MMBasic

Language Manual

Ver 4.0

Geoff Graham

For updates to this manual and more details on MMBasic
go to http://mmbasic.com

or http://geoffg.net/maximite.html

Copyright 2011, 2012 Geoff Graham

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

MMBasic is a Microsoft BASIC compatible implementation of the BASIC language with floating point and string variables, long variable names, arrays of floats or strings with multiple dimensions and powerful string handling.

MMBasic was originally written for the Maximite, a small computer based on the PIC32 microcontroller from Microchip. It now runs on a variety of hardware platforms including DOS.

This manual describes the MMBasic language. For details of running MMBasic on specific platforms please refer to the following documentation:

All Maximite Computers:	Maximite Hardware Manual	from: http://geoffg.net/maximite.html
UBW32 experimenter board:	UBW32 MMBasic User Manual	from: http://geoffg.net/ubw32.html
CGMMSTICK1 board:	http://www.circuitgizmos.com/products/cgmmstick1/cgmmstick1.shtml	
DuinoMite series:	DuinoMite MMBasic ReadMe	included with the DuinoMite update.
DOS:	DOS MMBasic ReadMe	from: http://mmbasic.com/downloads.html

Throughout this manual Maximite or MM refers to the Maximite family (Maximite, Colour Maximite, mini Maximite, UBW32, CGMMSTICK1, DuinoMite and others that run the standard Maximite firmware). DOS refers to the version that runs in a DOS box under Windows.

Contents

Functional Summary

Command and Program Input

At the prompt (the greater than symbol, ie, >) you can enter a command line followed by the enter key and it will be immediately run. This is useful for testing commands and their effects.

Line numbers are optional. If you use them you can enter a program at the command line by preceding each program line with a line number however; it is recommended that the full screen editor (the EDIT command) be used to enter and edit programs.

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 arrow key will move through a list of previously entered commands which can be edited and reused.

A program held in memory can be listed with LIST, run using the RUN command and cleared with the NEW command. You can interrupt MMBasic at any time by typing CTRL C and control will be returned to the prompt.

Keyboard/Display

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

Output will be simultaneously sent to the USB interface and the video display (VGA or composite). Either can be attached or removed at any time.

Line Numbers, Program Structure and Editing

In version 3.0 and later the use of line numbers is optional. MMBasic will still run programs written using line numbers, but it is recommended that new programs avoid them.

The structure of a program line is:

