CONTROLLER NUNCHUK CLOSE will close the Nunchuk port ‘n’. If the parameter is omitted channel 3 is closed. Also see the NUNCHUK() function for accessing the state of the device.
CSUB name [type [, type] …] hex [[hex[…] hex [[hex[…] END CSUB	Defines the binary code for an embedded machine code program module written in C or ARM assembler. The module will appear in MMBasic as the command 'name' and can be used in the same manner as a built-in command. Multiple embedded routines can be used in a program with each defining a different module with a different 'name'. The first 'hex' word is a 32 bit word which is the offset in bytes from the start of the CSUB to the entry point of the embedded routine (usually the function main()). The following hex words are the compiled binary code for the module. These are automatically programmed into MMBasic when the program is saved. Each 'hex' must be exactly eight hex digits representing the bits in a 32-bit word and be separated by one or more spaces or new

	lines. The command must be terminated by a matching END CSUB. Any errors in the data format will be reported when the program is run. During execution MMBasic will skip over any CSUB commands so they can be placed anywhere in the program. The type of each parameter can be specified in the definition. For example: `CSub MySub integer, integer, string`. This specifies that there will be three parameters, the first two being integers and the third a string. Note: • Up to ten arguments can be specified ('arg1', 'arg2', etc). • If a variable or array is specified as an argument the C routine will receive a pointer to the memory allocated to the variable or array and the C routine can change this memory to return a value to the caller. In the case of arrays, they should be passed with empty brackets e.g. arg(). In the CSUB the argument will be supplied as a pointer to the first element of the array. • Constants and expressions will be passed to the embedded C routine as pointers to a temporary memory space holding the value. • CSUBs must call routinechecks() every millisecond or so both to keep the USB keyboard active and also ensure the watchdog doesn't trigger. CSUBs that run to completion within a couple of milliseconds can ignore this.
CPU RESTART	Will force a restart of the processor. This will clear all variables and reset everything (eg, timers, COM ports, I^2C, etc) similar to a power up situation but without the power up banner. If OPTION AUTORUN has been set the program will restart.
DAC n, voltage	Sets the DAC channel (1 or 2) to the voltage requested. This command cannot be used if the DACs are in use for audio output.
DAC START frequency, DAC1array%() [,DAC2array%()] [,interrupt]	Sets up the DAC to create an arbitrary waveform. DAC1array%() and optional DAC2array%() should contain numbers in the range 0-4095 to suit the 12-bit DACs. The output occurs in the background and control returns to MMBasic immediately. The output runs continuously unless the optional interrupt is specified. In this case the contents of the array(s) is played once and the interrupt is triggered on completion. The software automatically and separately uses the number of items in each of the arrays to drive the DACs. The frequency is the rate at which the DACs change value. The maximum frequency is 700KHz. As an example if there are 180 items in the array c%() which are displayed at a frequency of 100,000 Hz this will give a waveform frequency of 100,000/180 = 555Hz. If there are 90 items in the array d%() at the same frequency of 100,000 Hz this will at the same time produce a waveform frequency of 100,000/90 = 1111Hz.
DAC STOP	Stops the DAC output and returns the DACs to normal use.
DATA constant[,constant]...	Stores numerical and string constants to be accessed by READ. In general string constants should be surrounded by double quotes ("). An exception is when the string consists of just alphanumeric characters that do not represent MMBasic keywords (such as THEN, WHILE, etc). In that case quotes are not needed. Numerical constants can also be expressions such as 5 * 60.

DATE$ = "DD-MM-YY" or DATE$ = "DD/MM/YY"	Set the date of the internal clock/calendar. DD, MM and YY are numbers, for example: DATE$ = "28-7-2024" The year can be abbreviated to two digits (ie, 24). The date is set to "01-01-2000" on first power up but the date will be remembered and kept updated as long as the battery is installed and can maintain a voltage of over 2.5V. Battery life should be 3 to 4 years even if the CMM2 is left powered off. Note that the time (set using the TIME$= command) will be lost when the power is cycled if a correct date is not set.
DEFINEFONT #n hex [[hex[…] hex [[hex[…] END DEFINEFONT	This will define an embedded font which can be used exactly same as the built in fonts (ie, selected using the FONT command or specified in the TEXT command). MMBasic must execute the font in order for it to be loaded. '#n' is the font's reference number (1 to 16). It can be the same as an existing font (except fonts 1, 6 and 7) and in that case it will replace that font. Each 'hex' must be exactly eight hex digits and be separated by spaces or new lines from the next. Multiple lines of 'hex' words can be used with the command terminated by a matching END DEFINEFONT.

DIM [type] decl [,decl]...
where 'decl' is:
var [length] [type] [init]
'var' is a variable name with optional dimensions
'length' is used to set the maximum size of the string to 'n' as in LENGTH n
'type' is one of FLOAT or INTEGER or STRING (the type can be prefixed by the keyword AS - as in AS FLOAT)
'init' is the value to initialise the variable and consists of:
= <expression>
For a simple variable one expression is used, for an array a list of comma separated expressions surrounded by brackets is used.
 	
Examples:
DIM nbr(50)
DIM INTEGER nbr(50)
DIM name AS STRING
DIM a, b$, nbr(100), strn$(20)
DIM a(5,5,5), b(1000)
DIM strn$(200) LENGTH 20
DIM STRING
	Declares one or more variables (ie, makes the variable name and its characteristics known to the interpreter).
	When OPTION EXPLICIT is used (as recommended) the DIM, LOCAL or STATIC commands are the only way that a variable can be created. If this option is not used then using the DIM command is optional and if not used the variable will be created automatically when first referenced.
	The type of the variable (ie, string, float or integer) can be specified in one of three ways:
	By using a type suffix (ie, !, % or $ for float, integer or string). For example:
	`DIM nbr%, amount!, name$`
	By using one of the keywords FLOAT, INTEGER or STRING immediately after the command DIM and before the variable(s) are listed. The specified type then applies to all variables listed (ie, it does not have to be repeated). For example:
	`DIM STRING first_name, last_name, city`
	By using the Microsoft convention of using the keyword "AS" and the type keyword (ie, FLOAT, INTEGER or STRING) after each variable. If you use this method the type must be specified for each variable and can be changed from variable to variable. For example:
	`DIM amount AS FLOAT, name AS STRING`
	Floating point or integer variables will be set to zero when created and strings will be set to an empty string (ie, ""). You can initialise the value of the variable with something different by using an equals symbol (=) and an expression following the variable definition. For example:
	`DIM STRING city = "Perth", house = "Brick"`
	The initialising value can be an expression (including other variables) and will be evaluated when the DIM command is executed. See the chapter "Defining and Using Variables" for more examples of the syntax.
	As well as declaring simple variables the DIM command will also declare arrayed variables (ie, an indexed variable with up to five dimensions). Note that this is different from the original Colour Maximite and Micromite versions of MMBasic which supported up to eight dimensions.
	Following the variable's name the dimensions are specified by a list of

strn(200) LENGTH 20
DIM a = 1234, b = 345
DIM STRING strn = "text"
DIM x%(3) = (11, 22, 33, 44)
	numbers separated by commas and enclosed in brackets. For example:
	`DIM array(10, 20)`
	Each number specifies the number of elements in each dimension. Normally the numbering of each dimension starts at 0 but the OPTION BASE command can be used to change this to 1.
	The above example specifies a two dimensional array with 11 elements (0 to 10) in the first dimension and 21 (0 to 20) in the second dimension. The total number of elements is 231 and because each floating point number on the Colour Maximite 2 requires 8 bytes a total of 1848 bytes of memory will be allocated.
	Strings will default to allocating 255 bytes (ie, characters) of memory for each element and this can quickly use up memory when defining arrays of strings. In that case the LENGTH keyword can be used to specify the amount of memory to be allocated to each element and therefore the maximum length of the string that can be stored. This allocation ('n') can be from 1 to 255 characters.
	For example: `DIM STRING s(5, 10)` will declare a string array with 66 elements consuming 16,896 bytes of memory while:
	`DIM STRING s(5, 10) LENGTH 20`
	Will only consume 1,386 bytes of memory. Note that the amount of memory allocated for each element is n + 1 as the extra byte is used to track the actual length of the string stored in each element.
	If a string longer than 'n' is assigned to an element of the array an error will be produced. Other than this, string arrays created with the LENGTH keyword act exactly the same as other string arrays. This keyword can also be used with non array string variables but it will not save any memory unless the length is less than 16 when it will both save memory and improve performance.
	In the above example you can also use the Microsoft syntax of specifying the type after the length qualifier. For example:
	`DIM s(5, 10) LENGTH 20 AS STRING`
	Arrays can also be initialised when they are declared by adding an equals symbol (=) followed by a bracketed list of values at the end of the declaration. For example:
	`DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)`
	or `DIM s$(3) = ("foo", "boo", "doo", "zoo")`
	Note that the number of initialising values must match the number of elements in the array including the base value set by OPTION BASE. If a multi dimensioned array is initialised then the first dimension will be initialised first followed by the second, etc.
	Also note that the initialising values must be after the LENGTH qualifier (if used) and after the type declaration (if used).

DO <statements> LOOP	This structure will loop forever; the EXIT DO command can be used to terminate the loop or control must be explicitly transferred outside of the loop by commands like GOTO or EXIT SUB (if in a subroutine).
DO WHILE expression <statements> LOOP	Loops while "expression" is true (this is equivalent to the older WHILE-WEND loop, also implemented in MMBasic). If, at the start, the expression is false the statements in the loop will not be executed, not even once.
DO <statements> LOOP UNTIL expression	Loops until the expression following UNTIL is true. Because the test is made at the end of the loop the statements inside the loop will be executed at least once, even if the expression is true.
DRAW3D	V5.07.00 and later includes a 3D engine. See the document “The CMM2 3D engine” for full details.

EDIT	Invoke the full screen editor. See the section *Full Screen Editor* for details of how to use the editor.
ELSE	Introduces a default condition in a multiline IF statement. See the multiline IF statement for more details.
ELSEIF expression THEN or ELSE IF expression THEN	Introduces a secondary condition in a multiline IF statement. See the multiline IF statement for more details.
END	End the running program and return to the command prompt.
END CSUB	Marks the end of a C subroutine. See the CSUB command. Each CSUB must have one and only one matching END CSUB statement.
END FUNCTION	Marks the end of a user defined function. See the FUNCTION command. Each function must have one and only one matching END FUNCTION statement. Use EXIT FUNCTION if you need to return from a function from within its body.

ENDIF
or
END IF
	Terminates a multiline IF statement.
	See the multiline IF statement for more details.

END SUB	Marks the end of a user defined subroutine. See the SUB command. Each sub must have one and only one matching END SUB statement. Use EXIT SUB if you need to return from a subroutine from within its body.
ERASE variable [,variable]...	Deletes variables and frees up the memory allocated to them. This will work with arrayed variables and normal (non array) variables. Arrays can be specified using empty brackets (eg, dat()) or just by specifying the variable's name (eg, dat). Use CLEAR to delete all variables at the same time (including arrays).
ERROR [error_msg$]	Forces an error and terminates the program. This is normally used in debugging or to trap events that should not occur.
EXECUTE command$	This executes the Basic command "command$". Use should be limited to basic commands that execute sequentially for example the GOTO statement will not work properly Things that are tested and work OK include GOSUB, Subroutine calls, other simple statements (like PRINT and simple assignments) Multiple statements separated by : are not allowed and will error The command sets an internal watchdog before executing the requested command and if control does not return to the command, like in a GOTO statement, the timer will expire. In this case you will get the message "Command timeout". RUN is a special case and will cancel the timer allowing you to use the command to chain programs if required.
EXIT DO EXIT FOR EXIT FUNCTION	EXIT DO provides an early exit from a DO...LOOP EXIT FOR provides an early exit from a FOR...NEXT loop. EXIT FUNCTION provides an early exit from a defined function.

EXIT SUB	EXIT SUB provides an early exit from a defined subroutine. The old standard of EXIT on its own (exit a do loop) is also supported.
FILES	Invoke the File Manager. To simply list the files on the SD card (as in the original Colour Maximite) use the command LIST FILES or LS. See the section *File Manager* for details of how to use the file manager.
FONT [#]font-number, scaling	This will set the default font for displaying text on the VGA screen. Fonts are specified as a number. For example, #2 (the # is optional) See the chapter "Basic Graphics" for details of the available fonts. 'scaling' can range from 1 to 15 and will multiply the size of the pixels making the displayed character correspondingly wider and higher. Eg, a scale of 2 will double the height and width.
FOR counter = start TO finish [STEP increment]	Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' is greater than 'finish'. The 'increment' can be an integer or floating point number. Note that using a floating point fractional number for 'increment' can accumulate rounding errors in 'counter' which could cause the loop to terminate early or late. 'increment' can be negative in which case 'finish' should be less than 'start' and the loop will count downwards. See also the NEXT command.
FRAMEBUFFER	This command allows you to create, use and remove a variable size framebuffer which should make many applications which have a working area bigger than the screen easier to program. While using a framebuffer setting a different graphics mode which changes the colour depth will cause an error. JPG files cannot be loaded to the framebuffer and will error if tried. The framebuffer is deleted by Ctrl-C and by running a new program.
FRAMEBUFFER CREATE HorizontalSize%, VerticalSize%	This command creates a framebuffer with the width and height specified in pixels. HorizontalSize>=MM.HRES and <=1600: VerticalSize>=MM.VRES and <=1200
FRAMEBUFFER WRITE	This command sets all drawing commands to write to the framebuffer and inherit the width and height defined.
FRAMEBUFFER BACKUP	This command creates a backup copy of the framebuffer. If a backup already exists it is overwritten. This allows the programmer to save the background before he/she starts writing non-static data to it. NB: It won't be possible to use this command if a very large framebuffer is specified in 12 or 16-bit colour depth. A sensible error will be given in this case.
FRAMEBUFFER RESTORE [x, y, w, h]	This command restores all or part of the framebuffer from the backup. This allows the programmer to "clean" all or part of the framebuffer before adding new non-static items
FRAMEBUFFER WINDOW x, y, page [,I or B]	This command copies an area MM.HRES by MM.VRES from the framebuffer with top left at x,y to the page specified, The optional parameter specifies if the copy is Immediate or during frame Blanking
FRAMEBUFFER CLOSE	This command releases the memory resources used by the framebuffer and backup allowing a new framebuffer to be created with a different size

FUNCTION xxx (arg1 [,arg2, …]) [AS <type>}
  <statements>
  <statements>
  xxx = <return value>
END FUNCTION
	Defines a callable function. This is the same as adding a new function to MMBasic while it is running your program.
	'xxx' is the function name and it must meet the specifications for naming a variable. The type of the function can be specified by using a type suffix (ie, xxx$) or by specifying the type using AS <type> at the end of the functions definition. For example:
	`FUNCTION xxx (arg1, arg2) AS STRING`
	'arg1', 'arg2', etc are the arguments or parameters to the function (the brackets are always required, even if there are no arguments). An array is specified by using empty brackets. ie, arg3(). The type of the argument can be specified by using a type suffix (ie, arg1$) or by specifying the type using `AS <type> (ie, arg1 AS STRING).`
	The argument can also be another defined function or the same function if recursion is to be used (the recursion stack is limited to 50 nested calls).
	To set the return value of the function you assign the value to the function's name. For example:
	`FUNCTION SQUARE(a)`
	`  SQUARE = a * a`
	`END FUNCTION`
	Every definition must have one END FUNCTION statement. When this is reached the function will return its value to the expression from which it was called. The command EXIT FUNCTION can be used for an early exit.
	You use the function by using its name and arguments in a program just as you would a normal MMBasic function. For example:
	`PRINT SQUARE(56.8)`
	When the function is called each argument in the caller is matched to the argument in the function definition. These arguments are available only inside the function.
	Functions can be called with a variable number of arguments. Any omitted arguments in the function's list will be set to zero or a null string.
	Arguments in the caller's list that are a variable and have the correct type will be passed by reference to the function. This means that any changes to the corresponding argument in the function will also be copied to the caller's variable and therefore may be accessed after the function has ended. Arrays are passed by specifying the array name with empty brackets (eg, arg()) and are always passed by reference and must be the correct type.
	You must not jump into or out of a function using commands like GOTO, GOSUB, etc. Doing so will have undefined side effects including the possibility of ruining your day.

GUI	This is a full implementation of the GUI controls as popularised on the Micromite Plus. This is a suite of advanced graphic controls that respond to input from a mouse. These include on screen switches, buttons, indicator lights, keyboard, etc. The GUI controls have their own manual: *GUI Controls and Programming.pdf* Note: • The mouse must be connected and working for the GUI commands to work and has been set up using the OPTION MOUSE command • The number of controls allowed is defined using the OPTION MAXCTRLS n command where "n" is the maximum number that can be used to identify a control. By default this is set to zero so it must be configured before the GUI controls can be used (recommended is 100). Differences compared to the Micromite Plus implementation: • The Colour Maximite 2 uses the mouse as the user interface rather than

	touch. • GUI BEEP not implemented. Use PLAY TONE or PULSE in the click interrupt routine if required • The TOUCH function is renamed CLICK.
GOTO target	Branches program execution to the target, which can be a line number or a label.
GUI CURSOR	The GUI CURSOR command provides a mechanism for displaying and manipulating a cursor on the screen. The cursor sits above all other graphics and nothing can overwrite it. BLIT, page copy, text, sprite, box etc. can all be used and the cursor will stay in view unless deliberately hidden. The cursor is always on PAGE 0 for colours 8 and 16 and page 1 for 12-bit colour and it can be moved even if the write page is somewhere else.
GUI CURSOR ON [cursorno [, x, y [,cursorcolour]]]	Cursor no can be 0 (Default: mouse type pointer) or 1 (cross), in addition the user can load his own cursor using a SPRITE look-alike file in which case this is cursor no. 2 For cursor numbers 0 and 1 the programmer can override the default white cursor by specifying the colour in the open command.
GUI CURSOR x, y	Moves the cursor to x, y. Does not display the cursor if hidden but just updates the location.
GUI CURSOR OFF	Turns off the cursor
GUI CURSOR HIDE	Hides the cursor but maintains its position
GUI CURSOR SHOW	Shows a hidden cursor in its stored position
GUI CURSOR COLOUR cursorcolour	Changes the colour of cursor number 0 or 1. Does not impact loaded cursors where its colours are specified by the cursor designer
GUI CURSOR LOAD "fname"	Loads a user cursor from a file in the Maximite sprite format with a minor change. The header is now Width, Height, Xoffset, Yoffset. The two offsets determine where on the cursor the pointer is defined to be. So the mouse cursor has offsets 0,0 and the cross has offsets 7,7
GUI CURSOR LINK MOUSE GUI CURSOUR UNLINK MOUSE	These commands link and unlink mouse activity to the cursor. When linked the cursor will automatically track the mouse activity. The mouse must be opened using the CONTROLLER MOUSE OPEN command before the cursor is linked.
GUI BITMAP x, y, bits [, width] [, height] [, scale] [, c] [, bc]	Displays the bits in a bitmap on the screen starting at 'x' and 'y' 'height' and 'width' are the dimensions of the bitmap as displayed on the screen and default to 8x8. 'scale' is optional and defaults to that set by the FONT command. 'c' is the drawing colour and 'bc' is the background colour. They are optional and default to the current foreground and background colours. The bitmap can be an integer or a string variable or constant and is drawn using the first byte as the first bits of the top line (bit 7 first, then bit 6, etc) followed by the next byte, etc. When the top line has been filled the next line of the displayed bitmap will start with the next bit in the integer or string. See the chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.

HELP [text]	This displays up to 3 lines of help that match the "text". text isn't quoted. If text isn't specified you can type characters into the help command and the display will show up to three lines than match. Use the delete key to remove characters/ You can use up arrow and down arrow to move through matches (e.g. for things like MM.INFO) Exit the help command with either ESC or F12 The intention of the facility is just to provide a quick check on command syntax and will never replace this written manual. The command can be called from within the editor using F12. In this case the HELP command is called with any characters surrounding the cursor that can form part of a command or function name. So if the cursor is positioned on the 'I' of a DIM command the help will immediately show the syntax for DIM
I2C	The I2C commands will send and receive data over an I^2C channel. I2C (no suffix) refers to channel 1 while commands I2C2 and I2C3 refer to channels 2 and 3 using the same syntax. Also see *Appendix B*.
HUMID pin, tvar, hvar [,DHT11]	Returns the temperature and humidity using the DHT22 sensor. Alternative versions of the DHT22 are the AM2303 or the RHT03 (all are compatible). 'pin' is the I/O pin connected to the sensor. Any I/O pin may be used. 'tvar' is the variable that will hold the measured temperature and 'hvar' is the same for humidity. Both must be present and both must be floating point variables. For example: HUMID 3, TEMP!, HUMIDITY! Temperature is measured in ºC and the humidity is percent relative humidity. Both will be measured with a resolution of 0.1. If an error occurs (sensor not connected or corrupt signal) both values will be 1000.0. Normally the signal pin of the DHT22 should be pulled up by a 1K to 10K resistor (4.7K recommended) to the supply voltage. The optional DHT11 parameter modifies the timings to work with the DHT11. Set to 1 for DHT11 and 0 or omit for DHT22. NB: the firmware will also accept the older DHT22 command
I2C OPEN speed, timeout	Enables the I^2C module in master mode. 'speed' is the clock speed (in KHz) to use and must be one of 100, 400 or 1000. 'timeout' is a value in milliseconds after which the master send and receive commands will be interrupted if they have not completed. The minimum value is 100. A value of zero will disable the timeout (though this is not recommended).
I2C WRITE addr, option, sendlen, senddata [,sendata ....]	Send data to the I^2C slave device. 'addr' is the slave's I^2C address. 'option' can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 'sendlen' is the number of bytes to send. 'senddata' is the data to be sent - this can be specified in various ways (all values sent will be between 0 and 255). Notes: • The data can be supplied as individual bytes on the command line. Example: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25

	• The data can be in a one dimensional array specified with empty brackets (ie, no dimensions). 'sendlen' bytes of the array will be sent starting with the first element. Example: I2C WRITE &H6F, 0, 3, ARRAY() The data can be a string variable (not a constant). Example: I2C WRITE &H6F, 0, 3, STRING$
I2C READ addr, option, rcvlen, rcvbuf	Get data from the I^2C slave device. 'addr' is the slave's I^2C address. 'option' can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 'rcvlen' is the number of bytes to receive. 'rcvbuf' is the variable or array used to save the received data - this can be: • A string variable. Bytes will be stored as sequential characters. • A one dimensional array of numbers specified with empty brackets. Received bytes will be stored in sequential elements of the array starting with the first. Example: I2C READ &H6F, 0, 3, ARRAY() A normal numeric variable (in this case rcvlen must be 1).
I2C CLOSE	Disables the master I^2C module and returns the I/O pins to a "not configured" state. They can then be configured using SETPIN. This command will also send a stop if the bus is still held.
IF expr THEN stmt [: stmt] or IF expr THEN stmt ELSE stmt	Evaluates the expression 'expr' and performs the statement following the THEN keyword if it is true or skips to the next line if false. If there are more statements on the line (separated by colons (:) they will also be executed if true or skipped if false. The ELSE keyword is optional and if present only one true statement is allowed following the THEN keyword. If 'expr' is resolved to be false the single statement following the ELSE keyword will be executed. The 'THEN statement' construct can be also replaced with: GOTO linenumber | label'. This type of IF statement is all on one line.

IF expression THEN
<statements>
[ELSEIF expression THEN
<statements>]
[ELSE
<statements>]
ENDIF
	Multiline IF statement with optional ELSE and ELSEIF cases and ending with ENDIF. Each component is on a separate line.
	Evaluates 'expression' and performs the statement(s) following THEN if the expression is true or optionally the statement(s) following the ELSE statement if false. The ELSEIF statement (if present) is executed if the previous condition is false and it starts a new IF chain with further ELSE and/or ELSEIF statements as required.
	One ENDIF is used to terminate the multiline IF.

IMAGE RESIZE x, y, width, height, new_x, new_y, new_width, new_height [,page_number] IMAGE RESIZE_FAST x, y, width, height, new_x, new_y, new_width, new_height [,page_number] [,flag]	This takes the part of the image with a top corner at 'x', 'y' and of specified 'width' and 'height' and resizes it writing it back to the area specified by 'new_x', 'new_y', 'new_width', new_height. The command will both increase and decrease the size of the part of the image chosen. It uses a bi-linear interpolation to generate the new pixels. The 'page number' is the page that the image data is read from it is then written to the current write page as specified by the PAGE WRITE n command. If 'page_number' is omitted the data is read from the write page. Use the text FRAMEBUFFER as the page no. to read from the framebuffer. IMAGE RESIZE uses bi-linear interpolation to resize the image.

	IMAGE RESIZE_FAST uses a nearest neighbour technique and is much faster but the resulting image quality will not be as good. If flag is set to 1 then black pixels are not written in the resized image.
IMAGE ROTATE x, y, width, height, new_x, new_y, angle! [,page_number] IMAGE ROTATE_FAST x, y, width, height, new_x, new_y, angle! [,page_number] [,flag]	Takes the part of the image with a top corner at 'x', 'y' and of specified 'width' and 'height' and rotates it about its centre in a clockwise direction by 'angle' (in degrees). Areas of the image that after rotation are outside of the area specified are cropped. The image is then drawn with the top left corner specified by new_x and new_y The 'page number' is the page that the image data is read from and it is written to the current write page as specified by the PAGE WRITE n command. If 'page_number' is omitted the data is read from the write page. Use the text FRAMEBUFFER as the page no. to read from the framebuffer. IMAGE ROTATE uses bi-linear interpolation to resize the image. IMAGE ROTATE_FAST uses a nearest neighbour technique and is much faster but the resulting image quality will not be as good. If flag is set to 1 then black pixels are not written in the rotated image.
IMAGE WARP_H x, y, w, h, x1, y1, h1, x2, y2, h2 [,readpage] [,flag] IMAGE WARP_V x, y, w, h, x1, y1, w1, x2, y2, w2 [,readpage] [,flag]	These commands allow you to modify an image by translating and/or stretching or compressing one of the axis. x, y, w, h define the top left coordinates and the width and height of the image to read from the current write page or optional read page In both cases x1 and x1 define the top left corner of the area to write In both cases x2 and y2 define the top right corner of the area to write When warping horizontally h1 and h2 define the height of the transformed area at the left edge and right edge When warping vertically w1 and w2 define the width of the transformed area at the top edge and bottom edge. If flag is set to 1 then black pixels are not written in the warped image.
INC var [,increment]	Increments the variable “var” by either 1 or, if specified, the value in increment. “increment” can have a negative. This is functionally the same as var = var + increment but is processed much faster
INPUT ["prompt$";] var1 [,var2 [, var3 [, etc]]]	Will take a list of values separated by commas (,) entered at the console and will assign them to a sequential list of variables. For example, if the command is: INPUT a, b, c And the following is typed on the keyboard: 23, 87, 66 Then a = 23 and b = 87 and c = 66 The list of variables can be a mix of float, integer or string variables. The values entered at the console must correspond to the type of variable. If a single value is entered a comma is not required (however that value cannot contain a comma). ‘prompt$’ is a string constant (not a variable or expression) and if specified it will be printed first. Normally the prompt is terminated with a semicolon (;) and in that case a question mark will be printed following the prompt. If the prompt is terminated with a comma (,) rather than the semicolon (;) the question mark will be suppressed.

INPUT #nbr, list of variables	Same as the normal INPUT command except that the input is read from a file previously opened for INPUT as '#fnbr' or a serial port previously opened for INPUT as 'nbr'. See the OPEN command. #0 can be used which refers to the console.
IR dev, key , int or IR CLOSE	Decodes NEC or Sony infrared remote control signals. An IR Receiver Module is used to sense the IR light and demodulate the signal. It should be connected to the IR pin (see the pinout tables). This command will automatically set that pin to an input. The IR signal decode is done in the background and the program will continue after this command without interruption. 'dev' and 'key' should be numeric variables and their values will be updated whenever a new signal is received ('dev' is the device code transmitted by the remote and 'key' is the key pressed). 'int' is a user defined subroutine that will be called when a new key press is received or when the existing key is held down for auto repeat. In the interrupt subroutine the program can examine the variables 'dev' and 'key' and take appropriate action. The IR CLOSE command will terminate the IR decoder and return the I/O pin to a not configured state. Note that for the NEC protocol the bits in 'dev' and 'key' are reversed. For example, in 'key' bit 0 should be bit 7, bit 1 should be bit 6, etc. This does not affect normal use but if you are looking for a specific numerical code provided by a manufacturer you should reverse the bits. This describes how to do it: http://www.thebackshed.com/forum/forum_posts.asp?TID=8367 See the chapter "Special Hardware Devices" for more details.
IR SEND pin, dev, key	Generate a 12-bit Sony Remote Control protocol infrared signal. 'pin' is the I/O pin to use. This can be any I/O pin which will be automatically configured as an output and should be connected to an infrared LED. Idle is low with high levels indicating when the LED should be turned on. 'dev' is the device being controlled and is a number from 0 to 31, 'key' is the simulated key press and is a number from 0 to 127. The IR signal is modulated at about 38KHz and sending the signal takes about 25mS.
KILL file$	Deletes the file or empty directory specified by 'file$'. If there is an extension it must be specified.
LET variable = expression	Assigns the value of 'expression' to the variable. LET is automatically assumed if a statement does not start with a command. For example: `Var = 56`
LINE x1, y1, x2, y2 [, LW [, C]]	Draws a line starting at the coordinates 'x1' and 'y1' and ending at 'x2' and 'y2'. 'LW' is the line's width and is only valid for horizontal or vertical lines. It defaults to 1 if not specified or if the line is a diagonal. 'C' is an integer representing the colour and defaults to the current foreground colour. All parameters can now be expressed as arrays and the software will plot the number of lines 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.

LINE INPUT [prompt$,] string-variable$	Reads an entire line from the console input into ‘string-variable$’. ‘prompt$’ is a string constant (not a variable or expression) and if specified it will be printed first. Unlike INPUT, this command will read a whole line, not stopping for comma delimited data items. A question mark is not printed unless it is part of ‘prompt$’.
LINE INPUT #nbr, string-variable$	Same as the LINE INPUT command except that the input is read from a file previously opened for INPUT as ‘#fnbr’ or a serial communications port previously opened for INPUT as ‘nbr’. See the OPEN command. #0 can be used which refers to the console. The # character is required.
LIST [file$] or LIST ALL [file$]	List a program on the serial console. LIST on its own will list the program with a pause at every screen full. LIST ALL will list the program without pauses. This is useful if you wish to transfer the program in the Maximite to a terminal emulator on a PC that has the ability to capture its input stream to a file. In most cases the filename 'file$' is required however if EDIT file$ or RUN file$ has been used previously the "current program name" will have been set and in that case LIST will default to using that filename.
LIST FILES [fspec$] [, sort]	Lists files in the current directory on the SD card. 'fspec$' (if specified) can contain search wildcards. Question marks (?) will match any character and an asterisk (*) will match any number of characters. If omitted, all files will be listed. For example: * Find all entries *.TXT Find all entries with an extension of TXT E*.* Find all entries starting with E X?X.* Find all three letter file names starting and ending with X ‘sort’ specifies the sort order as follows: size by ascending size time by ascending time/date name by file name (default if not specified) type by file extension
LIST COMMANDS or LIST FUNCTIONS	Lists all valid commands or functions
LIST PAGES	Lists the start address, width, height, and size of all the video pages for the current mode. In addition it shows whether for specific modes lines are duplicated in order to support the video output format.
LIST PROFILE LIST PROFILE CSV	Outputs information stored when OPTION PROFILING is ON. There are 5 comma-separated fields in the output The first field is the number of times a given statement has been run The second field is the average time in Microseconds taken to execute the statement The third field is the crunched statement as stored in memory. This is quoted to make it easier to load into excel etc. The fourth field is the filename in which the statement appears. This is blank if the statement is in the main program, otherwise it gives the filename of the

	include file in which the statement is to be found The fifth and final field gives the line number of the statement in the source file. When LIST PROFILE CSV is used a file is created in the same directory as the main program with the .BAS extension replaced with .CSV LIST PROFILE can only be used at the command line LIST PROFILE CSV can be used in a program allowing you to create and rename multiple files while a program is running if required
LOAD DATA fname$, address	Loads the binary contents of file "fname$" and stores it in CMM2 memory starting at "address". See also SAVE DATA
LOAD FONT file$	Load the font contained in 'file$' on the SD card and install it as font #8. See the section *Basic Graphics* earlier in this manual. You can convert font files designed for the original Colour Maximite using FontTweak from: https://www.c-com.com.au/MMedit.htm
LOAD BMP file$ [, x, y] or LOAD GIF [file$ [, x, y]] or LOAD JPG file$ [, x, y] or LOAD PNG file$ [, x, y] [, transparency_cut_off]	Load an image from the SD card and display it on the VGA monitor. "file$' is the name of the file and 'x' and 'y' are the screen coordinates for the top left hand corner of the image. If the coordinates are not specified the image will be drawn at the top left hand position on the screen. If an extension is not specified the appropriate extension will be added to the file name. All types of the BMP format are supported including black and white and true colour 24-bit images. The image can be of any size and pixels off the screen will be ignored. GIFs can be a single image or animated. If it is animated it will start playing in the background (ie, program execution will continue while it is playing). If an animated GIF is already running it will be replaced by the new one. If LOAD GIF is used without any parameters it will stop the currently playing animated GIF. JPG images cannot use progressive encoding and are limited to being completely within the screen resolution (ie, pixels cannot extend beyond the screen limits). MODE 2,16 is the optimum for displaying JPG images as the hardware decoder can write RGB565 pixels directly into the frame buffer. For all other modes, the firmware has to adjust the image by duplicating lines (mode 3) and/or converting from RGB565 to RGB332. PNG files must be in the RGB888 or ARGB8888 format and can be sized up to the current resolution of the screen. If the x & y start coordinates are specified pixels off the screen will be ignored. If the transparency level is specified and none-zero then: • If a PNG file is in ARGB8888 format the 'transparency_cut_off' parameter is used to determine whether the pixel should be solid or missing/transparent. Valid values are 1 to 15, no default. MMBasic compares the 4 most significant bits of the transparency data in the file with the cut off value and assigns a transparency of 0 or 15 depending on the comparison. This allows RGB(0,0,0) to be a valid solid colour. • If the file is in RGB888 format then an RGB level of 0,0,0 is used to determine transparency as there is no other information to use. If the 'transparency_cut_off' level is not specified all pixels will be loaded as solid colours as with any other image load.

LOCAL variable [, variables] See DIM for the full syntax.	Defines a list of variable names as local to the subroutine or function. This command uses exactly the same syntax as DIM and will create variables that will only be visible within the subroutine or function. They will be automatically discarded when the subroutine or function exits.
LONGSTRING	The LONGSTRING commands allow for the manipulation of strings longer than the normal MMBasic limit of 255 characters. Variables for holding long strings must be defined as single dimensioned integer arrays with the number of elements set to the number of characters required for the maximum string length divided by eight. The reason for dividing by eight is that each integer in an MMBasic array occupies eight bytes. Note that the long string routines do not check for overflow in the length of the strings. If an attempt is made to create a string longer than a long string variable's size the outcome will be undefined.
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 PRINT [#n,] src%()	Prints the longstring stored in ‘src%()’ to the file or COM port opened as ‘#n’. If ‘#n’ is not specified the output will be sent to the console.
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 RESIZE array%(), nbr	Sets the size of the longstring to nbr. This overrides the size set by other longstring commands so should be used with caution. Typical use would be in using a longstring as a byte array.
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 SETBYTE array%(), nbr, data	sets byte nbr to the value “data”, nbr respects OPTION BASE
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.
LOOP [UNTIL expression]	Terminates a program loop: see DO.
LS	An alias for the LIST FILES command.
MAP	The MAP commands allow the programmer to set the colours used in 8-bit colour modes. Each value in the 8-bit colour pallet can be set to an independent 24-bit colour.
MAP(n) = rgb%	This will assign the 24-bit colour 'rgb% to all pixels with the 8-bit colour value of 'n'. The change is activated after the MAP SET command
MAP MAXIMITE	This will set the colour map to the colours implemented in the original Colour Maximite.
MAP SET	This will cause MMBasic to update the colour map (set using MAP(n)=rgb%) during the next frame blanking interval.
MAP RESET	This will reset the colour map to the default colours. This map is used to assign 24-bit colours to individual values in the 8-bit colour space.
MATH	The math command performs many simple mathematical calculations that can be programmed in BASIC but there are speed advantages to coding looping structures in the firmware and there is the advantage that once debugged they are there for everyone without re-inventing the wheel. Note: 2 dimensional maths matrices are always specified DIM matrix(n_columns, n_rows) and of course the dimensions respect OPTION BASE. Quaternions are stored as a 5 element array w, x, y, z, magnitude.
Simple array arithmetic	
MATH SET nbr, array()	Sets all elements in array() to the value nbr. Note this is the fastest way of clearing an array by setting it to zero.
MATH SCALE in(), scale ,out()	This scales the matrix in() by the scalar scale and puts the answer in out(). Works for arrays of any dimensionality of both integer and float and can convert between. Setting b to 1 is optimised and is the fastest way of copying an entire array.
MATH ADD in(), num ,out()	This adds the value 'num' to every element of the matrix in() and puts the answer in out(). Works for arrays of any dimensionality of both integer and float and can convert between. Setting num to 0 is optimised and is a fast way of copying an entire array. in() and out() can be the same array.
MATH INTERPOLATE in1(),	This command This implements the following equation on every array

in2(), ratio, out()	element: out = (in2 - in1) * ratio + in1 Arrays can have any number of dimensions and must be distinct and have the same number of total elements. The command works with bot integer and floating point arrays in any mixture
MATH SLICE sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5] , destinationarray()	This command copies a specified set of values from a multi-dimensional array into a single dimensional array. It is much faster than using a FOR loop. The slice is specified by giving a value for all but one of the source array indicies and there should be as many indicies in the command, including the blank one, as there are dimensions in the source array e.g. OPTION BASE 1 DIM a(3,4,5) DIM b(4) MATH SLICE a(), 2, , 3, b() Will copy the elements 2,1,3 and 2,2,3 and 2,3,3 and 2,4,3 into array b()
MATH INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5] , sourcearray()	This is the opposite of MATH SLICE, has a very similar syntax, and allows you, for example, to substitute a single vector into an array of vectors with a single instruction e.g. OPTION BASE 1 DIM targetarray(3,4,5) DIM sourcearray(4)=(1,2,3,4) MATH INSERT targetarray(), 2, , 3, sourcearray() Will set elements 2,1,3 = 1 and 2,2,3 = 2 and 2,3,3 = 3 and 2,4,3 = 4
Matrix arithmetic	
MATH M_INVERSE array!(), inversearray!()	This returns the inverse of array!() in inversearray!(). The array must be square and you will get an error if the array cannot be inverted (determinant=0). array!() and inversearray!() cannot be the same.
MATH M_PRINT array()	Quick mechanism to print a 2D matrix one row per line.
MATH M_TRANSPOSE in(), out()	Transpose matrix in() and put the answer in matrix out(), both arrays must be 2D but need not be square. If not square then the arrays must be dimensioned in(m,n) out(n,m)
MATH M_MULT in1(), in2(), out()	Multiply the arrays in1() and in2() and put the answer in out()c. All arrays must be 2D but need not be square. If not square then the arrays must be dimensioned in1(m,n) in2(p,m) ,out(p,n)
Vector arithmetic	
MATH V_PRINT array()	Quick mechanism to print a small array on a single line
MATH V_NORMALISE inV(), outV()	Converts a vector inV() to unit scale and puts the answer in outV() (sqr(x*x + y*y +.....)=1 There is no limit on number of elements in the vector
MATH V_MULT matrix(), inV(), outV()	Multiplies matrix() and vector inV() returning vector outV(). The vectors and the 2D matrix can be any size but must have the same cardinality.

MATH V_CROSS inV1(), inV2(), outV()	Calculates the cross product of two three element vectors inV1() and inV2() and puts the answer in outV()
Quaternion arithmetic	
MATH Q_INVERT inQ(), outQ()	Invert the quaternion in inQ() and put the answer in outQ()
MATH Q_VECTOR x, y, z, outVQ()	Converts a vector specified by x , y, and z to a normalised quaternion vector outVQ() with the original magnitude stored
MATH Q_CREATE theta, x, y, z, outRQ()	Generates a normalised rotation quaternion outRQ() to rotate quaternion vectors around axis x,y,z by an angle of theta. Theta is specified in radians but respects the setting of OPTION ANGLE
MATH Q_EULER yaw, pitch, roll, outRQ()	Generates a normalised rotation quaternion outRQ() to rotate quaternion vectors as defined by the yaw, pitch and roll angles With the vector in front of the "viewer" yaw is looking from the top of the ector and rotates clockwise, pitch rotates the top away from the camera and roll rotates around the z-axis clockwise. The yaw, pitch and roll angles default to radians but respect the setting of OPTION ANGLE
MATH Q_MULT inQ1(), inQ2(), outQ()	Multiplies two quaternions inQ1() and inQ2() and puts the answer in outQ()
MATH Q_ROTATE , RQ(), inVQ(), outVQ()	Rotates the source quaternion vector inVQ() by the rotate quaternion RQ() and puts the answer in outVQ()
MATH FFT signalarray!(), FFTarray!()	Performs a fast fourier transform of the data in "signalarray!". "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "FFTarray" must be floating point and have dimension 2*N where N is the same as the signal array (e.g. f(1,1023) assuming OPTION BASE is zero) The command will return the FFT as complex numbers with the real part in f(0,n) and the imaginary part in f(1,n)
MATH FFT INVERSE FFTarray!(), signalarray!()	Performs an inverse fast fourier transform of the data in "FFTarray!". "FFTarray" must be floating point and have dimension 2*N where N must be a power of 2 (e.g. f(1,1023) assuming OPTION BASE is zero) with the real part in f(0,n) and the imaginary part in f(1,n). "signalarray" must be floating point and the single dimension must be the same as the FFT array. The command will return the real part of the inverse transform in "signalarray".
MATH FFT MAGNITUDE signalarray!(),magnitudearray!()	Generates magnitudes for frequencies for the data in "signalarray!" "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "magnitudearray" must be floating point and the size must be the same as the signal array The command will return the magnitude of the signal at various frequencies according to the formula: frequency at array position N = N * sample_frequency / number_of_samples
MATH FFT PHASE	Generates phases for frequencies for the data in "signalarray!".

signalarray!(), phasearray!()	"signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "phasearray" must be floating point and the size must be the same as the signal array The command will return the phase angle of the signal at various frequencies according to the formula above.
MATH 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: https://www.thebackshed.com/forum/ViewTopic.php?TID=13459&PID=166962#166962

MEMORY	List the amount of memory currently in use. For example:
	`Flash:`
	`  39K ( 6%) Program (1450 lines)`
	`  527K (94%) Free`
	
	`RAM:`
	`  0K ( 0%) 0 Variables`
	`  1K ( 0%) General`
	`24799K (100%) Free`
	
	Notes:
	• General memory is used by serial I/O buffers, etc.
	• Memory usage is rounded to the nearest 1K byte.

MEMORY SET address, byte, numberofbytes MEMORY SET BYTE address, byte, numberofbytes MEMORY SET SHORT address, short, numberofshorts	This command will set a region of memory to a value. BYTE = One byte per memory address. SHORT = Two bytes per memory address. WORD = Four bytes per memory address. INTEGER = Four bytes per memory address. FLOAT = Four bytes per memory address.