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

Instead of using a line number a label 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 instead of 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"
```

MMBasic finds a label much faster than a line number so labels are recommended for new programs.

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

Long programs (with or without line numbers) can be sent via USB to MMBasic using the XMODEM command (Maximite only) or the AUTO command.

Program and Data Storage

In DOS the drive letters are as supported by Windows. On the Maximite and Colour Maximite two “drives” are available for storing and loading programs and data:

- Drive “A:” is a virtual drive using the PIC32’s internal flash memory and has a size of 212KB.
- Drive “B:” is the SD card (if connected). It supports MMC, SD or SDHC memory cards formatted as FAT16 or FAT32 with capacities up to the largest that you can purchase.

File names must be in 8.3 format prefixed with an optional drive prefix A: or B: (the same as DOS or Windows). Long file names and directories are not supported. The default drive is B: and this can be changed with the DRIVE command.

On the Maximite MMBasic will look for a file on startup called “AUTORUN.BAS” in the root directory of the internal flash drive (A:) then the SD card (B:). If the file is found it will be automatically loaded and run, otherwise MMBasic will print a prompt (“>”) and wait for input.

Note that the video output will go blank for a short time while writing data to the internal flash drive A:. This is normal and is caused by a requirement to shut off the video while reprogramming the memory. When using drive A: you need to be careful not to wear out the flash (the same applies to SD cards). If drive A: is empty, you could write and delete a file on it every day for 175 years before you would reach the endurance limit - but if the interval was once a minute you would reach the limit in about 6 weeks.

Storage Commands and Functions

A program can be saved to either drive using the SAVE command. It can be reloaded using LOAD or merged with the current program using MERGE. A saved program can also be loaded and run using the RUN command. The RUN command can also be used within a running program, which enables one program to load and transfer control to another.

Data files can be opened using OPEN and read from using INPUT, LINE INPUT, or INPUT$() or written to using PRINT or WRITE. On the SD card both data and programs are stored using standard text and can be read and edited in Windows, Apple Mac, Linux, etc. An SD card can have up to 10 files open simultaneously while the internal flash drive has a maximum of one file open at a time.

You can list the programs stored on a drive with the FILES command, delete them using KILL and rename them using NAME. On an SD card the current working directory can be changed using CHDIR. A new directory can be created with MKDIR or an old one deleted with RMDIR.

Whenever specified a file name can be a string constant (ie, enclosed in double quotes) or a string variable. This means you must use double quotes if you are directly specifying a file name. Eg, RUN “TEST.BAS”

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 with the optional battery backed clock will never loose the time, on other Maximites the calendar will start from midnight 1st Jan 2000 on power up. On the DOS version it will use the system time.

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 milliseconds. You can reset the timer to zero or any other number by assigning a value to the TIMER.

Using SETTICK in the Maximite versions you can setup a “tick” which will generate a regular interrupt with a period from one millisecond to over a month. See Interrupts below.

Expressions

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

FNAME$ = “TEST”: RUN FNAME$ + ”.BAS”

Structured Statements

MMBasic supports a number of modern structured statements.

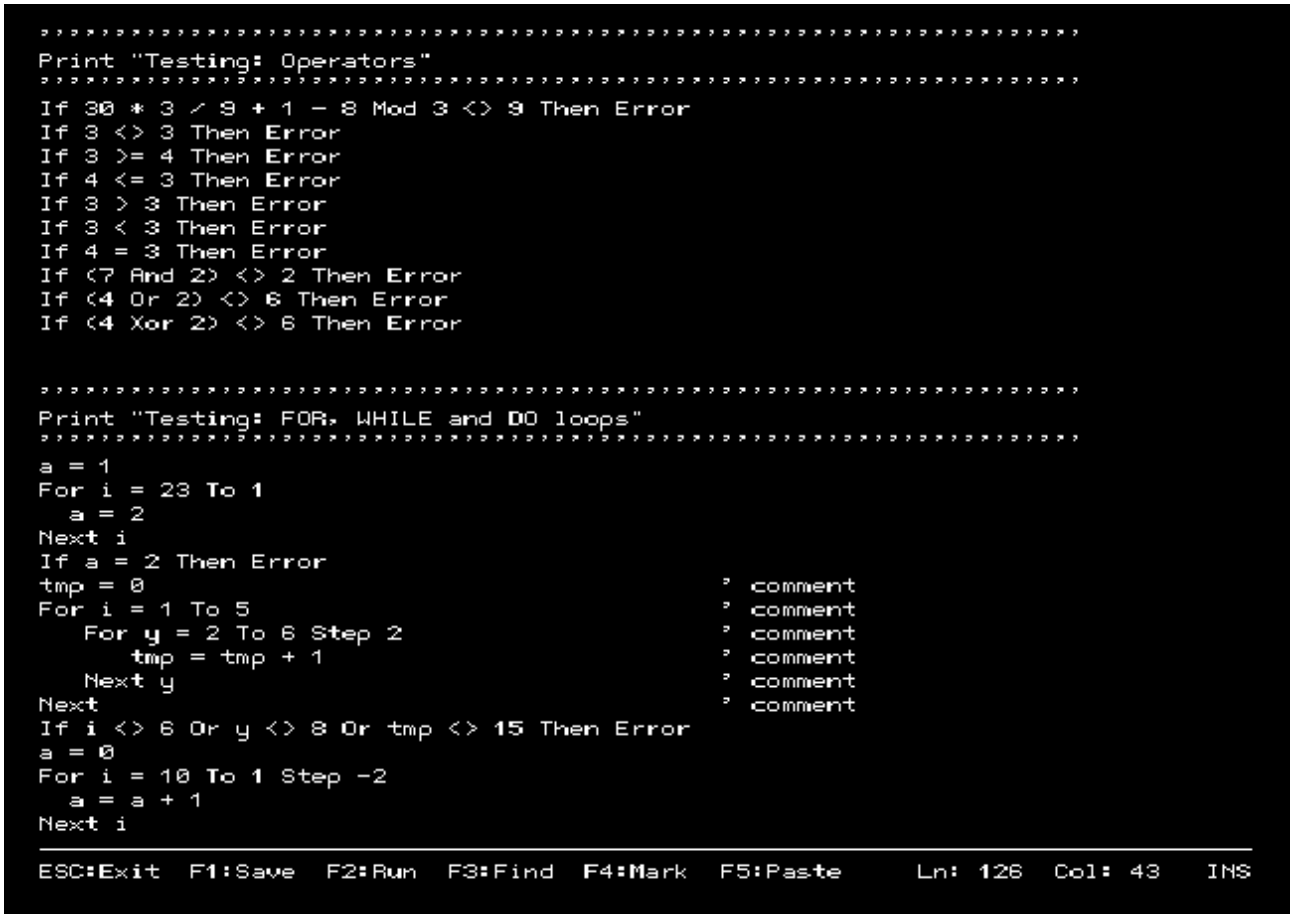
The DO WHILE … LOOP command and its variants make it easy to build loops without using the GOTO statement. Defined subroutines and functions make it easy to add your own commands to MMBasic.

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