MEMORY SET WORD address, word, numberofwords MEMORY SET INTEGER address, integervalue ,numberofintegers [,increment] MEMORY SET FLOAT address, floatingvalue ,numberofloats [,increment]	'increment' is optional and controls the increment of the 'address' pointer as the operation is executed. For example, if increment=3 then only every third element of the target is set. The default is 1.
MEMORY COPY sourceaddress, destinationaddres, numberofbytes MEMORY COPY INTEGER sourceaddress, destinationaddress, numberofintegers [,sourceincrement][,destinationincrement] MEMORY COPY FLOAT sourceaddress, destinationaddress, numberoffloats [,sourceincrement][,destinationincrement]	This command will copy one region of memory to another. COPY INTEGER and FLOAT will copy four bytes per operation. 'sourceincrement' is optional and controls the increment of the 'sourceaddress' pointer as the operation is executed. For example, if sourceincrement=3 then only every third element of the source will be copied. The default is 1. 'destinationincrement' is similar and operates on the 'destinationaddress' pointer.
MID$(str$, start [, num]) = str2$	The characters in 'str$', beginning at position 'start', are replaced by the characters in 'str2$'. The optional 'num' refers to the number of characters from 'str2' that are used in the replacement. If 'num' is omitted, all of 'str2' is used. Whether 'num' is omitted or included, the replacement of characters never goes beyond the original length of 'str$'.
MKDIR dir$	Make, or create, the directory 'dir$' on the SD card.
MMDEBUG MMDEBUG BREAK	By default the MMDEBUG command acts exactly like the PRINT command so you can use it liberally throughout your program during development and know that by simply setting #MMDEBUG OFF or removing any #MMDEBUG directives, your program will run normally without any impact on performance. The second form of the MMDEBUG command is MMDEBUG BREAK. in this case you get a command promptDEBUG> Here you can execute any normal MMBASIC commands as though you were at the normal command prompt allowing you to change any variables, print them, or execute any other user subroutines or built in commands. These should be commands that make sense at the command line (e.g. do not use GOTO). To exit this mode use the command CONTINUE
MODE r, bits [, bg [, int]]	Set the format for the VGA video output. 'r' is the screen resolution. It is a number from 1 to 13 as follows: 1 = 800 x 600 pixels 2 = 640 x 400 pixels

3 = 320 x 200 pixels
4 = 480 x 432 pixels
5 = 240 x 216 pixels
6 = 256 x 240 pixels
7 = 320 x 240 pixels
8 = 640 x 480 pixels
9 = 1024 x 768 pixels (12-bit mode not available)
10 = 848 x 480 pixels (widescreen format)
11 = 1280 x 720 pixels (widescreen format)
12 = 960 x 540 pixels (widescreen format, lines duplicated)
13 = 400 x 300 pixels
14 = 960 x 540 pixels (widescreen format, lines not duplicated – not supported by all monitors
15 = 1280x1024
16 = 1920x1080 (widescreen format, G2 only)
17 = 384 x 240 pixels

All resolutions except 10, 11, 12, 14, and 16 work perfectly with monitors that have an aspect ratio of 4:3 or widescreen monitors that can switch to that ratio (most widescreen monitors will do this automatically).

'bits' is the colour depth and can be 8, 12, 16 or 32 (G2 only) – see the table below.

'bg' is the background colour and can be used in the 12-bit mode. If pixels in layer 0 are not set to solid (transparency = 15) then the background will show through as determined by the transparency value of the pixel. This parameter is ignored in 8 and 16-bit modes.

'int' is a subroutine that will be called at the start of frame blanking.

The specifications of the colour depth ('bits') are:

	32-bit(G2)	**16-bit**	**12-bit**	**8-bit**
H/W Pixel Format	ARGB8888	RGB565	ARGB4444	RGB332
Nbr Bits/Pixel	32(4 bytes)	16 (2 bytes)	16 (2 bytes)	8 (1 byte)
Colours	4,294,967,296	65536	4096	256
Transparency	None	None	16-levels	None
Pages Used	1	1	2	1
Layers	1	1	2 + background	1

The pages available in the various modes are:

Mode	**32-bit(G2)**	**16-bit:G1/G2**	**12-bit:G1/G2**	**8-bit:G1/G2**
1	0 to 2	0 to 2/0 to 6	0 to 2/0 to 6	0 to 6/0 to 15
2	0 to 6	0 to 6/0 to 14	0 to 6/0 to 14	0 to 13/0 to 29
3	0 to 27	0 to 25/0 to 57	0 to 24/0 to 56	0 to 53/0 to 60
4	0 to 7	0 to 70 to 17	0 to 7/0 to 17	0 to 15/0 to 35
5	0 to 33	0 to 30/0 to 60	0 to 29/0 to 60	0 to 60
6	0 to 28	0 to 26/0-60	0 to 25/0 to 59	0 to 53/0 to 60

7	0 to 20	0 to 19/0 to 46	0 to 18/0 to 45	0 to 41/0 to 60
8	0 to 4	0 to 4/0 to 10	0 to 4/0 to 10	0 to 10/0 to 23
9	N/A	0 and 1/0 to 3	N/A	0 to 3/0 to 8
10	0 to 3	0 to 2/0 to 7	0 to 2/0 to 7	0 to 7/0 to 17
11	N/A	0/0 to 2	N/A	0 to 2/0 to 6
12	N/A	0/0 to 4	N/A	0 to 3/0 to 11
13	0 to 12	0 to 11/0 to 28	0 to 10/0 to 27	0 to 26/0 to 60
14	0 to 2	0 to 2/0 to 6	0 to 2/0 to 6	0 to 6/0 to 14
15	N/A	N/A	N/A	0 to 1/0 to 4
16	N/A	N/A	N/A	N/A, 0 to 2
17	0 to 16	0 to 15/0 to 38	0 to 14/0 to 37	0 to 34/0 to 60

The display always shows the contents of page 0 (32-bit, 16-bit and 8-bit) and pages 0 and 1 (12-bit) unless overridden with the PAGE DISPLAY command. Use PAGE WRITE and PAGE COPY to avoid artefacts of flashing and tearing. For 12-bit colour depth page 0 is the lower level and page 1 the upper so the stack is: background, page 0, page 1 with each one overwriting the previous in turn as defined by the transparency values of each individual pixel.

MM.INFO(MAX PAGES) and MM.INFO(PAGE ADDRESS n) are useful if you wish to PEEK or POKE the video memory. In all cases the memory is arranged as a two dimensional array x,y so, to get the address of a specific pixel on a specific page n.

Notes:

- With an 8-bit colour depth you would use:
  add% =MM.INFO(page address n)+ y* mm.hres +x
- For 12 or 16-bit colour depth you would use
  add% =MM.INFO(page address n)+ (y* mm.hres +x) * 2

For modes 3, 5, 6, 7, 12, and 13 each line in page 0 is duplicated to get square pixels so Y needs to be multiplied by 2 for PEEK and both lines y*2 and Y*2+1 need to be POKEd. e.g, for an 8-bit colour depth:

```
add1% = MM.INFO(page address n)+ (y * 2) * MM.HRES + x
add2% = MM.INFO(page address n)+ (y * 2 + 1) * MM.HRES + x
```

NB: this duplication will also apply to page 1 in 12-bit colour modes.

For the 12 and 16-bit modes you can use POKE SHORT and PEEK(SHORT) which are designed for this purpose.

The monitor will see the following resolutions:

Modes 1, 13	800 x 600 @ 60 Hz
Modes 2, 3, 4, 5, 6, 7, 8, 17	640 x 480 @ 75Hz
Mode 9	1024 x 768 @ 60Hz
Mode 10	848 x 480 @ 60 Hz
Mode 11	1280 x 720 @ 60 Hz
Mode 15	1280 x 1024 @ 60 Hz
Mode 14	1920 x 540 @ 60 Hz
Mode 12, 16	1920 x 1080 @ 60 Hz

NEW	Deletes the program in program memory, clears all variables including saved variables and resets the interpreter (ie, closes files, serial ports, etc).
NEXT [counter-variable] [, counter-variable], etc	NEXT comes at the end of a FOR-NEXT loop; see FOR. The 'counter-variable' specifies exactly which loop is being operated on. If no 'counter-variable' is specified the NEXT will default to the innermost loop. It is also possible to specify multiple variables as in: NEXT x, y, z
ON ERROR ABORT or ON ERROR IGNORE or ON ERROR SKIP [nn] or ON ERROR CLEAR	This controls the action taken if an error occurs while running a program and applies to all errors discovered by MMBasic including syntax errors, wrong data, missing hardware, SD Card access, etc. ON ERROR ABORT will cause MMBasic to display an error message, abort the program and return to the command prompt. This is the normal behaviour and is the default when a program starts running. ON ERROR IGNORE will cause any error to be ignored. ON ERROR SKIP will ignore an error in a number of commands (specified by the number 'nn') executed following this command. 'nn' is optional, the default if not specified is one. After the number of commands has completed (with an error or not) the behaviour of MMBasic will revert to ON ERROR ABORT. If an error occurs and is ignored/skipped the read only variable MM.ERRNO will be set to non zero and MM.ERRMSG$ will be set to the error message that would normally be generated. These are reset to zero and an empty string by ON ERROR CLEAR. They are also cleared when the program is run and when ON ERROR IGNORE and ON ERROR SKIP are used. ON ERROR IGNORE can make it very difficult to debug a program so it is strongly recommended that only ON ERROR SKIP be used.
ON KEY target or ON KEY ASCIIcode, target	The first version of the command sets an interrupt which will call 'target' user defined subroutine whenever there is one or more characters waiting in the serial console input buffer. Note that all characters waiting in the input buffer should be read in the interrupt subroutine otherwise another interrupt will be automatically generated as soon as the program returns from the interrupt. The second version allows you to associate an interrupt routine with a specific key press. This operates at a low level for both the USB keyboard and a serial console and if activated the key does not get put into the input buffer but merely triggers the interrupt. It uses a separate interrupt from the simple ON KEY command so can be used at the same time if required. In both variants, to disable the interrupt use numeric zero for the target, i.e.: ON KEY 0. or ON KEY ASCIIcode, 0
ONEWIRE RESET pin or ONEWIRE WRITE pin, flag, length, data [, data…] or ONEWIRE READ pin, flag, length, data [, data…]	Commands for communicating with 1-Wire devices. ONEWIRE RESET will reset the 1-Wire bus ONEWIRE WRITE will send a number of bytes ONEWIRE READ will read a number of bytes 'pin' is the I/O pin (located in the rear connector) to use. It can be any pin capable of digital I/O. 'flag' is a combination of the following options: 1 - Send reset before command 2 - Send reset after command 4 - Only send/recv a bit instead of a byte of data 8 - Invoke a strong pullup after the command (the pin will be set high and open drain disabled)

	'length' is the length of data to send or receive 'data' is the data to send or variable to receive. The number of data items must agree with the length parameter. See also *Appendix C*.
OPEN fname$ FOR mode AS [#]fnbr	Opens a file for reading or writing. 'fname' is the filename with an optional extension separated by a dot (.). Long file names with upper and lower case characters are supported. A directory path can be specified with the forward or backwards slash as a directory separator. The parent of the current directory can be specified by using a directory name of .. (two dots) and the current directory with . (a single dot). For example OPEN "../dir1/dir2/filename.txt" FOR INPUT AS #1 'mode' is INPUT, OUTPUT, APPEND or RANDOM. INPUT will open the file for reading and throw an error if the file does not exist. OUTPUT will open the file for writing and will automatically overwrite any existing file with the same name. APPEND will also open the file for writing but it will not overwrite an existing file; instead any writes will be appended to the end of the file. If there is no existing file the APPEND mode will act the same as the OUTPUT mode (i.e. the file is created then opened for writing). RANDOM will open the file for both read and write and will allow random access using the SEEK command. When opened the read/write pointer is positioned at the end of the file. 'fnbr' is the file number (1 to 10). The # is optional. Up to 10 files can be open simultaneously. The INPUT, LINE INPUT, PRINT, WRITE and CLOSE commands as well as the EOF() and INPUT$() functions all use 'fnbr' to identify the file being operated on. See also ON ERROR and MM.ERRNO for error handling.
OPEN comspec$ AS [#]fnbr	Will open a serial communications port for reading and writing. Two ports are available (COM1: and COM2:) and both can be open simultaneously. If OPTION CONSOLE SCREEN is used then the console serial port is available as COM3:. Using 'fnbr' the port can be written to and read from using any command or function that uses a file number. 'comspec$' is the communication specification and is a string (it can be a string variable) specifying the serial port to be opened and optional parameters. The default is 9600 baud, 8 data bits, no parity and one stop bit. It has the form `"COMn: baud, buf, int, int-trigger, (DEN or DEP), 7BIT, (ODD or EVEN), INV, OC, S2"` Where: • 'n' is the serial port number for either COM1:, COM2 or COM3:.:. • 'baud' is the baud rate. This can be any value between 1200 (the minimum) and 1000000 (1MHz). Default is 9600. • 'buf' is the receive buffer size in bytes (default size is 256). The transmit buffer is fixed at 256 bytes. • 'int' is a user defined subroutine which will be called when the serial port has received some data. The default is no interrupt. • 'int-trigger' sets the trigger condition for calling the interrupt subroutine. If it is a normal number the interrupt subroutine will be called when this number of characters has arrived in the receive queue. All parameters except the serial port name (COMn:) are optional. If any one

	parameter is left out then all the following parameters must also be left out and the defaults will be used. These options can be added to the end of 'comspec$' • 'INV' specifies that the transmit and receive polarity is inverted. • 'OC' will force the transmit pin (and DE on COM1:) to be open collector. The default is normal (0 to 3.3V) output. • 'S2' specifies that two stop bits will be sent following each character transmitted. • '7BIT' will specify that 7 bit transmit and receive is to be used. • 'ODD' will specify that an odd parity bit will be appended (8 bits will be transmitted if 7BIT is specified, otherwise 9) • 'EVEN' will specify that an even parity bit will be appended (8 bits will be transmitted if 7BIT is specified, otherwise 9) • 'DEP' will enable RS485 mode with positive output on COM1-DE • 'DEN' will enable RS485 mode with negative output on COM1-DE
OPEN comspec$ AS GPS [,timezone_offset] [,monitor]	Will open a serial communications port for reading from a GPS receiver. See the GPS function for details. The sentences interpreted are GPRMC, GNRMC, GPCGA and GNCGA. The timezone_offset parameter is used to convert UTC as received from the GPS to the local timezone. If omitted the timezone will default to UTC. The timezone_offset can be a any number between -12 and 14 allowing the time to be set correctly even for the Chatham Islands in New Zealand (UTC +12:45). If the monitor parameter is set to 1 then all GPS input is directed to the console. This can be stopped by closing the GPS channel.
OPTION	See the section *Options* earlier in this manual.
PAGE COPY n TO m [,when] [,t]	Copy the contents of one video page to another. 'n' is the source and 'm' is the destination. 'when' can be one of letters I, B, or D. If omitted it will default to I. I means do the copy immediately. It is the most efficient but risks causing screen artefacts B means wait until the next frame blanking and then do the copy. It is the least efficient but is absolutely determinate in its effect and no screen artefacts will ever be seen. D means carry on processing the next command and do the copy in the background when the next frame blanking occurs. This is efficient but must be used with care as subsequent drawing commands may or may not be included in the copy depending on the timing of the next screen blanking. If the optional field ', t' is used then only non-black pixels are copied.
PAGE DISPLAY n [,page]	Allow the user the ability to select which page(s) are displayed on the screen (does not impact the current write page). Normal example of use is: PAGE DISPLAY 3 This will display the current contents of page 3. In 12 bit mode you can also use: PAGE DISPAY 4,1 This sets the page that is to be displayed as the top layer This command executes in the next screen blanking period. When a program ends for any reason the normal display page 0 is restored. Note for modes 3 or 5 or 6 or 7 or 12 or 13 you can only select pages 0 or 1 for display in 8 or

	16 bit colour and pages 0, 1, or 2 in 12-bit colour
PAGE DISPLAY -1	Turns off the display. To turn it back on select a valid display page. This command is only valid in a program.
PAGE RESIZE pageno, w, h	This command changes the width and height of a given page in memory. ‘w’ can be set between 1 and the normal width for the current display mode. ‘h’ can be set between 1 and the normal height of the display mode. When PAGE WRITE is set to a resized page all graphics commands will respect the new values of width and height, including print output scrolling, and MM.HRES and MM.VRES will report the revised size.
PAGE SCROLL pageno, x, y [,fillcolour]	Will scroll the image on the page specified by 'pageno' by the amount defined by 'x' pixels to the right and 'y' pixels up. By default the area scrolled off the screen appears on the other side. If 'fillcolour' is specified it will replace the area left behind by the scroll with the colour specified. If 'fillcolour' is set to -1 then the area left behind by the scroll is left untouched. This is the most efficient version and is suitable if there is a black background.
PAGE STITCH frompage1, from_page_2, topage, offset	Will take the last horizontal resolution minus 'offset' columns from 'frompage1' and the first 'offset' columns from 'frompage2' and copies them to 'topage'
PAGE AND_PIXELS sourcepage1, sourcepage2, destinationpage PAGE OR_PIXELS sourcepage1, sourcepage2, destinationpage PAGE XOR_PIXELS sourcepage1, sourcepage2, destinationpage	These commands combine the pixels on sourcepage1 and sourcepage2 by ANDing, ORing, or XORing them. destinationpage can be the same as either of the sourcepages if required.
PAGE WRITE n	Instructs MMBasic to make all graphics commands write to page 'n'. If not used page 0 is the default. Set the page to FRAMEBUFFER to write to the framebuffer – see the FRAMEBUFFER command.
PAUSE delay	Halt execution of the running program for ‘delay’ ms. This can be a fraction. For example, 0.2 is equal to 200 µs. The maximum delay is 2147483647 ms (about 24 days). Note that interrupts will be recognised and processed during a pause.
PIN (pin) = value	For a ‘pin’ configured as digital output this will set the output to low (‘value’ is zero) or high (‘value’ non-zero). You can set an output high or low before it is configured as an output and that setting will be the default output when the SETPIN command takes effect. See the function PIN() for reading from a pin and the command SETPIN for configuring it.
PIXEL x, y [,c]	Set a pixel on an attached VGA monitor to a colour. 'x' is the horizontal coordinate and 'y' is the vertical coordinate of the pixel. 'c' is a 24 bit number specifying the colour. 'c' is optional and if omitted the current foreground colour will be used. All parameters can be expressed as arrays and the software will plot the number of pixels 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 a single variable or constant. See the chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.
PIXEL FILL x, y, c	Implements a flood fill by reading the colour of the pixel at coordinates x,y and replacing it and the entire area of connected pixels having the same color with the new colour “c”
PLAY EFFECT file$ [,interrupt]	This will play the WAV file ' file$' at the same time as a MOD file is playing. If a previous EFFECT file is playing this command will immediately terminate it and commence playing the new file. The file is played in the background, 'interrupt' is optional and is the name of a subroutine that will be called when the file has finished playing. Note: wav files played using PLAY EFFECT during mod file playback must have the same sample rate as the modfile output. Files can be mono or stereo.
PLAY TONE left , right [, dur [, interrupt]]	Generates two separate sine waves on the sound output left and right channels. The tone plays in the background (the program will continue running after this command). 'left' and 'right' are the frequencies in Hz to use for the left and right channels. 'dur' specifies the number of milliseconds that the tone will sound for. MMBasic will round the time to the next nearest complete waveform of the first frequency specified so that the tone will always finish with the DC level in the middle and no discontinuity. If the duration is not specified the tone will continue until explicitly stopped or the program terminates. 'interrupt' is optional and is an interrupt subroutine to call when the tone has completed. The frequency can be from 1Hz to 20KHz and is very accurate (it is based on a crystal oscillator). The frequency can be changed at any time by issuing a new PLAY TONE command.
PLAY WAV file$ [, interrupt] or PLAY FLAC file$ [, interrupt] or PLAY MP3 file$ [, interrupt]	Play an audio file on the audio (DAC) output. 'file$' is the file to play (the appropriate extension will be appended if missing). The file is played in the background, 'interrupt' is optional and is the name of a subroutine that will be called when the file has finished playing. For WAV files MMBasic will automatically compensate for the frequency, number of bits and number of channels of the WAV file. For FLAC files the supported frequencies are: 44100Hz 16-bit(CD quality) and 24-bit 48000Hz 16-bit and 24-bit 88200Hz 16-bit and 24-bit 96000Hz 24-bit Maximums for FLAC and WAV file playback are 96KHz 24-bit. Both will auto-configure to the file provided. As an indication, 96KHz 24-bit FLAC uses just over 50% of the CPU's resources. If ‘file$’ is a directory then the firmware will list all the files of the relevant type in that directory and start playing them one-by-one. To play files in the current directory use an empty string (ie, ""). Each file listed will play in turn and the optional interrupt will fire when all files have been played. The filenames are stored with full path so you can use CHDIR while tracks are playing without causing problems.