```
IF <condition> THEN              ' start a multiline IF
  <statements>
ELSEIF <condition> THEN          ' the ELSEIF is optional
  <statements>
ELSE                             ' the ELSE is optional
  <statements>
ENDIF                            ' must be used to terminate the IF
```

Full Screen Editor

An important productivity feature of MMBasic is the full screen editor (this is not available in the DOS version of MMBasic). It will work using an attached video screen (VGA or composite) and over USB with a VT100 compatible terminal emulator (Tera Term is recommended).



The full screen editor is invoked with the EDIT command. If you just type EDIT without anything else the editor will automatically start editing whatever is in program memory. If program memory is empty you will be presented with an empty screen.

The cursor will be automatically positioned at the last place that you were editing at and, if your program had just been stopped by an error, the cursor will be positioned at the line that caused the error.

You can also place a file name on the command line with the RUN command and in that case, the editor will edit that file and leave program memory untouched. This is handy for examining or changing files on the disk without disturbing your program.

If you are used to an editor like Notepad you will find that the operation of the full screen editor is familiar. The arrow keys will move your 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.

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 will list the various function keys used by the editor and their action. In more details these are:

ESC	This will cause the editor to abandon all changes and return to the command prompt with the program memory unchanged. If you have changed the text you will be asked if this is really what you want to do.

F1: SAVE	This will save the program to program memory and return to the command prompt. If you are editing a disk file it will save that file to the disk.
F2: RUN	This will save the program to program memory and immediately run it.
F3: FIND	This will prompt for the text that you want to search for. When you press enter the cursor will be placed at the start of the first entry found.
SHIFT-F3	Once you have used the search function once you can repeatedly search for the same text by pressing SHIFT-F3.
F4: MARK	This is described in detail below.
F5: PASTE	This will insert (at the current cursor position) the text that had been previously cut or copied (see below).

If you pressed the mark key (F4) the editor will change to the *mark mode*. In this mode you can use the arrow keys to mark a section of text which will be highlighted in reverse video. You can then delete, cut or copy the marked text. In this mode the status line will change to show the functions of the function keys in the mark mode. These keys are:

ESC	Will exit mark mode without changing anything.
F4: CUT	Will copy the marked text to the clipboard and remove it from the text.
F5: COPY	Will just copy the marked text to the clipboard.
DELETE	Will delete the text leaving the clipboard unchanged.

The best way to learn the full screen editor is to simply fire it up and experiment.

The editor is a very productive method of writing a program. You can program a function key to run the editor. So, with one key press you are into the editor where you can make a change. Then by pressing the F3 key you will save and run the program. If your program stops with an error you can press the edit key and be back in the editor with the cursor positioned at the line that caused the error. This edit/run/edit cycle is very fast.

If you are using the full screen editor over USB with Terra Term you must set Terra Term to a screen size of 80 characters by 36 lines. See Appendix G for details.

Input/Output

The following functions are only supported on the Maximite variants (not on the DOS version).

External Input/Output

You can configure an external I/O pin using the SETPIN command, set its output using the PIN()= command and read the current input value using the PIN() function. Digital I/O uses the number zero to represent a low voltage and any non-zero number for a high voltage. An analogue input will report the measured voltage as a floating point number.

The original Maximite has 20 I/O pins numbered 1 to 20. Pins 1 to 10 can be used for analog input and digital input/output with a maximum input voltage of 3.3V. Pins 11 to 20 are digital only but support input voltages up to 5V and can be set to open collector.

The DuinoMite has completely different and confusing allocations. See "DuinoMite MMBasic ReadMe.pdf"

Normally digital output is 0V (low) to 3.3V (high) but you can use open collector to drive 5V circuit. This means that the pin can be pulled down (when the output is low) but will go high impedance when the output is high. So, with a pull up resistor to 5V an output configured as open collector you can drive 5V logic signals. Typical value of the pull up resistor is 1K to 4.7K.

Arduino Connector

In addition to the 20 I/O pins described above the Colour Maximite has an extra 20 I/O pins on the Arduino compatible connector (40 I/O pins in total). These are labelled D0 to D13 and A0 to A5.

You can use the labels D0, D1, etc in the SETPIN and PIN statements or you can use their corresponding numbers (D0 = 21, D1 = 22, etc and A0 = 35, A1 = 36, etc). The digital pins (D0 to D13) have the same characteristics (5V, open collector, etc) as the digital pins 11 to 20 and the analog capable pins (A0 to A5) have the same capabilities as pins 1 to 10.

Communications

Two serial ports are supported with speeds up to 19200 baud with configurable buffer sizes and optional hardware flow control. The serial ports are opened using the OPEN command and any command or function that uses a file number can be used to send and receive data. See Appendix A for a full description.

Communications to slave or master devices on an I^2C bus is supported with eight commands (see Appendix B for a full description). MMBasic fully supports bus master and slave mode, 10 bit addressing, address masking and general call, as well as bus arbitration (ie, bus collisions in a multi-master environment).

The Serial Peripheral Interface (SPI) communications protocol is supported with the SPI command. See Appendix D for the details. The Dallas 1-Wire protocol is also supported. See Appendix C for details.