	All files in the directory are listed if the command is executed at the command prompt but the listing is suppressed in a program
PLAY MODFILE file$ [,samplerate]	Will play a MOD file on the DAC outputs. 'file$' is the MOD file to play (the extension of .mod will be appended if missing). The MOD file is played in the background and will run continuously until PLAY STOP is called. The MOD encoder supports 32 channels, 16-bit resolution (but the DACs are only 12 bit), 32 samples, no fixed maximum size. The optional parameter samplerate specifies the number of samples per second generated by the modfile engine. The default is 44100. Processor overhead is reduced by decreasing this. Valid values are 8000, 16000, 22050, 44100, 48000 Note: wav files played using PLAY EFFECT during modfile playback must have the same sample rate as the modfile output.
PLAY MODSAMPLE sampleno, channelno [,volume] [,samplerate]	Plays one of the samples in the MOD file concurrently with the main MOD file playback. This allows sound effects to be incorporated in the MOD file. “sampleno” can be in the range 1 to 32. Up to 4 samples can be played simultaneously on independent channels using the specified “channelno” which must be in the range 1 to 4. The optional “volume” should be set in the range 0 to 64 (default 64). The optional “samplerate” specifies the update rate for the sample. The default is 16000. Changing this will change the pitch of the sample and the duration of playback and it should be set to the sample's original rate for playback as recorded.
PLAY SOUND soundno, channelno, type [,frequency] [,volume]	Play a series of sounds simultaneously on the audio output. 'soundno' is the sound number and can be from 1 to 4 allowing for four simultaneous sounds on each channel. 'channelno' specifies the output channel. It can be L (left speaker), R (right speaker) or B (both speakers) 'type' is the type of waveform. It can be S (sine wave), Q (square wave), T (triangle wave), W (rising sawtooth), N (noise), P (periodic noise) or O (turn off sound). Type N is true white noise. In this case the frequency parameter specifies the number of periods of 1/70000 seconds that the output stays at a particular random value. Type P is periodic white noise. In this case the frequency is some sort of relationship to the periodic frequency of the noise 'frequency' is the frequency from 1 to 20000 (Hz) and it must be specified except when type is O. 'volume' is optional and must be between 1 and 25. It defaults to 25 The first time PLAY SOUND is called all other audio usage will be blocked and will remain blocked until PLAY STOP is called. Output can be stopped temporarily using PLAY PAUSE and PLAY RESUME. Calling SOUND on an already running 'soundno' will immediately replace the previous output. Individual sounds are turned off using type “O” Running 4 sounds simultaneously on both channels of the audio output consumes about 23% of the CPU.
PLAY PAUSE PLAY RESUME PLAY STOP	PLAY PAUSE will temporarily halt the currently playing file or tone. PLAY RESUME will resume playing a sound that was paused. PLAY STOP will terminate the playing of the file or tone. When the program terminates for whatever reason the sound output will also be automatically stopped.
PLAY NEXT	When playing a sequence of audio tracks (by using PLAY MP3 on a

PLAY PREVIOUS	directory holding multiple MP3 files) these commands can be used to skip forward or back a file. The commands PLAY PAUSE, RESUME, VOLUME can also be used.
PLAY TTS [PHONETIC] "text" [,speed] [,pitch] [,mouth] [,throat] [, interrupt]	Outputs text as speech on the DAC outputs. See http://www.retrobits.net/atari/sam.shtml for details of parameter usage. The command is non-blocking and the speech is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the speech has finished playing.
PLAY VOLUME left, right	Will adjust the volume of the audio output. 'left' and 'right' are the levels to use for the left and right channels and can be between 0 and 100 with 100 being the maximum volume. There is a linear relationship between the specified level and the output. The volume defaults to maximum when a program is run.
POKE BYTE addr%, byte or POKE SHORT addr%, short% or POKE WORD addr%, word% or POKE INTEGER addr%, int% or POKE FLOAT addr%, float! or POKE VAR var, offset, byte or POKE VARTBL, offset, byte	Will set a byte or a word within the CPU's virtual memory space. POKE BYTE will set the byte (ie, 8 bits) at the memory location 'addr%' to 'byte'. 'addr%' should be an integer. POKE SHORT will set the short integer (ie, 16 bits) at the memory location 'addr%' to 'word%'. 'addr%' and short%' should be integers. POKE WORD will set the word (ie, 32 bits) at the memory location 'addr%' to 'word%'. 'addr%' and 'word%' should be integers. POKE INTEGER will set the MMBasic integer (ie, 64 bits) at the memory location 'addr%' to int%'. 'addr%' and int%' should be integers. POKE FLOAT will set the word (ie, 64 bits) at the memory location 'addr%' to 'float!'. 'addr%' should be an integer and 'float!' a floating point number. POKE VAR will set a byte in the memory address of 'var'. 'offset' is the ±offset from the address of the variable. An array is specified as var(). POKE VARTBL will set a byte in MMBasic's variable table. 'offset' is the ±offset from the start of the variable table. Note that a comma is required after the keyword VARTBL.
POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour] POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour]	Draws a filled or outline polygon with n xy-coordinate pairs in xarray%() and yarray%(). If 'fillcolour' is omitted then just the polygon outline is drawn. If 'bordercolour' is omitted then it will default to the current default foreground colour. If the last xy-coordinate pair is not the same as the first the firmware will automatically create an additional xy-coordinate pair to complete the polygon. The size of the arrays should be at least as big as the number of x,y coordinate pairs. 'n' can be an array and the colours can also optionally be arrays as follows: POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour] The elements of array n() define the number of xy-coordinate pairs in each of the polygons. e.g DIM n(1)=(3,3) would define that 2 polygons are to be drawn with three vertices each. The size of the n array determines the number of polygons that will be drawn unless an element is found with the value zero in which case the firmware only processes polygons up to that point. The x,y-coordinate pairs for all the polygons are stored in xarray%() and yarray%(). The xarray%() and yarray%() parameters must have at least as many elements as the total of the values in the n array. Each polygon can be closed with the first and last elements the same. If the last element is not the same as the first the firmware will automatically create an additional x,y-coordinate pair to complete the polygon. If fill

	colour is omitted then just the polygon outlines are drawn.
	The colour parameters can be a single value in which case all polygons are drawn in the same colour or they can be arrays with the same cardinality as n. In this case each polygon drawn can have a different colour of both border and/or fill. For example, this will draw 3 triangles in yellow, green and red:
	`DIM c%(2)=(3,3,3)`
	`DIM x%(8)=(100,50,150,100,50,150,100,50,150)`
	`DIM y%(8)=(50,100,100,150,200,200,250,300,300)`
	`DIM fc%(2)=(rgb(yellow),rgb(green),rgb(red))`
	`POLYGON c%(),x%(),y%(),fc%(),fc%()`

PORT(start, nbr [,start, nbr]…) = value	Set a number of I/O pins simultaneously (ie, with one command). 'start' is an I/O pin number and the lowest bit in 'value' (bit 0) will be used to set that pin. Bit 1 will be used to set the pin 'start' plus 1, bit 2 will set pin 'start'+2 and so on for 'nbr' number of bits. I/O pins used must be numbered consecutively and any I/O pin that is invalid or not configured as an output will cause an error. The start/nbr pair can be repeated if an additional group of output pins needed to be added. For example; PORT(15, 4, 23, 4) = &B10000011 Will set eight I/O pins. Pins 15 and 16 will be set high while 17, 18, 23, 24 and 25 will be set to a low and finally 26 will be set high. This command can be used to conveniently communicate with parallel devices like LCD displays. Any number of I/O pins (and therefore bits) can be used from 1 to the number of I/O pins on the chip. See the PORT function to simultaneously read from a number of pins.
PRINT expression [[,;]expression] … etc	Outputs text to the console (either the VGA screen or the serial or both if they are available). Multiple expressions can be used and must be separated by either a: • Comma (,) which will output the tab character • Semicolon (;) which will not output anything (it is just used to separate expressions). • Nothing or a space which will act the same as a semicolon. A semicolon (;) at the end of the expression list will suppress the automatic output of a carriage return/ newline at the end of a print statement. When printed, a number is preceded with a space if positive or a minus (-) if negative but is not followed by a space. Integers (whole numbers) are printed without a decimal point while fractions are printed with the decimal point and the significant decimal digits. Large floating point numbers (greater than six digits) are printed in scientific number format. The function TAB() can be used to space to a certain column and the string functions can be used to justify or otherwise format strings.
PRINT #nbr, expression [[,;]expression] … etc	Same as the normal PRINT command except that the output is directed to a file previously opened for OUTPUT or APPEND as '#fnbr' or to a serial communications port previously opened as 'nbr'. See the OPEN command. #0 can be used which refers to the console.
PRINT #GPS, string$	Outputs a NMEA string to an opened GPS device. The string must start with a $ character and end with a * character. The checksum is calculated automatically by the firmware and is appended to the string together with the carriage return and line feed characters.

PRINT @(x [, y]) expression Or PRINT @(x, [y], m) expression	Same as the standard PRINT command except that the cursor is positioned at the coordinates x, y expressed in pixels. If y is omitted the cursor will be positioned at "x" on the current line. Example: PRINT @(150, 45) "Hello World" The @ function can be used anywhere in a print command. Example: PRINT @(150, 45) "Hello" @(150, 55) "World" The @(x,y) function can be used to position the cursor anywhere on or off the screen. For example, PRINT @(-10, 0) "Hello" will only show "llo" as the first two characters could not be shown because they were off the screen. The @(x,y) function will automatically suppress the automatic line wrap normally performed when the cursor goes beyond the right screen margin. If 'm' is specified the mode of the video operation will be as follows: m = 0 Normal text (white letters, black background) m = 1 The background will not be drawn (ie, transparent) m = 2 The video will be inverted (black letters, white background) m = 5 Current pixels will be inverted (transparent background)
PULSE pin, width	Will generate a pulse on 'pin' with duration of 'width' ms. 'width' can be a fraction. For example, 0.01 is equal to 10µs and this enables the generation of very narrow pulses. The generated pulse is of the opposite polarity to the state of the I/O pin when the command is executed. For example, if the output is set high the PULSE command will generate a negative going pulse. Notes: • 'pin' must be configured as an output. • For a pulse of less than 3 ms the accuracy is ± 1 µs. • For a pulse of 3 ms or more the accuracy is ± 0.5 ms. • A pulse of 3 ms or more will run in the background. Up to five different and concurrent pulses can be running in the background and each can have its time changed by issuing a new PULSE command or it can be terminated by issuing a PULSE command with zero for 'width'.
PWM 1, freq, 1A or PWM 1, freq, 1A, 1B or PWM 1, freq, 1A, 1B, 1C or PWM 2, freq, 2A or PWM 2, freq, 2A, 2B or PWM channel, STOP	Generate a pulse width modulated (PWM) output for driving analog circuits, sound output, etc. There are a total of five outputs designated as PWM in the diagrams on pages 6 and 7 (they are also used for the SERVO command). Controller 1 can have one, two or three outputs while controller 2 can have one or two outputs. Both controllers are independent and can be turned on and off and have different frequencies. '1' or '2' is the controller number and 'freq' is the output frequency . 1A, 1B and 1C are the duty cycle for each of the controller 1 outputs while 2A and 2B are the duty cycle for the controller 2 outputs. The specified I/O pins will be automatically configured as outputs while any others will be unaffected and can be used for other duties. The duty cycle for each output is independent of the others and is specified as a percentage. If it is close to zero the output will be a narrow positive pulse, if 50 a square wave will be generated and if close to 100 it will be a very wide positive pulse Minimum frequency is 1Hz, maximum is 24MHz. Duty cycle and frequency accuracy will depend on frequency. The frequency can be any value of 240,000,000/n. The output will run continuously in the background while the program is running and can be stopped using the STOP command. The frequency and duty cycle can be changed at any time (without stoping the

	output) by issuing a new PWM command. The PWM function will take control of any specified outputs and when stopped the pins will be returned to a high impedance "not configured" state.
RBOX x, y, w, h [, r] [,c] [,fill]	Draws a box with rounded corners on the VGA monitor starting at 'x' and 'y' which is 'w' pixels wide and 'h' pixels high. 'r' is the radius of the corners of the box. It defaults to 10. 'c' specifies the colour and defaults to the default foreground colour if not specified. 'fill' is the fill colour. It can be omitted or set to -1 in which case the box will not be filled. 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', '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 chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.
READ variable[, variable]...	Reads values from DATA statements and assigns these values to the named variables. Variable types in a READ statement must match the data types in DATA statements as they are read. Arrays can be used as variables (specified with empty brackets, eg, a()) and in that case the size of the array is used to determine how many elements are to be read. If the array is multidimensional then the leftmost dimension will be the fastest moving. See also DATA and RESTORE.
REM string	REM allows remarks to be included in a program. Note the Microsoft style use of the single quotation mark to denote remarks is also supported and is preferred.
RENAME old$ AS new$	Rename a file or a directory from ‘old$’ to ‘new$’. Both are strings. A directory path can be used in both 'old$' and 'new$'. If the paths differ the file specified in 'old$' will be moved to the path specified in 'new$' with the file name as specified.
RESTORE [line]	Resets the line and position counters for the READ statement. If ‘line’ is specified the counters will be reset to the beginning of the specified line. ‘line’ can be a line number or label. A variable can also be used as the parameter. In that case a numerical variable should be used for a line number and a string variable for a label If ‘line’ is not specified the counters will be reset to the start of the program.
RMDIR dir$	Remove, or delete, the directory ‘dir$’ on the SD card.
RUN file$ or RUN file$, cmdline	Run the program 'file$' held on the SD card. Note that 'file$' must be a string constant (ie, "MYPROG.BAS") including the quotes required around a string constant. It cannot be a variable or expression. If 'cmdline' is specified it will be available to the running program as the string returned by MM.CMDLINE$. 'cmdline' is not processed by MMBasic so it can contain numbers, commas, quoted strings, etc. It is the responsibility of the running program to decode this string of characters. 'file$' can be omitted and in that case MMBasic will run the "current

	program name" which is the file last used by RUN, EDIT or AUTOSAVE.
SAVE DATA fname$, address, size	Saves "size" bytes to file "fname$" starting from "address". This allows areas of the Colour Maximite 2 memory to be saved as binary files. See also LOAD DATA
SAVE IMAGE file$ [,x, y, w, h]	Save the current image on the VGA screen as a 24-bit BMP file. 'file$' is the name of the file. If an extension is not specified ".BMP" will be added to the file name. 'x', 'y', 'w' and 'h' are optional and are the coordinates (x and y are the top left coordinate) and dimensions (width and height) of the area to be saved. If not specified the whole screen will be saved.
SEEK [#]fnbr, pos	Will position the read/write pointer in a file that has been opened on the SD card for RANDOM access to the 'pos' byte. The first byte in a file is numbered one so SEEK #5,1 will position the read/write pointer to the start of the file.

SELECT CASE value
CASE testexp [[, testexp] …]
<statements>
<statements>
CASE ELSE
<statements>
<statements>
END SELECT
	Executes one of several groups of statements, depending on the value of an expression. 'value' is the expression to be tested. It can be a number or string variable or a complex expression. 'testexp' is the value that 'exp' is to be compared against. It can be:
	• A single expression (ie, 34, "string" or PIN(4)*5) to which it may equal
	• A range of values in the form of two single expressions separated by the keyword "TO" (ie, 5 TO 9 or "aa" TO "cc")
	• A comparison starting with the keyword "IS" (which is optional). For example: IS > 5, IS <= 10.
	When a number of test expressions (separated by commas) are used the CASE statement will be true if any one of these tests evaluates to true.
	If 'value' cannot be matched with a 'testexp' it will be automatically matched to the CASE ELSE. If CASE ELSE is not present the program will not execute any <statements> and continue with the code following the END SELECT.
	When a match is made the <statements> following the CASE statement will be executed until END SELECT or another CASE is encountered when the program will then continue with the code following the END SELECT.
	An unlimited number of CASE statements can be used but there must be only one CASE ELSE and that should be the last before the END SELECT.
	
	Example:
	`SELECT CASE nbr%`
	`CASE 4, 9, 22, 33 TO 88`
	`statements`
	`CASE IS < 4, IS > 88, 5 TO 8`
	`statements`
	`CASE ELSE`
	`statements`
	`END SELECT`
	Each SELECT CASE must have one and one only matching END SELECT statement. Any number of SELECT…CASE statements can be nested inside the CASE statements of other SELECT…CASE statements.

SERVO 1 [, freq], 1A or SERVO 1 [, freq], 1A, 1B or SERVO 1 [, freq], 1A, 1B, 1C or SERVO 2 [, freq], 2A or SERVO 2 [, freq], 2A, 2B or SERVO channel, STOP	Generate a constant stream of positive going pulses for driving a servo. The Maximite has two servo controllers with the first being able to control up to three servos and the second two servos. Both controllers are independent and can be turned on and off and have different frequencies. This command uses the I/O pins that are designated as PWM in the external I/O diagram (the two commands are very similar). '1' or '2' is the controller number. 'freq' is the output frequency (between 20Hz and 1000 Hz) and is optional. If not specified it will default to 50 Hz 1A, 1B and 1C are the pulse widths for each of the controller 1 outputs while 2A and 2B are the pulse widths for the controller 2 outputs. The specified I/O pins will be automatically configured as outputs while any others will be unaffected and can be used for other duties. The pulse width for each output is independent of the others and is specified in milliseconds, which can be a fractional number (ie, 1.536). For accurate positioning the output resolution is about 0.005 ms. The minimum value is 0.01ms while the maximum is 18.9ms. Most servos will accept a range of 0.8ms to 2.2ms. The output will run continuously in the background while the program is running and can be stopped using the STOP command. The pulse widths of the outputs can be changed at any time (without stoping the output) by issuing a new SERVO command. The SERVO function will take control of any specified outputs and when stopped the pins will be returned to a high impedance "not configured" state.
SETPIN pin, cfg [, option]	Will configure an external I/O pin. 'pin' is the I/O pin to configure, 'cfg' is the mode that the pin is to be set to and 'option' is an optional parameter. 'cfg' is a keyword and can be any one of the following: OFF — Not configured or inactive AIN — Analog input (ie, measure the voltage on the input). 'option' can be used to specify the number of bits in the conversion. Valid values are 8, 10, 12, 14, and 16. The default (if not specified) is 16 bits. The more bits the longer the conversion will take. A single conversion takes between 0.2mSec (8-bit) to 0.9mSec (16-bit). DIN — Digital input If 'option' is omitted the input will be high impedance If 'option' is the keyword "PULLUP" a simulated resistor will be used to pull up the input pin to 3.3V If the keyword "PULLDOWN" is used the pin will be pulled down to zero volts. The pull up/down is a constant current of about 50µA. FIN — Frequency input 'option' can be used to specify the gate time (the length of time used to count the input cycles). It can be any number between 10 ms and 100000 ms. Note that the PIN() function will always return the frequency correctly scaled in Hz regardless of the gate time used. If 'option' is omitted the gate time will be 1 second. PIN — Period input 'option' can be used to specify the number of input cycles to average the period measurement over. It can be any number between 1 and 10000. Note that the PIN() function will always return the average period of one cycle correctly scaled in ms regardless of the number of cycles used for the average. If 'option' is omitted the period of just one cycle will be used. CIN — Counting input DOUT — Digital output 'option' can be "OC" in which case the output will be open

	collector (or more correctly open drain). The functions PIN() and PORT() can also be used to return the value on one or more output pins . Previous versions of MMBasic used numbers for 'cfg' and the mode OOUT. For backwards compatibility they will still be recognised. See the function PIN() for reading inputs and the statement PIN()= for setting an output. See the command below if an interrupt is configured.
SETPIN pin, cfg, target [, option]	Will configure ‘pin’ to generate an interrupt according to ‘cfg’. Any I/O pin capable of digital input can be configured to generate an interrupt with a maximum of ten interrupts configured at any one time. 'cfg' is a keyword and can be any one of the following: OFF Not configured or inactive INTH Interrupt on low to high input INTL Interrupt on high to low input INTB Interrupt on both (ie, any change to the input) ‘target' is a user defined subroutine which will be called when the event happens. Return from the interrupt is via the END SUB or EXIT SUB commands. 'option' can be the keywords "PULLUP" or "PULLDOWN" as specified for a normal input pin (SETPIN pin DIN). If 'option' is omitted the input will be high impedance. This mode also configures the pin as a digital input so the value of the pin can always be retrieved using the function PIN().
SETTICK period, target [, nbr] SETTICK FAST frequency, target	This will setup a periodic interrupt (or "tick"). Four tick timers are available ('nbr' = 1, 2, 3 or 4). 'nbr' is optional and defaults to timer number 1. The time between interrupts is ‘period’ milliseconds and ‘target' is the interrupt subroutine which will be called when the timed event occurs. The period can range from 1 to 2147483647 ms (about 24 days). These interrupts can be disabled by setting ‘period’ to zero This allows you to exceed the current maximum rate of 1 interrupt per millisecond (1000Hz) and has been tested up to 50KHz. If the interrupt routine overruns the time available then interrupts will be lost. If the program is executing a statement that takes longer than the time between interrupts the interrupts will be stacked.
SORT array() [,indexarray()] [,flags] [,startposition] [,elementstosort]	This command takes an array of any type (integer, float or string) and sorts it into ascending order in place. It has an optional parameter ‘indexarray%()’. If used this must be an integer array of the same size as the array to be sorted. After the sort this array will contain the original index position of each element in the array being sorted before it was sorted. Any data in the array will be overwritten.This allows connected arrays to be sorted. See the section *Sorting Data* in the tutorial Programming with the Colour Maximite 2 for an example. The ‘flag’ parameter is optional and valid flag values are: bit0: 0 (default if omitted) normal sort - 1 reverse sort bit1: 0 (default) case dependent - 1 sort is case independent (string sorts only). The optional ‘startposition’ defines which element in the array to start the sort. Default is 0 (OPTION BASE 0) or 1 (OPTION BASE 1) The optional ‘elementstosort’ defines how many elements in the array should be sorted. The default is all elements after the startposition.