Interrupts

Any external I/O pin can be configured to generate an interrupt using the SETPIN command with up to 29 interrupts (including the tick interrupt) active at any one time. Interrupts can be set up to occur on a rising or falling digital input signal and will cause an immediate branch to a specified line number or label (similar to a GOSUB). The target can be the same or different for each interrupt. Return from an interrupt is via the IRETURN statement. All statements (including GOSUB/RETURN) can be used within an interrupt.

If two or more interrupts occur at the same time they will be processed in order of pin numbers (ie, an interrupt on pin 1 will have the highest priority). During processing of an interrupt all other interrupts are disabled until the interrupt routine returns with an IRETURN. During an interrupt (and at all times) the value of the interrupt pin can be accessed using the PIN() function.

A periodic interrupt (or regular “tick”) with a period specified in milliseconds can be setup using the SETTICK statement. This interrupt has the lowest priority.

Interrupts can occur at any time but they are disabled during INPUT statements. If you need to get input from the keyboard while still accepting interrupts you should use the INKEY$ function. 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.

For most programs MMBasic will respond to an interrupt in under 100µS. To prevent slowing the main program by too much an interrupt should be short and execute the IRETURN statement as soon as possible. Also remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non-intuitive bugs.

Audio and PWM Output

On the Maximite variants there are a number of ways that you can use the sound output. You can play synthesised music, generate tones or generate program controlled voltages (PWM).

PLAYMOD

This command will play synthesised music in the background while the program is running. The music must be in the MOD format and the file containing the music must be located on the internal flash drive (drive A:). The audio is high quality and MMBasic will generate full stereo on the Colour Maximite.

The MOD format is a music file format originating from the MOD file format on Amiga systems in the late 1980s. It is not a recording of the music (like a MP3 file) - instead it contains instructions for synthesising the music. On the original Amiga this task was performed by dedicated hardware.

MMBasic will read this file and continuously play the music in the background while the program that launched the music will continue running in the foreground. Be aware that synthesising music is a CPU intensive activity and uses a lot of memory and this could affect the performance of the program.

A description of the MOD format can be found at: http://en.wikipedia.org/wiki/MOD_(file_format)

A large selection of files that can be played on the Maximite can be found at: http://modarchive.org (look for files with the .MOD extension). Because the file must be located on drive A: to play it would be wise to select reasonably small files.

You can also create your own music using a tracker. For an example see: http://www.modplug.com

TONE

This command will create two tones for the Colour Maximite that will be outputted separately on the left and right sound channels. On the monochrome Maximite only one tone is generated. The tone is a synthesised sine wave and can be in the range of 1Hz to 20KHz with a resolution of 1Hz and is very accurate as it is locked to the PIC32's crystal oscillator. When the frequency is changed there is no interruption in the output so the output can be made to glide smoothly across a range of frequencies.

The playing time can be specified in milliseconds and the tone will play in the background (ie, the program continues running).

SOUND

The sound command is included only for compatibility with older programs. It generates a single frequency square wave and should be replaced with the tone or PWM command in new programs.

PWM

The PWM (Pulse Width Modulation) command allows the Maximite to generate two square waves with programmed controlled duty cycle. By varying the duty cycle you can generate a program controlled voltage outputs for use in controlling external devices that require an analog input (power supplies, motor controllers, etc). The Colour Maximite has two channels while the monochrome Maximite has a single channel.

The frequency for both channels is the same and can be from 20Hz to 1MHz. The duty cycle for each channel can be independently set from between 0% and 100% with a 0.1% resolution.

This command uses the sound output for generating the PWM signal so the components on this output may need to be changed to allow this output to work as a PWM output.

Graphics and Working with Colour

Graphics

Graphics commands operate on the video output only (not USB). Coordinates are measured in pixels with x being the horizontal coordinate and y the vertical coordinate. The top left of the screen is at location x = 0 and y = 0, and the bottom right of the screen defined by the read-only variables x = MM.HRES and y = MM.VRES which change depending on the video mode selected (VGA or composite). Increasing positive numbers represent movement down the screen and to the right.

You can clear the screen with CLS and an individual pixel can be turned on or off and its colour set with PIXEL(x,y) = . You can draw lines and boxes with LINE, and circles using CIRCLE. You can also set the screen location (in pixels) of the PRINT output using @(x,y) and the SAVEBMP command will save the current screen as a BMP file. LOADBMP will load and display a bitmap image stored on the SD card.

Working with Colour

The Colour Maximite supports eight colours (black, blue, green, cyan, red, purple, yellow and white). The monochrome Maximite or DuinoMite support just two (black and white). In most places you can also specify the colour as -1 to invert a pixel (this is useful in animation).

Throughout MMBasic you can refer to the colours by their name or their corresponding numbers where black = 0, blue = 1, green = 2, etc through to white = 7. Commands such as LINE and CIRCLE use this colour or number to specify the colour to draw. For example:

```
CIRCLE (100, 100), 50, CYAN
```
will draw a circle in cyan.

```
CIRCLE (100, 100), 50, 3
```
will also draw a circle in cyan (colour = 3).

You can also specify a default colour that will be used for all screen output with the COLOUR command. For example: `COLOUR PURPLE` will set the colour of text to purple (and any other output where the colour is not specified). The COLOUR command also takes a second parameter for the background colour. So, `COLOUR YELLOW, BLUE` specifies that text will be displayed in yellow on a blue background.

In addition to the COLOUR command you can embed colour codes into text strings to change the colour of the text. For example, the following will display each word in a different colour:

```
Txt$ = "This is " + chr$(130) + "Red " + chr$(134) + "Yellow"
PRINT Txt$
```

The embedded character is the number 128 plus the colour number. So, red is 128+2 or 130 and yellow is 128+6 or 134. You can also set the background colour in a similar way using the number 192 plus the colour number. For example 196 will set the background to red. You can also use the colour name to get the colour number so this alternative example will print yellow letters on a red background:

```
PRINT CHR$(128+YELLOW) CHR$(192+RED) " ALARM "
```

The colours are reset to the defaults (set by the COLOUR command) when the print command terminates.

Colour Modes

The video system can be configured into one of four modes using the MODE command. This enables the programmer to trade off the number of colours used on the screen and the graphic resolution against the amount of memory required by the video driver. Modes 1 and 4 use the least amount of memory while mode 3 uses the most. The syntax of the MODE command is: MODE colour-mode, palette

The 'colour-mode' can be one of four numbers:

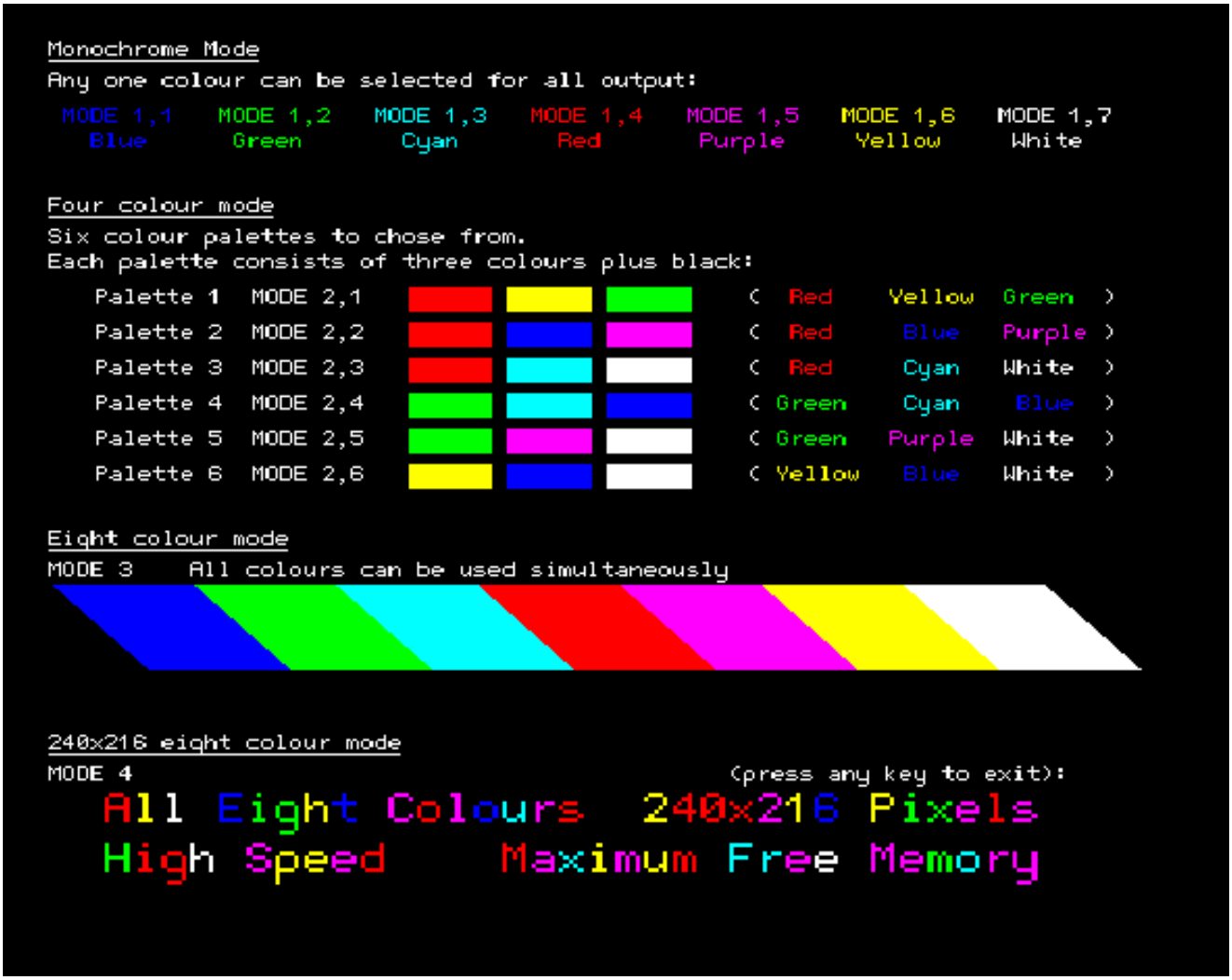
1 Monochrome mode. In this mode the Colour MMBasic operates the same as the monochrome MMBasic for the Maximite and has the maximum amount of free memory available for programs and data. The second argument of the MODE command ('palette') selects the colour to be used for all output. It can be any colour number from black to white.