	Any of the optional parameters may be omitted so, for example, to sort just the first 50 elements of an array you could use: SORT array(), , , ,50
SPI OPEN speed, mode, bits or SPI READ nbr, array() or SPI WRITE nbr, data1, data2, data3, … etc or SPI WRITE nbr, string$ or SPI WRITE nbr, array() or SPI CLOSE	Communications via an SPI channel. The command SPI refers to channel 1. The command SPI2 refers to channel 2 and has an identical syntax. 'nbr' is the number of data items to send or receive 'data1', 'data2', etc can be float or integer and in the case of WRITE can be a constant or expression. If 'string$' is used 'nbr' characters will be sent. 'array' must be a single dimension float or integer array and 'nbr' elements will be sent or received. See Appendix D for the details.
SPRITE	The SPRITE commands are used to manipulate small graphic images on the VGA screen. These are useful when writing games. The maximum size of a sprite is MM.HRES-1 and MM.VRES-1 See also the SPRITE() functions.
SPRITE CLOSE [#]n	Closes sprite “n” and releases its memory resources allowing the sprite number to be re-used. The command will give an error if other sprites are copied from this one unless they are closed first.
SPRITE CLOSE ALL	Closes all sprites and releases all sprite memory. The screen is not changed.
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 HIDE ALL	Hides all the sprites allowing the background to be manipulated. The following commands cannot be used when all sprites are hidden: SPRITE SHOW (SAFE) SPRITE HIDE (SAFE, ALL) SPRITE SWAP SPRITE MOVE SPRITE SCROLLR SPRITE SCROLL
SPRITE HIDE SAFE [#]n	Removes sprite n from the display and replaces the stored background. Automatically hides all more recent sprites as well as the requested one and then replaces them afterwards. This ensures that sprites that are covered by other sprites can be removed without the user tracking the write order. Of course this version is less performant than the simple version and should only be used it there is a risk of the sprite being partially covered.
SPRITE INTERRUPT sub	Specifies the name of the subroutine that will be called when a sprite collision occurs. See Appendix E for how to use the function SPRITE to interrogate details of what has collided

SPRITE LOAD fname$ [,start_sprite_number]	Loads the file ‘fname$’ which must be formatted as an original Colour Maximite sprite file. See the original Colour Maximite *MMBasic Language Manual* for the file format. Multiple sprite files can be loaded by specifying a different ‘start_sprite_number’ for each file. The programmer is responsible for making sure that the sprites do not overlap.
SPRITE LOADARRAY [#]n, w, h, array%()	Creates the sprite 'n' with width 'w' and height 'h' by reading w*h RGB888 values from 'array%()'. The RGB888 values must be stored in order of columns across and then rows down starting at the top left. This allows the programmer to create simple sprites in a program without needing to load them from disk or read them from the display. The firmware will generate an error if 'array%()' is not big enough to hold the number of values required.
SPRITE LOADPNG [#]n, fname$ [, transparency_cut_off]	Loads the PNG image ‘fname$’ as sprite number ‘n’. If the PNG file is in ARGB8888 format the ‘transparency_cut_off’ parameter is used to determine whether the pixel should be solid or missing/transparent. Valid values are 1 to 15, default is 8. MMBasic compares the 4 most significant bits of the transparency data in the file with the cut off value and assigns a transparency of 0 or 15 depending on the comparison. This allows RGB(0,0,0) to be a valid solid colour. If the file is in RGB888 format then an RGB level of 0,0,0 is used to determine transparency as there is no other information to use.
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 RESTORE	Restores all the sprites. NB that any position changes previously requested using SPRITE NEXT will be actioned by the RESTORE and collision detection will be run
SPRITE READ [#]n, x , y, w, h [,pagenumber]	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. The optional parameter page number specifies which page is to be read to create the sprite. The default is the current write page. Set the page to FRAMEBUFFER to read from the framebuffer – see the FRAMEBUFFER command.
SPRITE SCROLL x, y [,col]	Scrolls the background and any sprites on layer 0 'x' pixels to the right and 'y' pixels up. 'x' can be any number between -MM.HRES-1 and MM.HRES-1, 'y' 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. If 'col' is set to -1 the

	scrolled area will be left untouched.
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 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 SHOW [#]n, x,y, 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 and will be replaced when the sprite is hidden or moved further. ‘orientation’ is optional and can be: 0 - normal display (default if omitted) 1 - mirrored left to right 2 - mirrored top to bottom 3 - rotated 180 degrees (= 1+2)
SPRITE SHOW SAFE [#]n, x,y, layer [,orientation] [,ontop]	Shows a sprite and automatically compensates for any other sprites that overlap it. If the sprite is not already being displayed the command acts exactly the same as SPRITE SHOW. If the sprite is already shown it is moved and remains in its position relative to other sprites based on the original order of writing. i.e. if sprite 1 was written before sprite 2 and it is moved to overlap sprite 2 it will display under sprite 2. If the optional "ontop" parameter is set to 1 then the sprite moved will become the newest sprite and will sit on top of any other sprite it overlaps. Refer to SPRITE SHOW for details of the orientation parameter.
SPRITE SWAP [#]n1, [#]n2 [,orientation]	Replaces the sprite ‘n1’ with the sprite ‘n2’. The sprites must have the same width and height and ‘n1’ must be displayed or an error will be generated. Refer to SPRITE SHOW for details of the orientation parameter. The replacement sprite inherits the background from the original as well as its position in the list of order drawn.
SPRITE TRANSPARENCY [#]n, transparency	Transparency can be between 1 and 15 and changes all pixels with a non-zero transparency in the stored sprite to the new level.
SPRITE WRITE [#]n, x y [,orientation]	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. The optional 'orientation' parameter defaults to 4 and specifies how the stored image data is changed as it is written out. It is the bitwise AND of the following values: &B001 = mirrored left to right &B010 = mirrored top to bottom &B100 = don't copy transparent pixels

STATIC variable [, variables] See DIM for the full syntax.	Defines a list of variable names which are local to the subroutine or function. These variables will retain their value between calls to the subroutine or function (unlike variables created using the LOCAL command). This command uses exactly the same syntax as DIM. The only difference is that the length of the variable name created by STATIC and the length of the subroutine or function name added together cannot exceed 32 characters. Static variables can be initialised to a value. This initialisation will take effect only on the first call to the subroutine (not on subsequent calls).
SUB xxx (arg1 [,arg2, …]) <statements> <statements> END SUB	Defines a callable subroutine. This is the same as adding a new command to MMBasic while it is running your program. 'xxx' is the subroutine name and it must meet the specifications for naming a variable. 'arg1', 'arg2', etc are the arguments or parameters to the subroutine. An array is specified by using empty brackets. ie, arg3(). The type of the argument can be specified by using a type suffix (ie, arg1$) or by specifying the type using `AS <type> (ie, arg1 AS STRING)`. Every definition must have one END SUB statement. When this is reached the program will return to the next statement after the call to the subroutine. The command EXIT SUB can be used for an early exit. You use the subroutine by using its name and arguments in a program just as you would a normal command. For example: `MySub a1, a2` When the subroutine is called each argument in the caller is matched to the argument in the subroutine definition. These arguments are available only inside the subroutine. Subroutines can be called with a variable number of arguments. Any omitted arguments in the subroutine's list will be set to zero or a null string. Arguments in the caller's list that are a variable and have the correct type will be passed by reference to the subroutine. This means that any changes to the corresponding argument in the subroutine will also be copied to the caller's variable and therefore may be accessed after the subroutine has ended. Arrays are passed by specifying the array name with empty brackets (eg, arg()) and are always passed by reference. Brackets around the argument list in both the caller and the definition are optional.
TEMPR START pin [, precision]	This command can be used to start a conversion running on a DS18B20 temperature sensor connected to 'pin'. Normally the TEMPR() function alone is sufficient to make a temperature measurement so usage of this command is optional. This command will start the measurement on the temperature sensor. The program can then attend to other duties while the measurement is running and later use the TEMPR() function to get the reading. If the TEMPR() function is used before the conversion time has completed the function will wait for the remaining conversion time before returning the value. Any number of these conversions (on different pins) can be started and be running simultaneously. 'precision' is the resolution of the measurement and is optional. It is a number between 0 and 3 meaning: 0 = 0.5ºC resolution, 100 ms conversion time. 1 = 0.25ºC resolution, 200 ms conversion time (this is the default). 2 = 0.125ºC resolution, 400 ms conversion time. 3 = 0.0625ºC resolution, 800 ms conversion time.

TEXT x, y, string$ [,alignment$] [, font] [, scale] [, c] [, bc]	Displays a string on the VGA monitor starting at 'x' and 'y'. 'string$' is the string to be displayed. Numeric data should be converted to a string and formatted using the Str$() function. ' alignment$' is a string expression or string variable consisting of 0, 1 or 2 letters 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. The default alignment is left/top. A third letter can be used in the alignment 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 (ie, upside down), 'U' the text will be rotated counter clockwise by 90º and 'D' the text will be rotated clockwise by 90º 'font' and 'scale' are optional and default to that set by the FONT command. 'c' is the drawing colour and 'bc' is the background colour. They are optional and default to the current foreground and background colours. See the chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates.
TIME$ = "HH:MM:SS" or TIME$ = "HH:MM" or TIME$ = "HH"	Sets the time of the internal clock. MM and SS are optional and will default to zero if not specified. For example TIME$ = "14:30" will set the clock to 14:30 with zero seconds. Note: • The time will be set to "00:00:00" on first power up. • The time will be remembered and kept updated as long as the battery is installed and can maintain a voltage of over 2.5V. • IMPORTANT: The date must also be set (using DATE$=) otherwise the correct time will be lost after the power is cycled. • Battery life should be 3 to 4 years even if the computer is powered off.
TIME$ = ±sec	Adds or subtracts 'sec' seconds from the current time being maintained by MMBasic. This makes it easier to fine tune the current time.
TIMER = msec	Resets the timer to a number of milliseconds. Normally this is just used to reset the timer to zero but you can set it to any positive integer. See the TIMER function for more details.
TRACE ON or TRACE OFF or TRACE LIST nn	TRACE ON/OFF will turn on/off the trace facility. This facility will print the number of each line (counting from the beginning of the program) in square brackets as the program is executed. This is useful in debugging programs. TRACE LIST will list the last 'nn' lines executed in the format described above. MMBasic is always logging the lines executed so this facility is always available (ie, it does not have to be turned on).
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	Draws a triangle on the VGA monitor with the corners at X1, Y1 and X2, Y2 and X3, Y3. 'C' is the colour of the triangle and defaults to the current foreground colour. 'FILL' is the fill colour and defaults to no fill (it can also be set to -1 for no fill). All parameters can be expressed as arrays and the software will plot the number of triangles as determined by the dimensions of the smallest array unless X1 = Y1 = X2 = Y2 = X3 = Y3 = -1 in which case processing will stop at that point '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.

TURTLE	The TURTLE commands are a simple method of drawing graphic images on the VGA screen. They are suitable for newcomers and students.
TURTLE RESET	Clears the screen and re-initialises the turtle system. The turtle is located at MM.HRES\2, MM.VRES\2. The default pen colour is white and the fill colour is green. The initial heading is up (0 degrees)
TURTLE DRAW TURTLE	Draws a turtle at the current location and in the current orientation
TURTLE DRAW PIXEL x, y	Draw a 1-pixel dot at the given location using the current draw colour, regardless of current turtle location or pen status.
TURTLE FILL PIXEL x, y	Draw a 1-pixel dot at the given location using the current fill colour, regardless of current turtle location or pen status.
TURTLE DRAW LINE x1, y1, x2, y2	Draw a straight line between the given coordinates, regardless of current turtle location or pen status.
TURTLE DRAW CIRCLE x, y, r	Draw a circle at the given coordinates with the given radius, regardless of current turtle location or pen status.
TURTLE PEN UP	Lifts the pen so moves do not write to the screen
TURTLE PEN DOWN	Lowers the pen so moves do write to the screen
TURTLE FORWARD n	Moves the turtle n pixels in the current heading
TURTLE BACKWARD n	Moves the turtle n pixels opposite the current heading
TURTLE DOT	Draw a 1-pixel dot at the current location, regardless of pen status
TURTLE TURN LEFT deg	Turn the turtle to the left (anti-clockwise) by the specified number of degrees.
TURTLE TURN RIGHT deg	Turn the turtle to the right (clockwise) by the specified number of degrees.
TURTLE BEGIN FILL	Start filling. Call this before drawing a polygon to activate the bookkeeping required to run the filling algorithm later.
TURTLE END FILL	End filling. Call this after drawing a polygon to trigger the fill algorithm. The filled polygon may have up to 128 sides.
TURTLE HEADING deg	Rotate the turtle to the given absolute heading (in degrees). 0 degrees means facing straight up. 90 degrees means facing to the right.
TURTLE PEN COLOUR col	Set the current drawing colour. Colours are specified as per normal drawing commands
TURTLE FILL COLOUR col	Set the current fill colour. Colours are specified as per normal drawing commands
TURTLE MOVE x, y	Move the turtle to the specified location, drawing a straight line if the pen is down.

UPDATE FIRMWARE	Switch the ARM Cortex-M7 CPU into firmware update mode. This is the same as switching the BOOT CONFIG switch on the Waveshare CPU board to SYSTEM or moving the BOOT0 jumper to PROG and allows for quick firmware updates without opening the Maximite's case. You should cycle the power following a firmware update, this will return the ARM Cortex-M7 CPU to normal mode. Cycling the power will also recover if this command needed to be cancelled.
VAR SAVE var [, var]… or VAR RESTORE or VAR CLEAR	VAR SAVE will save one or more variables to non volatile memory where they can be restored later (normally after a power interruption). 'var' can be any number of numeric or string variables and/or arrays. Arrays are specified by using empty brackets. For example: var() VAR RESTORE will retrieve the previously saved variables and insert them (and their values) into the variable table. The VAR SAVE command can be used repeatedly. Variables that had been previously saved will be updated with their new value and any new variables (not previously saved) will be added to the saved list for later restoration. VAR CLEAR will erase all saved variables. Also, the saved variables will be automatically cleared by the NEW command or when a new program is loaded via AUTOSAVE, XMODEM, etc. This command is normally used to save calibration data, options, and other data which needs to be retained across a power interruption. Normally the VAR RESTORE command is placed at the start of the program so that previously saved variables are restored and immediately available to the program when it starts. Notes: • The storage space available to this command is 4KB. The memory used is battery backed RAM which operates at high speed and can be written to an unlimited number of times without restriction (unlike the Micromite). • Using VAR RESTORE without a previous save will have no effect and will not generate an error. • If, when using RESTORE, a variable with the same name already exists its value will be overwritten. • Saved arrays must be declared (using DIM) before they can be restored. • Be aware that string arrays can rapidly use up all the memory allocated to this command. The LENGTH qualifier can be used when a string array is declared to reduce the size of the array (see the DIM command). This is not needed for ordinary string variables.
WATCHDOG timeout or WATCHDOG OFF	Starts the watchdog timer which will automatically restart the processor when it has timed out. This can be used to recover from some event that disabled the running program (such as an endless loop or a programming or other error that halts a running program). This can be important in an unattended control situation. 'timeout' is the time in milliseconds (ms) before a restart is forced. This command should be placed in strategic locations in the running BASIC program to constantly reset the watchdog timer and therefore prevent it from counting down to zero. If the timer count does reach zero (perhaps because the BASIC program has stopped running) the Maximite will be restarted and the automatic variable MM.WATCHDOG will be set to true (ie, 1) indicating that an error occurred. On a normal startup MM.WATCHDOG will be set to false (ie, 0).

	WATCHDOG OFF will disable the watchdog timer (this is the default on a reset or power up). The timer is also turned off when the break character (normally CTRL-C) is used on the console to interrupt a running program.
WII CLASSIC OPEN [n] [,interrupt [,bitmask]] or WII CLASSIC CLOSE [n]	See the CONTROLLER COMMAND
WII NUNCHUK OPEN [n] [,Zinterrupt [,Cinterrupt]] or WII NUNCHUK CLOSE [n]	See the CONTROLLER COMMAND
XMODEM SEND file$ [,comportno] or XMODEM RECEIVE file$ [,comportno]	Transfers a BASIC program to or from a remote computer using the XModem protocol. The transfer is done over the serial console connection. XMODEM SEND will send 'file$' held on the Colour Maximite's SD card to the remote device. SEND can be abbreviated to S. XMODEM RECEIVE will accept 'file$' sent by the remote device and save it on the Colour Maximite's SD card. If the file already exists it will be overwritten when receiving a file. RECEIVE can be abbreviated to R. The XModem protocol requires a cooperating software program running on the remote computer and connected to its serial port. It has been tested on Tera Term running on Windows and it is recommended that this be used. After running the XMODEM command in MMBasic select: File -> Transfer -> XMODEM -> Receive/Send from the Tera Term menu to start the transfer. The transfer can take up to 15 seconds to start and if the XMODEM command fails to establish communications it will return to the MMBasic prompt after 60 seconds and leave the program memory untouched. If 'commportno' is specified the transfer will take place over the serial port specified (1 or 2). In this case the port must have been previously opened with an appropriate baudrate. Download Tera Term from http://ttssh2.sourceforge.jp/

Functions

Note that the functions related to communications functions (I^2C, 1-Wire, and SPI) are not listed here but are described in the appendices at the end of this document.

Square brackets indicate that the parameter or characters are optional.

ABS(number)	Returns the absolute value of the argument 'number' (ie, any negative sign is removed and the positive number is returned).
ACOS(number)	Returns the inverse cosine of the argument 'number' in radians.
ASC(string$)	Returns the ASCII code for the first letter in the argument ‘string$’.
ASIN(number)	Returns the inverse sine value of the argument 'number' in radians.
ATAN2(y, x)	Returns the arc tangent of the two numbers x and y as an angle expressed in radians. It is similar to calculating the arc tangent of y / x, except that the signs of both arguments are used to determine the quadrant of the result.
ATN(number)	Returns the arctangent of the argument 'number' in radians.
BASE$(base, number [, chars])	Returns a string giving the base value for the 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s). Base can be between 2 and 36. Numbers greater than 9 are represented by a letter as per the HEX notation. Example: PRINT BASE$(36, 35) will display "Z" Internally the functions BIN$, OCT$, and HEX$ use this function.
BAUDRATE(comm [, timeout])	Returns the baudrate of any data received on the serial communications port 'comm'). This will sample the port over the period of 'timeout' seconds. 'timeout' will default to one second if not specified. Returns zero if no activity on the port within the timeout period.
BIN$(number [, chars])	Returns a string giving the binary (base 2) value for the 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).
BIN2STR$(type, value [,BIG])	Returns a string containing the binary representation of 'value'. 'type' can be: INT64 signed 64-bit integer converted to an 8 byte string UINT64 unsigned 64-bit integer converted to an 8 byte string INT32 signed 32-bit integer converted to a 4 byte string UINT32 unsigned 32-bit integer converted to a 4 byte string INT16 signed 16-bit integer converted to a 2 byte string UINT16 unsigned 16-bit integer converted to a 2 byte string INT8 signed 8-bit integer converted to a 1 byte string UINT8 unsigned 8-bit integer converted to a 1 byte string SINGLE single precision floating point number converted to a 4 byte string DOUBLE double precision floating point number converted to a 8 byte string By default the string contains the number in little-endian format (ie, the least significant byte is the first one in the string). Setting the third parameter to

	'BIG' will return the string in big-endian format (ie, the most significant byte is the first one in the string) In the case of the integer conversions, an error will be generated if the 'value' cannot fit into the 'type' (eg, an attempt to store the value 400 in a INT8). This function makes it easy to prepare data for efficient binary file I/O or for preparing numbers for output to sensors and saving to flash memory. See also the function STR2BIN
BOUND(array() [,dimension]	This returns the upper limit of the array for the dimension requested. The dimension defaults to one if not specified. Specifying a dimension value of 0 will return the current value of OPTION BASE. Unused dimensions will return a value of zero. For example: DIM myarray(44,45) BOUND(myarray(),2) will return 45
CALL(userfunname$, [,userfunparameters,....])	This is an efficient way of programmatically calling user defined functions. (See also the CALL command). In many cases it can be used to eliminate complex SELECT and IF THEN ELSEIF ENDIF clauses and is processed in a much more efficient manner. "userfunname$" can be any string or variable or function that resolves to the name of a normal user function (not an in-built command). "userfunparameters" are the same parameters that would be used to call the function directly. A typical use for this command could be writing any sort of emulator where one of a large number of functions should be called depending on a some variable. It also provides a method of passing a function name to another subroutine or function as a variable.
CHOICE(condition, ExpressionIfTrue, ExpressionIfFalse)	This function allows you to do simple either/or selections more efficiently and faster than using IF THEN ELSE ENDIF clauses. The condition is anything that will resolve to nonzero (true) or zero (false). The expressions are anything that you could normally assign to a variable or use in a command and can be integers, floats or strings. Examples: PRINT CHOICE(1, "hello","bye") will print "Hello" PRINT CHOICE (0, "hello","bye") will print "Bye" a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) will print 4
CHR$(number)	Returns a one-character string consisting of the character corresponding to the ASCII code indicated by argument 'number'.
CINT(number)	Round numbers with fractional portions up or down to the next whole number or integer. For example, 45.47 will round to 45 45.57 will round to 46 -34.45 will round to -34 -34.55 will round to -35 See also INT() and FIX().
CLASSIC(funct [, channel])	Returns data from a Wii Classic controller. 'channel' is optional and is the I^2C channel for the controller (defaults to 3, the front panel). 'funct' is a 1 or 2 letter code indicating the information to return as follows:

	LX returns the position of the analog left joystick x axis LY returns the position of the analog left joystick y axis RX returns the position of the analog right joystick x axis RY returns the position of the analog right joystick y axis L returns the position of the analog left button R returns the position of the analog right button B returns a bitmap of the state of all the buttons. A bit will be set to 1 if the button is pressed. T returns the ID code of the controller - should be hex &H4200101 The button bitmap is as follows: BIT 0: Button R BIT 1: Button start BIT 2: Button home BIT 3: Button select BIT 4: Button L BIT 5: Button down cursor BIT 6: Button right cursor BIT 7: Button up cursor BIT 8: Button left cursor BIT 9: Button ZR BIT 10: Button x BIT 11: Button a BIT 12: Button y BIT 13: Button b BIT 14: Button ZL These bit positions are also used in the interrupt bitmask specified in the WII CLASSIC OPEN command
COS(number)	Returns the cosine of the argument 'number' in radians.
CWD$	Returns the current working directory on the SD card as a string. The format is: `A:/dir1/dir2`. See also MM.INFO(DIRECTORY) which will return the same thing but will always have a '/' character at the end
DATE$	Returns the current date based on MMBasic's internal clock as a string in the form "DD-MM-YYYY". For example, "28-07-2012". The internal clock/calendar will keep track of the time and date including leap years. To set the date use the command DATE$ =.
DATETIME$(n)	Returns the date and time corresponding to the epoch number n (number of seconds that have elapsed since midnight GMT on January 1, 1970). The format of the returned string is "dd-mm-yyyy hh:mm:ss". Use the text NOW to get the current datetime string, i.e. ? DATETIME$(NOW)
DAY$(date$)	Returns the day of the week for a given date as a string "Monday", "Tuesday" etc. The format for date$ is "DD-MM-YY", "DD-MM-YYYY", or "YYYY-MM-DD". Use NOW to get the day for the current date, e.g. PRINT DAY$(NOW)
DEG(radians)	Converts 'radians' to degrees.

DIR$(fspec, type)
or
DIR$(fspec)
or
DIR$( )
	Will search an SD card for files and return the names of entries found.
	'fspec' is a file specification using wildcards the same as used by the FILES command. Eg, "*" will return all entries, "*.TXT" will return text files.
	'type' is the type of entry to return and can be one of:
	ALL Search for both files and directories
	DIR Search for directories only
	FILE Search for files only (the default if 'type' is not specified)
	The function will return the first entry found. To retrieve subsequent entries use the function with no arguments. ie, DIR$( ). The return of an empty string indicates that there are no more entries to retrieve.
	This example will print all the files in a directory:
	`f$ = DIR$("*", FILE)`
	`DO WHILE f$ <> ""`
	`PRINT f$`
	`f$ = DIR$()`
	`LOOP`
	You must change to the required directory before invoking this command.

DISTANCE(trigger, echo) or DISTANCE(trig-echo)	Measure the distance to a target using the HC-SR04 ultrasonic distance sensor. Four pin sensors have separate trigger and echo connections. 'trigger' is the I/O pin connected to the "trig" input of the sensor and 'echo' is the pin connected to the "echo" output of the sensor. Three pin sensors have a combined trigger and echo connection and in that case you only need to specify one I/O pin to interface to the sensor. Note that any I/O pins used with the HC-SR04 should be 5V capable as the HC-SR04 is a 5V device. The I/O pins are automatically configured by this function and multiple sensors can be used on different I/O pins. The value returned is the distance in centimetres to the target or -1 if no target was detected or -2 if there was an error (ie, sensor not connected).
DRAW3D()	V5.07.00 includes a 3D engine. See the document “The CMM2 3D engine” for full details
EOF([#]nbr)	Will return true if the file previously opened on the SD card for INPUT with the file number ‘#fnbr’ is positioned at the end of the file. For a serial communications port this function will return true if there are no characters waiting in the receive buffer. #0 can be used which refers to the console's input buffer. The # is optional. Also see the OPEN, INPUT and LINE INPUT commands and the INPUT$ function.
EPOCH(DATETIME$)	Returns the epoch number (number of seconds that have elapsed since midnight GMT on January 1, 1970) for the supplied DATETIME$ string. The format for DATETIME$ is “dd-mm-yyyy hh:mm:ss”, “dd-mm-yy hh:mm:ss”, or “yyyy-mm-dd hh:mm:ss”,. Use NOW to get the epoch number for the current date and time, i.e. PRINT EPOCH(NOW)
EVAL(string$)	Will evaluate 'string$' as if it is a BASIC expression and return the result. 'string$' can be a constant, a variable or a string expression. The expression can use any operators, functions, variables, subroutines, etc that are known at the time of execution. The returned value will be an integer, float or string depending on the result of the evaluation.

	For example: S$ = "COS(RAD(30)) * 100" : PRINT EVAL(S$) Will display: 86.6025
EXP(number)	Returns the exponential value of 'number', ie, e^x where x is 'number'.

FIELD$(string1, nbr, string2 [, string3])	Returns a particular field in a string with the fields separated by delimiters.
	'nbr' is the field to return (the first is nbr 1). 'string1' is the string to search and 'string2' is a string holding the delimiters (more than one can be used).
	'string3' is optional and if specified will include characters that are used to quote text in 'string1' (ie, quoted text will not be searched for a delimiter).
	For example:
	`S$ = "foo, boo, zoo, doo"`
	`r$ = FIELD$(s$, 2, ",")`
	will result in r$ = "boo". While:
	`s$ = "foo, 'boo, zoo', doo"`
	`r$ = FIELD$(s$, 2, ",", "'")`
	will result in r$ = "boo, zoo".

FIX(number)	Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point. For example 9.89 will return 9 and -2.11 will return -2. The major difference between FIX and INT is that FIX provides a true integer function (ie, does not return the next lower number for negative numbers as INT() does). This behaviour is for Microsoft compatibility. See also CINT() .
FORMAT$(nbr [, fmt$])	Will return a string representing 'nbr' formatted according to the specifications in the string 'fmt$'. The format specification starts with a % character and ends with a letter. Anything outside of this construct is copied to the output as is. The structure of a format specification is: % [flags] [width] [.precision] type Where 'flags' can be: - Left justify the value within a given field width 0 Use 0 for the pad character instead of space + Forces the + sign to be shown for positive numbers space Causes a positive value to display a space for the sign. Negative values still show the – sign 'width' is the minimum number of characters to output, less than this the number will be padded, more than this the width will be expanded. 'precision' specifies the number of fraction digits to generate with an e, or f type or the maximum number of significant digits to generate with a g type. If specified, the precision must be preceded by a dot (.). 'type' can be one of: g Automatically format the number for the best presentation. f Format the number with the decimal point and following digits e Format the number in exponential format If uppercase G or F is used the exponential output will use an uppercase E. If the format specification is not specified "%g" is assumed. Examples: format$(45) will return 45 format$(45, "%g") will return 45

	format$(24.1, “%g”) will return 24.1 format$(24.1,”%f”) will return 24.100000 format$(24.1, “%e”) will return 2.410000e+01 format$(24.1,"%09.3f") will return 00024.100 format$(24.1,"%+.3f") will return +24.100 format$(24.1,"**%-9.3f**") will return **24.100 **
GETSCANLINE	This will report on the line that is currently being drawn on the VGA monitor. Using this to time updates to the screen can avoid timing effects caused by updates while the screen is being updated. The first visible line will return a value of 0. Any line number above MM.VRES is in the frame blanking period.
GPS()	The GPS functions are used to return data from a serial communications channel opened as GPS. The function GPS(VALID) should be checked before any of these functions are used to ensure that the returned value is valid.
GPS(ALTITUDE)	Returns current altitude (if sentence GGA is enabled).
GPS(DATE)	Returns the normal date string corrected for local time e.g. “12-01-2020”.
GPS(DOP)	Returns DOP (dilution of precision) value (if sentence GGA is enabled).
GPS(FIX)	Returns DOP (dilution of precision) value (if sentence GGA is enabled).
GPS(GEOID)	Returns the geoid-ellipsoid separation (if sentence GGA is enabled).
GPS(LATITUDE)	Returns the latitude in degrees as a floating point number, values are negative for South of equator
GPS LONGITUDE)	Returns the longitude in degrees as a floating point number, values are negative for West of the meridian.
GPS(SATELLITES)	Returns number of satellites in view (if sentence GGA is enabled).
GPS(SPEED)	Returns the ground speed in knots as a floating point number.
GPS(TIME)	Returns the normal time string corrected for local time e.g. “12:09:33”.
GPS(TRACK)	Returns the track over the ground (degrees true) as a floating point number.
GPS(VALID)	Returns: 0=invalid data, 1=valid data
HEX$(number [, chars])	Returns a string giving the hexadecimal (base 16) value for the 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).
INKEY$	Checks the console input buffer and, if there is one or more characters waiting in the queue, will remove the first character and return it as a single character in a string. If this is a carriage return, it is likely that there will be a line feed character following as often the enter key will produce a CR/LF pair. If the input buffer is empty this function will immediately return with an empty string (ie, "").

INPUT$(nbr, [#]fnbr)	Will return a string composed of ‘nbr’ characters read from a file on the SD card previously opened for INPUT with the file number ‘#fnbr’. This function will read all characters including carriage return and new line without translation. Will return a string composed of ‘nbr’ characters read from a serial communications port opened as 'fnbr'. This function will return as many characters as are waiting in the receive buffer up to ‘nbr’. If there are no characters waiting it will immediately return with an empty string. #0 can be used which refers to the console's input buffer. The # is optional. Also see the OPEN command.
INSTR([start-position,] string-searched$, string-pattern$)	Returns the position at which 'string-pattern$' occurs in 'string-searched$', beginning at 'start-position'. Both the position returned and 'start-position' use 1 for the first character, 2 for the second, etc. The function returns zero if 'string-pattern$' is not found.
INT(number)	Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behaviour is for Microsoft compatibility, the FIX() function provides a true integer function. See also CINT() .
JSON$(array%(),string$)	Returns a string representing a specific item out of the JSON input stored in the longstring array%() Examples taken from api.openweathermap.org JSON$(a%(), “name”) JSON$(a%(), “coord.lat”) JSON$(a%(), “weather[0].description”) JSON$(a%(),”list[4].weather[0].description
KEYDOWN(n)	Return the decimal ASCII value of the USB keyboard key that is currently held down or zero if no key is down. The decimal values for the function and arrow keys are listed in Appendix F. This function will report multiple simultaneous key presses and the parameter 'n' is the number of the keypress to report. KEYDOWN(0) will return the number of keys being pressed For example, if "c", "g" and "p" are pressed simultaneously KEYDOWN(0) will return 3, KEYDOWN(1) will return 99, KEYDOWN(2) will return 103, etc. The keys do not need to be pressed simultaneously and will report in the order pressed. Taking a finger off a key will promote the next key pressed to #1. The first key ('n' = 1) is entered in the keyboard buffer (accessible using INKEY$) while keys 2 to 6 can only be accessed via this function. Using this function will clear the console input buffer. KEYDOWN(7) will give any modifier keys that are pressed. These keys do not add to the count in keydown(0) The return value is a bitmask as follows: lalt ? 1, lctrl ? 2, lgui ? 4, lshift ? 8, ralt ? 16, rctrl ? 32, rgui ? 64, rshift ? 128 KEYDOWN(8) will give the current status of the lock keys. These keys do not add to the count in keydown(0) The return value is a bitmask as follows: caps_lock ? 1, num_lock ? 2, scroll_lock ? 4 Note that some keyboards will limit the number of active keys that they can report on.

LCASE$(string$)	Returns ‘string$’ converted to lowercase characters.
LCOMPARE(array1%(), array2%())	Compare the contents of two long string variables array1%() and array2%(). The returned is an integer and will be -1 if array1%() is less than array2%(). It will be zero if they are equal in length and content and +1 if array1%() is greater than array2%(). The comparison uses the ASCII character set and is case sensitive.
LEFT$(string$, nbr)	Returns a substring of ‘string$’ with ‘nbr' of characters from the left (beginning) of the string.
LEN(string$)	Returns the number of characters in 'string$'.
LGETBYTE(array%(), n)	Returns the numerical value of the 'n'th byte in the LONGSTRING held in 'array%()'. This function respects the setting of OPTION BASE in determining which byte to return.
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%()
LOC([#]fnbr)	For a file on the SD card opened as RANDOM this will return the current position of the read/write pointer in the file. Note that the first byte in a file is numbered 1. For a serial communications port opened as 'fnbr' this function will return the number of bytes received and waiting in the receive buffer to be read. #0 can be used which refers to the console's input buffer. The # is optional.
LOF([#]fnbr)	For a file on the SD card this will return the current length of the file in bytes. For a serial communications port opened as 'fnbr' this function will return the space (in characters) remaining in the transmit buffer. Note that when the buffer is full MMBasic will pause when adding a new character and wait for some space to become available. The # is optional.
LOG(number)	Returns the natural logarithm of the argument 'number'.
MAP(number)	This function returns the RGB332 colour that is used internally to map to the colour lookup table (CLUT) when using 8-bit colour depths. For example MAP(255) will always return &HE0E0C0 which the firmware will then convert to 255. This is a way of selecting the normal colour that will find the place in the CLUT you want. It is not a window into the CLUT. See also the MAP command.

MATH	The math function performs many simple mathematical calculations that can be programmed in Basic but there are speed advantages to coding looping structures in C and there is the advantage that once debugged they are there for everyone without re-inventing the wheel.
Simple functions	
MATH(ATAN3 x,y)	Returns ATAN3 of x and y
MATH(COSH a)	Returns the hyperbolic cosine of a
MATH(LOG10 a)	Returns the base 10 logarithm of a
MATH(SINH a)	Returns the hyperbolic sine of a
MATH(TANH a)	Returns the hyperbolic tan of a
Simple Statistics	
MATH(CHI a())	Returns the Pearson's chi-squared value of the two dimensional array a())
MATH(CHI_p a())	Returns the associated probability in % of the Pearson's chi-squared value of the two dimensional array a())
MATH(CORREL a(), a())	Returns the Pearson's correlation coefficient between arrays a() and b()
MATH(MAX a())	Returns the maximum of all values in the a() array, a() can have any number of dimensions
MATH(MEAN a())	Returns the average of all values in the a() array, a() can have any number of dimensions
MATH(MEDIAN a())	returns the median of all values in the a() array, a() can have any number of dimensions
MATH(MIN a())	Returns the minimum of all values in the a() array, a() can have any number of dimensions
MATH(SD a())	Returns the standard deviation of all values in the a() array, a() can have any number of dimensions
MATH(SUM a())	Returns the sum of all values in the a() array, a() can have any number of dimensions
Vector Arithmetic	
MATH(MAGNITUDE v())	Returns the magnitude of the vector v(). The vector can have any number of elements
MATH(DOTPRODUCT v1(), v2())	Returns the dot product of two vectors v1() and v2(). The vectors can have any number of elements but must have the same cardinality
Matrix Arithmetic	
MATH(M_DETERMINANT array!())	Returns the determinant of the array. The array must be square.

MAX(arg1 [, arg2 [, …]]) or MIN(arg1 [, arg2 [, …]])	Returns the maximum or minimum number in the argument list. Note that the comparison is a floating point comparison (integer arguments are converted to floats) and a float is returned.
MID$(string$, start) or MID$(string$, start, nbr)	Returns a substring of 'string$' beginning at 'start' and continuing for 'nbr' characters. The first character in the string is number 1. If 'nbr' is omitted the returned string will extend to the end of 'string$'
MOUSE(funct [, channel])	Returns data from a Mouse controller supporting the Hobbytronic protocol. 'channel' is optional and is the I^2C channel for the controller (defaults to 2, pins 27 and 28). 'funct' is a 1 letter code indicating the information to return as follows: X returns the value of the mouse X-position Y returns the value of the mouse Y-position L returns the value of the left mouse button (1 if pressed) R returns the value of the right mouse button (1 if pressed) W returns the value of the scroll wheel mouse button (1 if pressed) D This allows you to detect a double click of the left mouse button . The algorithm say the two clicks must occur between 100 and 500 milliseconds apart. The report via MOUSE(D) is then valid for 500mSec before it times out or until it is read. T returns 0
MOUSE(funct, 0)	Returns data from a PS2 mouse 'funct' is a 1 letter code indicating the information to return as follows: X returns the value of the mouse X-position Y returns the value of the mouse Y-position L returns the value of the left mouse button (1 if pressed) R returns the value of the right mouse button (1 if pressed) W returns the value of the scroll wheel mouse button (1 if pressed) D This allows you to detect a double click of the left mouse button. The algorithm requires that the two clicks must occur between 100 and 500 milliseconds apart. The report via MOUSE(D) is then valid for 500mSec before it times out or until it is read. T This returns 3 if a PS2 mouse has a scroll wheel or 0 if not.
NUNCHUK(channel, funct)	Returns data from a Nunchuk controller. 'channel' is optional and is the I^2C channel for the controller (defaults to 3, the front panel). 'funct' is a 2 or 3 letter code indicating the information to return as follows: JX returns the value of the joystick X-axis JY returns the value of the joystick Y-axis AX returns the value of the x acceleration AY returns the value of the y acceleration AZ returns the value of the z acceleration Z returns the value of the z button (1 if pressed) C returns the value of the c button (1 if pressed) T returns the id code of the Wii device: &HA4200000=Original, &HA4200101=Classic, &HA4200402=Balance JXL returns the calibrated X value of the joystick in the far left position

	JXC returns the calibrated X value of the joystick in the centre position JXR returns the calibrated X value of the joystick in the far right position JYT returns the calibrated Y value of the joystick in the top position JYC returns the calibrated Y value of the joystick in the centre position JYB returns the calibrated Y value of the joystick in the bottom position AX0 returns the calibrated zero gravity value of the x-axis accelerometer AX1 returns the calibrated one gravity value of the x-axis accelerometer AY0 returns the calibrated zero gravity value of the y-axis accelerometer AY1 returns the calibrated one gravity value of the y-axis accelerometer AZ0 returns the calibrated zero gravity value of the z-axis accelerometer AZ1 returns the calibrated one gravity value of the z-axis accelerometer
OCT$(number [, chars])	Returns a string giving the octal (base 8) representation of 'number'. 'chars' is optional and specifies the number of characters in the string with zero as the leading padding character(s).
PEEK(BYTE addr%) PEEK(SHORT addr%) PEEK(WORD addr%) PEEK(INTEGER addr%) PEEK(FLOAT addr% PEEK(VARADDR var) PEEK(VARHEADER var) PEEK(VAR var, ±offset) PEEK(VARTBL, ±offset) PEEK(PROGMEM, ±offset)	Will return a byte or a word within the CPU's virtual memory space. BYTE will return the byte (8-bits) located at 'addr%' SHORT will return the short integer (16-bits) located at 'addr%' WORD will return the word (32-bits) located at 'addr%' INTEGER will return the integer (64-bits) located at 'addr%' FLOAT will return the floating point number (64-bits) located at 'addr%' VARADDR will return the address (32-bits) of the variable 'var' in memory. An array is specified as var(). VARHEADER will return the address (32-bits) of the variable descriptor of the variable var in memory. An array is specified as var() VAR, will return a byte in the memory allocated to 'var'. An array is specified as var(). VARTBL, will return a byte in the memory allocated to the variable table maintained by MMBasic. Note that there is a comma after VARTBL. PROGMEM, will return a byte in the memory allocated to the program. Note that there is a comma after the keyword PROGMEM. Note that 'addr%' should be an integer.
PI	Returns the value of pi.
PIN(pin)	Returns the value on the external I/O 'pin'. Zero means digital low, 1 means digital high and for analog inputs it will return the measured voltage as a floating point number. Frequency inputs will return the frequency in Hz. A period input will return the period in milliseconds while a count input will return the count since reset (counting is done on the positive rising edge). The count input can be reset to zero by resetting the pin to counting input (even if it is already so configured). This function will also return the state of a pin configured as an output. Also see the SETPIN and PIN() = commands.
PIN(function)	Returns the value of a special function. 'function' is a string (ie, it can be a string variable or string constant). For example PRINT PIN("BAT"). It can be one of: "BAT" The voltage of the backup battery.

	"TEMP" The temperature of the ARM Cortex-M7 processor's core. "DAC1" The output voltage of DAC1 (on the audio output). "DAC2" The output voltage of DAC2 (on the audio output). "SREF" The stored calibrated value of the internal reference voltage measured with a supply of exactly 3.3V. This is programmed into the chip during production. "IREF" The measured value of the internal reference voltage. The actual value of VREF+ can be calculated as: 3.3 * PIN("SREF") / PIN("IREF") and this can be used to set OPTION VCC.
PIXEL(x, y [,page_number])	Returns the colour of a pixel on the VGA monitor. 'x' is the horizontal coordinate and 'y' is the vertical coordinate of the pixel. See the chapter "Basic Drawing Commands" for a definition of the colours and graphics coordinates. The optional parameter page_number specifies which page is to be read. The default is the current write page. Set the page number to FRAMEBUFFER to read from the framebuffer – see the FRAMEBUFFER command
PORT(start, nbr [,start, nbr]…)	Returns the value of a number of I/O pins in one operation. 'start' is an I/O pin number and its value will be returned as bit 0. 'start'+1 will be returned as bit 1, 'start'+2 will be returned as bit 2, and so on for 'nbr' number of bits. I/O pins used must be numbered consecutively and any I/O pin that is invalid or not configured as an input will cause an error. The start/nbr pair can be repeated up to 25 times if additional groups of input pins need to be added. This function will also return the state of a pin configured as an output. It can be used to conveniently communicate with parallel devices like memory chips. Any number of I/O pins (and therefore bits) can be used from 1 to the number of I/O pins on the chip. See the PORT command to simultaneously output to a number of pins.
PULSIN(pin, polarity) or PULSIN(pin, polarity, t1) or PULSIN(pin, polarity, t1, t2)	Measures the width of an input pulse from 1µs to 1 second with 0.1µs resolution. 'pin' is the I/O pin to use for the measurement, it must be previously configured as a digital input. 'polarity' is the type of pulse to measure, if zero the function will return the width of the next negative pulse, if non zero it will measure the next positive pulse. 't1' is the timeout applied while waiting for the pulse to arrive, 't2' is the timeout used while measuring the pulse. Both are in microseconds (µs) and are optional. If 't2' is omitted the value of 't1' will be used for both timeouts. If both 't1' and 't2' are omitted then the timeouts will be set at 100000 (ie, 100ms). This function returns the width of the pulse in microseconds (µs) or -1 if a timeout has occurred. The measurement is accurate to ±1 µs. Note that this function will cause the running program to pause while the measurement is made and interrupts will be ignored during this period.
RAD(degrees)	Converts 'degrees' to radians.
RGB(red, green, blue [, trans]) or RGB(shortcut [, trans])	Generates an RGB true colour value. 'red', 'blue' and 'green' represent the intensity of each colour. A value of zero represents black and 255 represents full intensity. 'shortcut' allows common colours to be specified by naming them. The colours that can be named are white, black, blue, green, cyan, red, magenta, yellow, brown and grey or gray (USA spelling). For example, RGB(red) or

	RGB(cyan). There is also one special colour ‘notblack’, For any video mode this is the darkest colour that is not treated as black by various graphics commands. 'trans' is the level of transparency for colour depths 4 and 12. It is optional and defaults to 15 if not specified.
RIGHT$(string$, number-of-chars)	Returns a substring of ‘string$’ with ‘number-of-chars’ from the right (end) of the string.
RND(number) or RND	Returns a pseudo-random number in the range of 0 to 0.999999. The 'number' value is ignored if supplied. The Colour Maximite 2 uses the hardware random number generator in the ARM Cortex-M7 to deliver true random numbers. This means that the RANDOMIZE command is no longer needed and is not supported.
SGN(number)	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN(number)	Returns the sine of the argument 'number' in radians.
SPACE$(number)	Returns a string of blank spaces 'number' characters long.
SPI(data) or SPI2(data)	Send and receive data using an SPI channel. A single SPI transaction will send data while simultaneously receiving data from the slave. ‘data’ is the data to send and the function will return the data received during the transaction. ‘data’ can be an integer or a floating point variable or a constant.
SPRITE()	The SPRITE functions return information regarding sprites which are small graphic images on the VGA screen. These are useful when writing games. See also the SPRITE commands.
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. If the collision was with the edge of the screen then the return value will be: &HF1 collision with left of screen &HF2 collision with top of screen &HF4 collision with right of screen &HF8 collision with bottom of screen
SPRITE(D ,[#]s1, [#]s2)	Returns the distance between the centres of sprites ‘s1’ and ‘s2’ (returns -1 if either sprite is not active)
SPRITE(E, [#]n	Returns a bitmap indicating any edges of the screen the sprite is in contact with: 1 =left of screen, 2=top of screen, 4=right of screen, 8=bottom of screen
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(V,spriteno1,spriteno2)	Returns the vector from 'spriteno1' to 'spriteno2' in radians. The angle is based on the clock so if 'spriteno2' is above 'spriteno1' on the screen then the answer will be zero. This can be used on any pair of sprites that are visible. If either sprite is not visible the function will return -1. This is particularly useful after a collision if the programmer wants to make some differential decision based on where the collision occurred. The angle is calculated between the centre of each of the sprites which may of course be different sizes.
SPRITE(T, [#]n)	Returns a bitmap showing all the sprites currently touching the requested sprite Bits 0-63 in the returned integer represent a current collision with sprites 1 to 64 respectively
SPRITE(V,[#]so1, [#]s2)	Returns the vector from sprite 's1' to 's2' in radians. The angle is based on the clock so if 's2' is above 's1' on the screen then the answer will be zero. This can be used on any pair of sprites that are visible. If either sprite is not visible the function will return -1. This is particularly useful after a collision if the programmer wants to make some differential decision based on where the collision occurred. The angle is calculated between the centre of each of the sprites which may of course be different sizes.
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.
STR2BIN(type, string$ [,BIG])	Returns a number equal to the binary representation in ‘string$’. ‘type’ can be: INT64 converts 8 byte string representing a signed 64-bit integer to an integer UINT64 converts 8 byte string representing an unsigned 64-bit integer to an integer INT32 converts 4 byte string representing a signed 32-bit integer to an integer UINT32 converts 4 byte string representing an unsigned 32-bit integer to an integer INT16 converts 2 byte string representing a signed 16-bit integer to an integer UINT16 converts 2 byte string representing an unsigned 16-bit integer to an integer INT8 converts 1 byte string representing a signed 8-bit integer to an integer UINT8 converts 1 byte string representing an unsigned 8-bit integer to an integer SINGLE converts 4 byte string representing single precision float to a float DOUBLE converts 8 byte string representing single precision float to a float By default the string must contain the number in little-endian format (ie, the least significant byte is the first one in the string). Setting the third

	parameter to 'BIG' will interpret the string in big-endian format (ie, the most significant byte is the first one in the string). This function makes it easy to read data from binary data files, interpret numbers from sensors or efficiently read binary data from flash memory chips. An error will be generated if the string is the incorrect length for the conversion requested See also the function BIN2STR$
SQR(number)	Returns the square root of the argument 'number'.
STR$(number) or STR$(number, m) or STR$(number, m, n) or STR$(number, m, n, c$)	Returns a formatted string in decimal (base 10) representation of 'number'. If 'm' is specified sufficient spaces will be added to the start of the number to ensure that the number of characters before the decimal point (including the negative or positive sign) will be at least 'm' characters. If 'm' is zero or the number has more than 'm' significant digits no padding spaces will be added. If 'm' is negative, positive numbers will be prefixed with the plus symbol and negative numbers with the negative symbol. If 'm' is positive then only the negative symbol will be used. 'n' is the number of digits required to follow the decimal place. If it is zero the string will be returned without the decimal point. If it is negative the output will always use the exponential format with 'n' digits resolution. If 'n' is not specified the number of decimal places and output format will vary automatically according to the number. 'c$' is a string and if specified the first character of this string will be used as the padding character instead of a space (see the 'm' argument). Examples: STR$(123.456) will return "123.456" STR$(-123.456) will return "-123.456" STR$(123.456, 1) will return "123.456" STR$(123.456, -1) will return "+123.456" STR$(123.456, 6) will return " 123.456" STR$(123.456, -6) will return " +123.456" STR$(-123.456, 6) will return " -123.456" STR$(-123.456, 6, 5) will return " -123.45600" STR$(-123.456, 6, -5) will return " -1.23456e+02" STR$(53, 6) will return " 53" STR$(53, 6, 2) will return " 53.00" STR$(53, 6, 2, "*") will return "****53.00"
STRING$(nbr, ascii) or STRING$(nbr, string$)	Returns a string 'nbr' bytes long consisting of either the first character of string$ or the character representing the ASCII value 'ascii' which is a decimal number in the range of 32 to 126.
TAB(number)	Outputs spaces until the column indicated by 'number' has been reached on the console output. The tab function will not work when printing to a file but will behave like the SPACE$ function.
TAN(number)	Returns the tangent of the argument 'number' in radians.
TEMPR(pin)	Return the temperature measured by a DS18B20 temperature sensor connected to 'pin' (which does not have to be configured). The returned value is degrees C with a default resolution of 0.25ºC. If there is an error during the measurement the returned value will be 1000.

	The time required for the overall measurement is 200ms and interrupts will be ignored during this period. Alternatively the TEMPR START command can be used to start the measurement and your program can do other things while the conversion is progressing. When this function is called the value will then be returned instantly assuming the conversion period has expired. If it has not, this function will wait out the remainder of the conversion time before returning the value. The DS18B20 can be powered separately by a 3V to 5V supply or it can operate on parasitic power. See the chapter "Special Hardware Devices" for more details.
TIME$	Returns the current time based on MMBasic's internal clock as a string in the form "HH:MM:SS" in 24 hour notation. For example, "14:30:00". If the OPTION MILLISECONDS ON command has been used this function will return the time including milliseconds as a decimal fraction of the seconds. For example: "14:35:06.239". To set the current time use the command TIME$ = .
TIMER	Returns the elapsed time in milliseconds (eg, 1/1000 of a second) since reset. This is a fractional floating point number with a resolution of 1µs. The timer is reset to zero on power up or a CPU restart and you can also reset it to any value by using TIMER as a command.
UCASE$(string$)	Returns ‘string$’ converted to uppercase characters.
VAL(string$)	Returns the numerical value of the ‘string$’. If 'string$' is an invalid number the function will return zero. This function will recognise the &H prefix for a hexadecimal number, &O for octal and &B for binary.

Obsolete Commands and Functions

These commands and functions are mostly included to assist in converting programs written for Microsoft BASIC. For new programs the corresponding modern commands in MMBasic should be used.

These commands may be removed in the future to recover memory for other features.

DHT22	Use HUMID
GOSUB target	Initiates a subroutine call to the target, which can be a line number or a label. The subroutine must end with RETURN. New programs should use defined subroutines (ie, SUB…END SUB).
IF condition THEN linenbr	For Microsoft compatibility a GOTO is assumed if the THEN statement is followed by a number. A label is invalid in this construct. New programs should use: IF condition THEN GOTO linenbr | label
IRETURN	Returns from an interrupt when the interrupt destination was a line number or a label. New programs should use a user defined subroutine as an interrupt destination. In that case END SUB or EXIT SUB will cause a return from the interrupt.
ON nbr GOTO | GOSUB target[,target, target,...]	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of 'nbr'; if it is 1, the first target is called, if 2, the second target is called, etc. Target can be a line number or a label. New programs should use SELECT CASE.
POS	For the console, returns the current cursor position in the line in characters.
RETURN	RETURN concludes a subroutine called by GOSUB and returns to the statement after the GOSUB.

Appendix A
Serial Communications

Two serial ports are available for asynchronous serial communications labelled COM1: and COM2:. In addition, if the serial console is disabled then that port is available as COM3:.

After being opened the serial port will have an associated file number and you can use any commands that operate with a file number to read and write to/from it. A serial port is also closed using the CLOSE command.

The following is an example:

```
OPEN "COM1:4800" AS #5    ' open the first serial port with a speed of 4800 baud
PRINT #5, "Hello"         ' send the string "Hello" out of the serial port
dat$ = INPUT$(20, #5)     ' get up to 20 characters from the serial port
CLOSE #5                  ' close the serial port
```

The OPEN Command

A serial port is opened using the command:

```
OPEN comspec$ AS #fnbr
```

'fnbr' is the file number to be used. It must be in the range of 1 to 10. The # is optional.

'comspec$' is the communication specification and is a string (it can be a string variable) specifying the serial port to be opened and optional parameters. The default is 9600 baud, 8 data bits, no parity and one stop bit.

It has the form `"COMn: baud, buf, int, int-trigger, 7BIT, (ODD or EVEN), INV, OC, S2"` where:

- 'n' is the serial port number for either COM1:, COM2 or COM3:.:.
- 'baud' is the baud rate. This can be any value between 1200 (the minimum) and 1000000 (1MHz). Default is 9600.
- 'buf' is the receive buffer size in bytes (default size is 256). The transmit buffer is fixed at 256 bytes.
- 'int' is a user defined subroutine which will be called when the serial port has received some data. The default is no interrupt.
- 'int-trigger' sets the trigger condition for calling the interrupt subroutine. It is an integer and the interrupt subroutine will be called when this number of characters has arrived in the receive queue.

All parameters except the serial port name (COMn:) are optional. If any parameter is left out then all the following parameters must also be left out and the defaults will be used.

The following options can be added to the end of 'comspec$'

- 'INV' specifies that the transmit and receive polarity is inverted. Default is non inverted.
- 'OC' will force the transmit pin to be open collector. The default is normal (0 to 3.3V) output.
- 'S2' specifies that two stop bits will be sent following each character transmitted. Default is one stop bit.
- '7BIT' will specify that 7 bit transmit and receive is to be used. Default is 8 bits.
- 'ODD' will specify that an odd parity bit will be appended (8 bits will be transmitted if 7BIT is specified, otherwise 9)
- 'EVEN' will specify that an even parity bit will be appended (8 bits will be transmitted if 7BIT is specified, otherwise 9)
- 'DEP' will enable RS485 mode with a positive output on the COM1-DE pin
- 'DEN' will enable RS485 mode with a negative output on the COM1-DE pin

Input/Output Pin Allocation

When a serial port is opened the pins used by the port will be automatically set to input or output as required and the SETPIN and PIN commands will be disabled for the pins. When the port is closed (using the CLOSE command) all pins used by the serial port will be set to a not-configured state and the SETPIN command can then be used to reconfigure them.

The connections for each COM port are shown in the I/O connector pinout diagrams in the beginning of this manual. Note that Tx means an output from the Maximite and Rx means an input to the Maximite.

The signal polarity is standard for devices running at TTL voltages (for RS232 voltages see below). Idle is voltage high, the start bit is voltage low, data uses a high voltage for logic 1 and the stop bit is voltage high.

These signal levels allow you to directly connect to devices like GPS modules (which generally use TTL voltage levels).

When a serial port is opened MMBasic will enable an internal pullup resistor (to Vdd) on the Rx (receive data) pin. This has a value of about 100K and its purpose is to prevent the input from floating if it is left unconnected. Normally this is fine but it can cause a problem if you have an external resistor in series with the Rx pin, in that case this resistor and the pullup resistor will form a voltage divider limiting how high or low the voltage on the Rx pin can swing and that in turn might mean that the input signal is not recognised. The solution is to use the command SERIAL PULLUP DISABLE to disable it.

Examples

Opening a serial port using all the defaults:

```
OPEN "COM2:" AS #2
```

Opening a serial port specifying only the baud rate (4800 bits per second):

```
OPEN "COM2:4800" AS #1
```

Opening a serial port specifying the baud rate (9600 bits per second) and receive buffer size (1KB):

```
OPEN "COM1:9600, 1024" AS #8
```

The same as above but with two stop bits enabled:

```
OPEN "COM1:9600, 1024, S2" AS #8
```

An example specifying everything including an interrupt, an interrupt level, inverted and two stop bits:

```
OPEN "COM1:19200, 1024, ComIntLabel, 256, INV, S2" AS #5
```

Reading and Writing

Once a serial port has been opened you can use any command or function that uses a file number to read from and write to the port. Data received by the serial port will be automatically buffered in memory by MMBasic until it is read by the program and the INPUT$() function is the most convenient way of doing that. When using the INPUT$() function the number of characters specified will be the maximum number of characters returned but it could be less if there are less characters in the receive buffer. In fact the INPUT$() function will immediately return an empty string if there are no characters available in the receive buffer.

The LOC() function is also handy; it will return the number of characters waiting in the receive buffer (ie, the maximum number characters that can be retrieved by the INPUT$() function). Note that if the receive buffer overflows with incoming data the serial port will automatically discard the oldest data to make room for the new data.

The PRINT command is used for outputting to a serial port and any data to be sent will be held in a memory buffer while the serial port is sending it. This means that MMBasic will continue with executing the commands after the PRINT command while the data is being transmitted. The one exception is if the output buffer is full and in that case MMBasic will pause and wait until there is sufficient space before continuing. The LOF() function will return the amount of space left in the transmit buffer and you can use this to avoid stalling the program while waiting for space in the buffer to become available.

If you want to be sure that all the data has been sent (perhaps because you want to read the response from the remote device) you should wait until the LOF() function returns 256 (the transmit buffer size) indicating that there is nothing left to be sent.

Serial ports can be closed with the CLOSE command. This will wait for the transmit buffer to be emptied then free up the memory used by the buffers, cancel the interrupt (if set) and set all pins used by the port to the not configured state. A serial port is also automatically closed when commands such as RUN and NEW are issued.

Interrupts

The interrupt subroutine (if specified) will operate the same as a general interrupt on an external I/O pin.

When using interrupts you need to be aware that it will take some time for MMBasic to respond to the interrupt and more characters could have arrived in the meantime, especially at high baud rates. For example, if you have specified the interrupt level as 250 characters and a buffer of 256 characters then quite easily the buffer will have overflowed by the time the interrupt subroutine can read the data. In this case the buffer should be increased to 512 characters or more.

Low Cost RS-232 Interface

The RS-232 signalling system is used by modems, hardwired serial ports on a PC, test equipment, etc. It is the same as the serial TTL system used on the Colour Maximite 2 with two exceptions:

- The voltage levels of RS-232 are +12V and -12V where TTL serial uses +3.3V and zero volts.
- The signalling is inverted (the idle voltage is -12V, the start bit is +12V, etc).

It is possible to purchase cheap RS-232 to TTL converters on the Internet but it would be handy if it was possible to directly interface to RS-232.

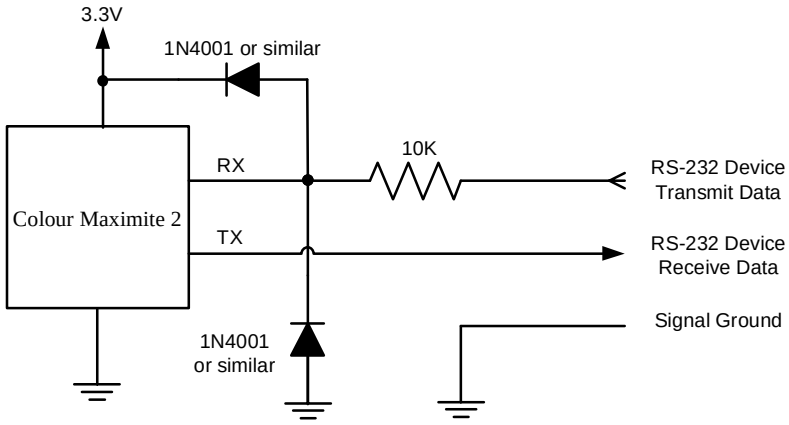
The first issue is that the signalling polarity is inverted with respect to TTL. On the Colour Maximite 2 COM1: can be specified to invert the transmit and receive signal (the 'INV' option) so that is an easy fix.

For the receive data (that is the ±12V signal from the remote RS-232 device) it is easy to limit the voltage using a series resistor of (say) 10KΩ and two diodes that will clamp the input voltage to the 3.3V rail and ground. The input impedance of the I/O pin is very high so the resistor will not cause a voltage drop but it does mean that when the signal swings to the maximum ±12V it will be safely clipped by the diodes.

For the transmit signal (from the Colour Maximite 2 to the RS-232 device) you can connect this directly to the input of the remote device. The output will only swing the signal from zero to 3.3V but most RS-232 inputs have a threshold of about +1V so the signal will still be interpreted as a valid signal.

These measures break the rules for RS-232 signalling, but if you only want to use it over a short distance (a metre or two) it should work fine.

Use this circuit:



And open COM1: with the invert option. For example:

```
OPEN "COM1: 4800, INV" AS #1
```

Appendix B
I²C Communications

The Colour Maximite 2 implements three I²C channels, two on the rear I/O connector and the third dedicated to the front panel Wii connector. All operate in master mode (slave mode is not available).

There are four commands that can be used:

I2C OPEN speed, timeout

Enables the I²C module in master mode. The I2C command refers to channel 1 while commands I2C2 and I2C3 refer to channels 2 and 3 using the same syntax.

'speed' is the clock speed (in KHz) to use and must be one of 100, 400 or 1000.

'timeout' is a value in milliseconds after which the master send and receive commands will be interrupted if they have not completed. The minimum value is 100. A value of zero will disable the timeout (though this is not recommended).

I2C WRITE addr, option, sendlen, senddata [,sendata ....]

Send data to the I²C slave device. The I2C command refers to channel 1 while commands I2C2 and I2C3 refer to channels 2 and 3 using the same syntax.

'addr' is the slave's I²C address.

'option' can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command)

'sendlen' is the number of bytes to send.

'senddata' is the data to be sent - this can be specified in various ways (all values sent will be between 0 and 255):

- The data can be supplied as individual bytes on the command line. Example: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25
- The data can be in a one dimensional array specified with empty brackets (ie, no dimensions). 'sendlen' bytes of the array will be sent starting with the first element. Example: I2C WRITE &H6F, 0, 3, ARRAY()
- The data can be a string variable (not a constant). Example: I2C WRITE &H6F, 0, 3, STRING$

I2C READ addr, option, rcvlen, rcvbuf

Get data from the I²C slave device. The I2C command refers to channel 1 while commands I2C2 and I2C3 refer to channels 2 and 3 using the same syntax.

'addr' is the slave's I²C address.

'option' can be 0 for normal operation or 1 to keep control of the bus after the command (a stop condition will not be sent at the completion of the command)

'rcvlen' is the number of bytes to receive.

'rcvbuf' is the variable or array used to save the received data - this can be:

- A string variable. Bytes will be stored as sequential characters in the string.
- A one dimensional array of numbers specified with empty brackets. Received bytes will be stored in sequential elements of the array starting with the first. Example: I2C READ &H6F, 0, 3, ARRAY()
- A normal numeric variable (in this case rcvlen must be 1).

I2C CLOSE

Disables the master I²C module and returns the I/O pins to a "not configured" state. They can then be configured using SETPIN. This command will also send a stop if the bus is still held.

The I2C command refers to channel 1 while commands I2C2 and I2C3 refer to channels 2 and 3 using the same syntax.

Following an I²C write or read command the automatic variable MM.I2C will be set to indicate the result of the operation as follows:

0 = The command completed without error.
1 = Received a NACK response
2 = Command timed out

7-Bit Addressing

The standard addresses used in these commands are 7-bit addresses (without the read/write bit). MMBasic will add the read/write bit and manipulate it accordingly during transfers.

Some vendors provide 8-bit addresses which include the read/write bit. You can determine if this is the case because they will provide one address for writing to the slave device and another for reading from the slave. In these situations you should only use the top seven bits of the address. For example: If the read address is 9B (hex) and the write address is 9A (hex) then using only the top seven bits will give you an address of 4D (hex).

Another indicator that a vendor is using 8-bit addresses instead of 7-bit addresses is to check the address range. All 7-bit addresses should be in the range of 08 to 77 (hex). If your slave address is greater than this range then probably your vendor has specified an 8-bit address.

I/O Pins

Refer to the rear panel I/O connector diagram at the beginning of this manual for the pin numbers used for the I²C channels 1 and 2. Their signals are marked as data line (SDA) and clock (SCL). I²C channel 3 is routed to the Wii connector on the front panel. When the I2C CLOSE command is used the I/O pins are reset to a "not configured" state. Then can then be configured as per normal using SETPIN.

Both the data line (SDA) and clock (SCL) for all three I²C ports have 10K pullup resistors installed on the motherboard so external pullup resistors are not required. These should be considered if these pins are to be used as general purpose I/O pins.

When running the I²C bus at above 100 kHz the cabling between the devices becomes important. Ideally the cables should be as short as possible (to reduce capacitance) and also the data and clock lines should not run next to each other but have a ground wire between them (to reduce crosstalk).

If the data line is not stable when the clock is high, or the clock line is jittery, the I²C peripherals can get "confused" and end up locking the bus (normally by holding the clock line low). If you do not need the higher speeds then operating at 100 kHz is the safest choice.

Example

As an example, the following program will read and display the current time (hours and minutes) maintained by a PCF8563 real time clock chip connected to I²C channel 2:

```
DIM AS INTEGER RData(2)         ' this will hold received data
I2C2 OPEN 100, 1000             ' open the I2C channel
I2C2 WRITE &H51, 0, 1, 3        ' set the first register to 3
I2C2 READ &H51, 0, 2, RData()   ' read two registers
I2C2 CLOSE                      ' close the I2C channel
PRINT "Time is " RData(1) ":" RData(0)
```

Appendix C
1-Wire Communications

The 1-Wire protocol was developed by Dallas Semiconductor to communicate with chips using a single signalling line. This implementation was written for MMBasic by Gerard Sexton.

There are three commands that you can use:

ONEWIRE RESET pin	Reset the 1-Wire bus
ONEWIRE WRITE pin, flag, length, data [, data…]	Send a number of bytes
ONEWIRE READ pin, flag, length, data [, data…]	Get a number of bytes

Where:

pin - The I/O pin (located in the rear connector) to use. It can be any pin capable of digital I/O.

flag - A combination of the following options:

1 - Send reset before command

2 - Send reset after command

4 - Only send/recv a bit instead of a byte of data

8 - Invoke a strong pullup after the command (the pin will be set high and open drain disabled)

length - Length of data to send or receive

data - Data to send or variable to receive.
The number of data items must agree with the length parameter.

And the automatic variable

MM.ONEWIRE	Returns true if a device was found

After the command is executed, the I/O pin will be set to the not configured state unless flag option 8 is used.

When a reset is requested the automatic variable MM.ONEWIRE will return true if a device was found. This will occur with the ONEWIRE RESET command and the ONEWIRE READ and ONEWIRE WRITE commands if a reset was requested (flag = 1 or 2).

The 1-Wire protocol is often used in communicating with the DS18B20 temperature measuring sensor and to help in that regard MMBasic includes the TEMPR() function which provides a convenient method of directly reading the temperature of a DS18B20 without using these functions.

Appendix D
SPI Communications

The Serial Peripheral Interface (SPI) communications protocol is used to send and receive data between integrated circuits. The command SPI refers to channel 1 and SPI2 refers to channel 2. SPI2 is not listed below however it is available on the Colour Maximite 2 and has an identical syntax.

I/O Pins

The SPI OPEN command will automatically configure the relevant I/O pins on the rear I/O connector (listed at the start of this manual). MISO stands for Master In Slave Out and because the Colour Maximite 2 is always the master that pin will be configured as an input. Similarly MOSI stands for Master Out Slave In and that pin will be configured as an output.

When the SPI CLOSE command is used these pins will be returned to a "not configured" state. They can then be configured as per normal using SETPIN.

SPI Open

To use the SPI function the SPI channel must be first opened. The syntax for opening the SPI channel is:

```
SPI OPEN speed, mode, bits
```

Where:

- ‘speed’ is the speed of the clock. This can be 25000000, 12500000, 6250000, 3125000, 1562500, 781250, 390625 or 195315 (ie, 25MHz, 12.5MHz, 6.25MHz, 3.125MHz, 1562.5KHz, 781.25KHz, 390.625KHz or 195.3125KHz). For any other values the firmware will select the next valid speed that is equal or slower than the speed requested.
- 'mode' is a single numeric digit representing the transmission mode – see Transmission Format below.
- 'bits' is the number of bits to send/receive. This can be 8, 16 or 32.
- It is the responsibility of the program to separately manipulate the CS (chip select) pin if required.

Transmission Format

The most significant bit is sent and received first. The format of the transmission can be specified by the 'mode' as shown below. Mode 0 is the most common format.

Mode	Description	CPOL	CPHA
0	Clock is active high, data is captured on the rising edge and output on the falling edge	0	0
1	Clock is active high, data is captured on the falling edge and output on the rising edge	0	1
2	Clock is active low, data is captured on the falling edge and output on the rising edge	1	0
3	Clock is active low, data is captured on the rising edge and output on the falling edge	1	1

For a more complete explanation see: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Standard Send/Receive

When the SPI channel is open data can be sent and received using the SPI function. The syntax is:

```
received_data = SPI(data_to_send)
```

Note that a single SPI transaction will send data while simultaneously receiving data from the slave. ‘data_to_send’ is the data to send and the function will return the data received during the transaction. ‘data_to_send’ can be an integer or a floating point variable or a constant.

If you do not want to send any data (ie, you wish to receive only) any number (eg, zero) can be used for the data to send. Similarly if you do not want to use the data received it can be assigned to a variable and ignored.

Bulk Send/Receive

Data can also be sent in bulk:

```
SPI WRITE nbr, data1, data2, data3, … etc
```

or

```
SPI WRITE nbr, string$
```

or

```
SPI WRITE nbr, array()
```

In the first method 'nbr' is the number of data items to send and the data is the expressions in the argument list (ie, 'data1', data2' etc). The data can be an integer or a floating point variable or a constant.

In the second or third method listed above the data to be sent is contained in the 'string$' or the contents of 'array()' (which must be a single dimension array of integer or floating point numbers). The string length, or the size of the array must be the same or greater than nbr. Any data returned from the slave is discarded.

Data can also be received in bulk:

```
SPI READ nbr, array()
```

Where 'nbr' is the number of data items to be received and array() is a single dimension integer array where the received data items will be saved. This command sends zeros while reading the data from the slave.

SPI Close

If required the SPI channel can be closed as follows (the I/O pins will be set to inactive):

```
SPI CLOSE
```

Examples

The following example shows how to use the SPI port for general I/O. It will send a command 80 (hex) and receive two bytes from the slave SPI device using the standard send/receive function:

```
PIN(10) = 1 : SETPIN 10, DOUT     ' pin 10 will be used as the enable signal
SPI OPEN 5000000, 3, 8            ' speed is 5MHz and the data size is 8 bits
PIN(10) = 0                       ' assert the enable line (active low)
junk = SPI(&H80)                  ' send the command and ignore the return
byte1 = SPI(0)                    ' get the first byte from the slave
byte2 = SPI(0)                    ' get the second byte from the slave
PIN(10) = 1                       ' deselect the slave
SPI CLOSE                         ' and close the channel
```

The following is similar to the example given above but this time the transfer is made using the bulk send/receive commands:

```
OPTION BASE 1                     ' our array will start with the index 1
DIM data%(2)                      ' define the array for receiving the data
PIN(10) = 1 : SETPIN 10, DOUT     ' pin 10 will be used as the enable signal
SPI OPEN 5000000, 3, 8            ' speed is 5MHz, 8 bits data
PIN(10) = 0                       ' assert the enable line (active low)
SPI WRITE 1, &H80                 ' send the command
SPI READ 2, data%()               ' get two bytes from the slave
PIN(10) = 1                       ' deselect the slave
SPI CLOSE                         ' and close the channel
```

Appendix E
Sprites

The concept of the sprite implementation is as follows:

- Sprites are full colour and of any size. The collision boundary is the enclosing rectangle.
- Sprites are loaded to a specific number (1 to 64).
- Sprites are displayed using the SPRITE SHOW command.
- For each SHOW command the user must select a "layer". This can be between 0 and 10.
- Sprites collide with sprites on the same layer, layer 0, or the screen edge.
- Layer 0 is a special case and sprites on all other layers will collide with it.
- The SCROLL commands leave sprites on all layers except layer 0 unmoved.
- Layer 0 sprites scroll with the background and this can cause collisions.
- There is no practical limit on the number of collisions caused by SHOW or SCROLL commands.
- The SPRITE() function allows the user to fully interrogate the details of a collision.
- A SHOW command will overwrite the details of any previous collisions for that sprite.
- A SCROLL command will overwrite details of previous collisions for ALL sprites.
- To restore a screen to a previous state sprites should be removed in the opposite order to which they were written (ie, last in first out).

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. Eg:

```
SPRITE INTERRUPT collision
```

The following is an example program 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 = &HF1 THEN
      PRINT "collision with left of screen"
    ELSE IF j = &HF2 THEN
      PRINT "collision with top of screen"
    ELSE IF j = &HF4 THEN
      PRINT "collision with right of screen"
    ELSE IF j = &HF8 THEN
      PRINT "collision with bottom of screen"
    ELSE
      ' SPRITE(C, #n, #m) returns details of the mth collision
      PRINT "Collision with sprite ", SPRITE(C, S, i)
    ENDIF
  NEXT i
END SUB
```

Appendix F
Special Keyboard Keys

MMBasic generates a single unique character for the function keys and other special keys on the keyboard. These are shown in this table as hexadecimal and decimal numbers:

Keyboard Key	Key Code (Hex)	Key Code (Decimal)
DEL	7F	127
Up Arrow	80	128
Down Arrow	81	129
Left Arrow	82	130
Right Arrow	83	131
Insert	84	132
Home	86	134
End	87	135
Page Up	88	136
Page Down	89	137
Alt	8B	139
F1	91	145
F2	92	146
F3	93	147
F4	94	148
F5	95	149
F6	96	150
F7	97	151
F8	98	152
F9	99	153
F10	9A	154
F11	9B	155
F12	9C	156
PrtScr/SysRq	9D	157
PAUSE/BREAK	9E	158
SHIFT_TAB	9F	159
SHIFT_DEL	A0	160
SHIFT_DOWN_ARROW	A1	161
SHIFT_RIGHT_ARROW	A3	163

If the shift key is simultaneously pressed with the function keys F1 to F12 then 40 (hex) is added to the code (this is the equivalent of setting bit 6). For example Shift-F10 will generate DA (hex).

The shift modifier works with the function keys F1 to F12; it is ignored for the other keys except TAB, DEL, DOWN_ARROW, and RIGHT_ARROW as identified above.

MMBasic will translate most VT100 escape codes generated by terminal emulators such as Tera Term and Putty to these codes (excluding the shift and control modifiers). This means that a terminal emulator operating over a USB or a serial port opened as console will generate the same key codes as a directly attached keyboard.

Appendix G
Loading the Firmware

The ARM Cortex-M7 processor includes its own programmer/bootloader so the Colour Maximite 2 firmware can be easily loaded via USB using a personal computer or laptop (special hardware is not needed). Just follow these steps. Note that the same firmware file will work on either the Generation 1 or 2 versions of the Colour Maximite 2 (the hardware is automatically detected).

If you have a version using the Waveshare CPU board:

- Plug the Waveshare CPU board into its position on the motherboard.
- Remove all the jumpers on the Waveshare CPU board.
- Set the power switch to "5VIN"
- Set the BOOT CONFIG switch to "SYSTEM"

If you have a version without the Waveshare module (ie, the Generation 2 desugn) the programming function is provided by a set of three jumper pins labelled “PGM” and “RUN”. Place a jumper on the common and “PGM” pins.

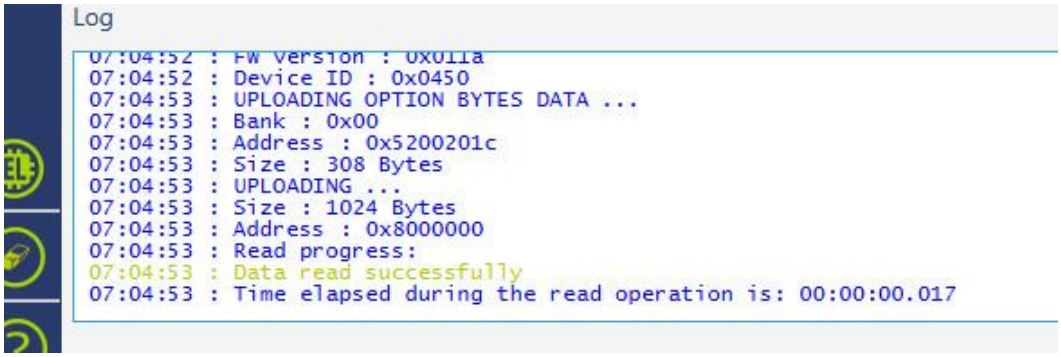
Make sure that the Type-B USB power cable to the Colour Maximite is disconnected.

Using a USB Type-A to Type-A cable connect the USB Keyboard port on the Colour Maximite 2 to a USB port on your desktop computer. This will power up the Colour Maximite regardless of the position of the power switch. You should also hear a sound from your computer as it connects to the Colour Maximite.

Go to https://www.st.com/en/development-tools/stm32cubeprog.html
and download the STM32CubeProgrammer software. This is free software but STM do require you to have an STM account or provide your name and email address. They will email you a link to download the software. Then install this software on your computer (Windows, Linux and macOS are supported).

Run the STM32CubeProgrammer software on your computer. On the top right of the program window select USB as the communications method. If the program does not recognise the USB connection click on the small blue circle to the right of the Port drop down list to refresh the entry. Your screen should look like the illustration on the right (the USB port number may vary).

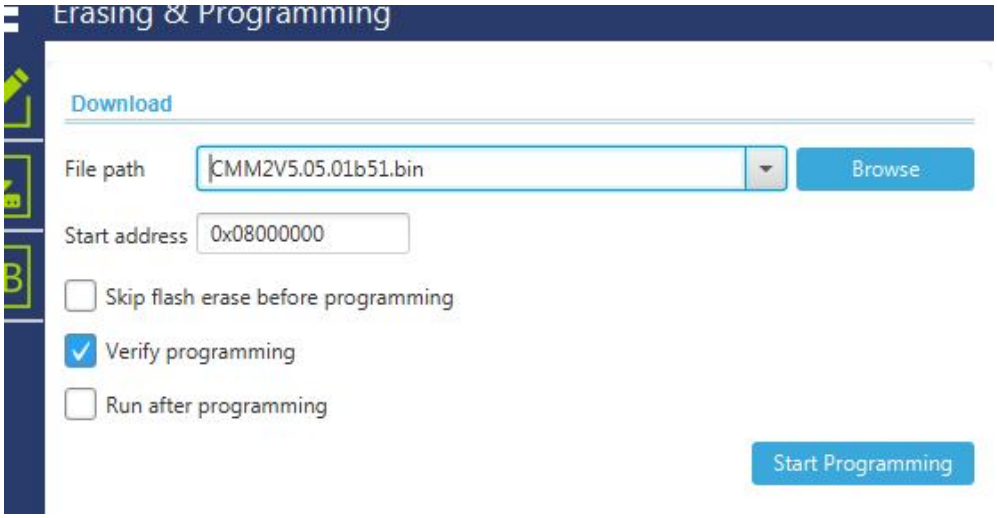
Click on the "Connect" button. You should then see a series of messages as shown in the screenshot below finishing with the message "Data read successfully". Any messages in red will indicate an error.



Click on the download button ( ) on the left side of the STM32CubeProgrammer window and the software will switch to the "Erasing and Programming" mode as shown below.

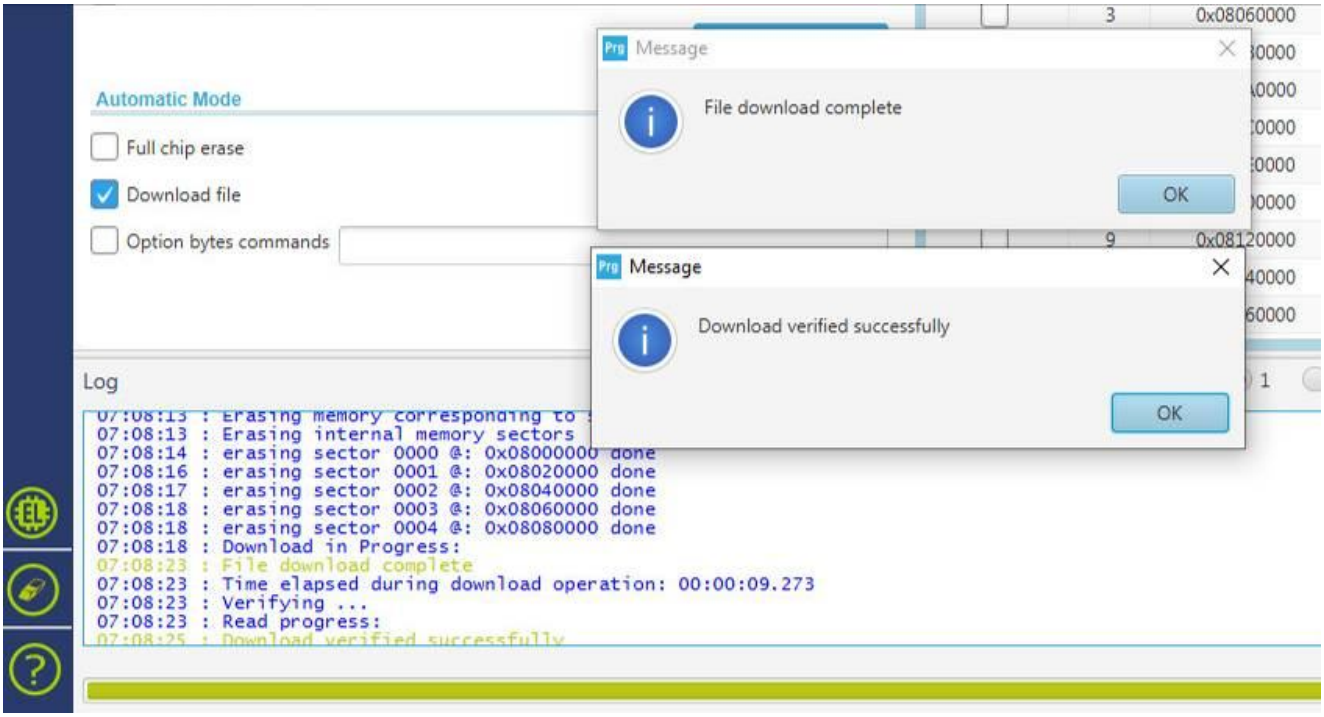
Use the "Browse button" to select the firmware file (it will have an extension of .bin).
Tick the "Verify programming" checkbox.

Finally, click on the "Start Programming" button.



The STM32CubeProgrammer software will then program the firmware into the flash memory on the ARM Cortex-M7 in the Colour Maximite 2 (the STM32CubeProgrammer software calls this "downloading"). After a short time a dialog box will pop up saying that "File download completed". Do not do anything at this point as the software will then start reading back the firmware programmed into the flash. When this has completed successfully another dialog box will pop up saying "Download verified successfully" as shown below.

The whole operation will take under a minute and any messages in red will indicate an error.



Then:

- Dismiss all the dialog boxes and close the STM32CubeProgrammer software.
- On the Waveshare module set the BOOT CONFIG switch to "Flash" or on versions without the Waveshare module the jumper should be between the common and the "RUN" pins.
- Remove the USB Type-A to Type-A cable from the USB Keyboard port.
- Plug in the VGA monitor and USB keyboard.
- Plug the Colour Maximite into power and set the front panel power switch to ON.

On power up you should now see on the VGA monitor the Maximite logo and the version number of the firmware that you have just loaded.

If your monitor remains blank check that you are not still in programming mode (ie, reset the BOOT CONFIG switch or the programming jumper). If you do not have a VGA monitor you can use the serial console (over USB) to check the firmware installation. This is described in the section *Hardware Features* and *Serial Console* in this manual.

When MMBasic is first loaded it will prompt for the keyboard type, the screen type and the date/time. On subsequent firmware upgrades MMBasic will preserve these settings (in addition to OPTION RTC CALIBRATE) and will not prompt for them again. These can be changed later using the relevant OPTION commands.

If you wish to load another version of the firmware (either earlier or later) this can be done by repeating the steps above.

Alternative Method

An alternative method of loading the firmware is to use serial transfer over USB. To do this you must first install the serial over USB driver on your desktop computer as described in the section *Hardware Features* and *Serial Console* in this manual.

If you have a version using the Waveshare CPU board:

- Remove all the jumpers on the Waveshare CPU board.
- Set the power switch to "5VIN"
- Set the BOOT CONFIG switch to "System"

If your version does not use the Waveshare CPU board:

- Place a jumper between the common and "PGM" pins on the mainboard.

Then:

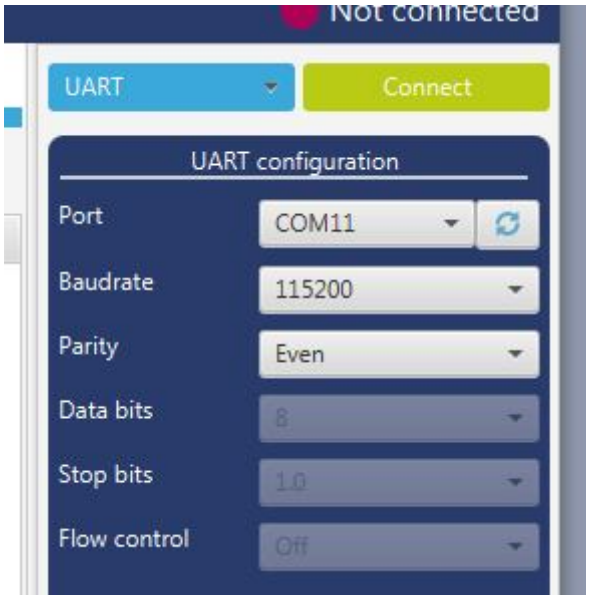
- Connect your desktop computer to the USB Type-B connector on the Colour Maximite.
- Set the front panel power switch to ON and/or press the reset button on the Waveshare CPU board.

This should power up the Colour Maximite which will then connect to your desktop computer via USB.

In Windows the connection will appear in Device Manager as "USB Serial Port" as illustrated on the right (the COM number will probably be different):

Install the STM32CubeProgrammer software on your computer as described above.

Run the software and select UART in the top right corner (as illustrated on the right). Then select the correct COM port number as reported in the Windows Device Manager. Finally make sure that the baudrate is set to 115200 baud and the parity set to even:



From then on the process is the same as that described above when using a direct USB connection via the keyboard port:

- Click on "Connect".
- Select "Erase & Programming" mode.
- Browse for the firmware file.
- Tick the "Verify programming" checkbox.
- Click on "Program".

The whole operation will take about 5 minutes.

Note that after the initial firmware has been loaded you can use the UPDATE FIRMWARE command to place the ARM Cortex-M7 into download mode (equivalent to setting the BOOT CONFIG switch to "SYSTEM or jumper to “PGM”) thereby avoiding opening the case.

When the programming/verify has completed set the BOOT CONFIG switch to "Flash" on the CPU board (or return the jumpers to the “RUN” position) and press the RESET button.

Third Method

If you have a version using the Waveshare board It is also possible to program the firmware using the micro USB connector on this board. This method is not recommended as the Waveshare module must be removed to access this connector and doing this too many times will inevitably damage the module’s pins. However, this is a handy method of testing the Waveshare board - if you can load the firmware without error it is a good indication that the module is working correctly.

To use this method remove the Waveshare CPU module from the motherboard, place shorting jumpers on all header pins except PA9-VBUS, set the power switch to "USB” and the BOOT CONFIG switch to "SYSTEM". Plug a USB cable into the micro USB connector on the top of the module and the other end into your desktop computer - both LEDs on the module should illuminate and it should connect to your computer.

Then, using the steps listed for loading the firmware via the USB keyboard port, load and verify the Colour Maximite 2 firmware using this USB cable and your desktop computer.

Linux and the Raspberry Pi

Loading the firmware from a Linux computer and/or the Raspberry Pi has some special considerations and these are explained here: http://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=12171