2 Four colour mode. In this mode four colours (including black) are available. The actual colours are selected by a number (1 to 6) used in the second argument of the MODE command ('palette'). See the following image or the MODE command for a listing of the actual colours available.

3 Eight colour mode. In this mode all eight colours are available and can be used simultaneously anywhere on the screen. The 'palette' argument is not required and will be ignored if specified. MODE 3 uses the most memory but there still is plenty left for programs and data. This is the default when the Colour Maximite is first powered up.

4 240x216 pixel mode. In this mode all eight colours are available and the video resolution is halved (meaning that characters and graphics are doubled in size). This mode is most suitable for games as all colours are available, it has the maximum amount of free memory and drawing of graphics is very fast. The 'palette' argument is not required and will be ignored if specified.

This is an example of what can be selected in all four colour modes:



You can change the mode and the palette at any time and as often as you need, even within a running program.

Scan Line Colour Override

In mode 1 (monochrome) there is an additional facility to change the colour of each horizontal line of pixels on the screen using the SCANLINE command. This is intended mostly for programmers writing games and provides limited control over colour while still providing the maximum amount of free memory. The syntax is:

SCANLINE colour, startline, endline

This command can only be used in MODE 1, 7 (monochrome with the colour set to white) and is used to set the colour for each horizontal scan line of pixels on the screen. 'colour' is the colour to be used and can be any one of the eight colours, 'startline' is the starting scan line to be set to that colour and 'endline' is the ending line. The scan lines are numbered from 0 at the top of the screen to 431 at the bottom of the screen. The numbering is the same as that used when specifying the vertical coordinates of a pixel.

You can use multiple SCANLINE commands to set multiple scan lines to different colours. For example:

```
SCANLINE RED, 0, 9          ' set the top 10 lines to red
SCANLINE YELLOW, 120        ' and set only line 120 to yellow
SCANLINE BLUE, 200, 219     ' and set a band of 20 lines to blue
```

To turn off the override imposed by the use of SCANLINE commands you can use the MODE command to reselect mode 1 or a change to a different mode. It is also automatically turned off when control is returned to the command prompt.

Game Playing Features

MMBasic 4.x introduces a number of features that are intended to make it easier to write games on the Maximite.

MODE 4

The colour MODE 4 described in the previous section is mostly intended for games. It provides eight colours and leaves plenty of free memory for the other aspects of an animated game (the program, sprites, arrays, and so on).

Because this colour mode has only one quarter of the pixels the graphics operations are much faster due to the fact that there are fewer pixels that need to be manipulated by MMBasic when drawing on the screen.

BLIT

This command will move an area of the video screen from one location to another. The destination can overlap the source area and the BLIT command will copy the video data correctly to avoid corruption. On the Colour Maximite you can also independently specify what colour planes to copy.

This method of moving video data is much faster than copying pixels one by one and allows for rapid animation on the screen. It can also be used to replicate a pattern like a border or a brick wall to build a complete image.

SPRITE

A sprite is a 16x16 bit graphic image that can be moved about on the screen independently of the background. When the sprite is displayed MMBasic will automatically save the background text and graphics under the sprite and when the sprite is turned off or moved MMBasic will restore the background.

The sprites are defined in a file which is loaded into memory using the SPRITE LOAD command, the number of sprites contained in the file is only limited by the amount of available memory. Each sprite in the file can contain pixels of any colour (on the Colour Maximite) and can also have transparent pixels which allow the background to show through. See Appendix H for a detailed description of creating a sprite file.

To manipulate the sprites you can use the command SPRITE ON which will display a specific sprite at a specified location on the screen. SPRITE MOVE will move a sprite to a new location and restore the background. SPRITE OFF will remove a sprite from the screen and restore the background.

Sprites should not overlap but if they do you should turn them off in the reverse sequence that you turned them on before you turn them on again at their new location. This will enable the background image to be correctly maintained.

For example, the following two sprites overlap:

```
SPRITE ON 1, 100, 150      ' sprite 1 is drawn at x = 100, y = 150
SPRITE ON 2, 110, 160      ' sprite 2 overlaps
```

To move the sprites they need to be turned off in the reverse sequence:

```
SPRITE OFF 2
SPRITE OFF 1
```

Then they can be redrawn at their new location:

```
SPRITE ON 1, 104, 154      ' sprite 1 is drawn at x = 104, y = 154
SPRITE ON 2, 116, 166      ' sprite 2 still overlaps
```

Because sprites are drawn so fast the user is unaware that the sprite has been turned off then redrawn.

LOADBMP and FONTS

The LOADBMP command will load a colour or monochrome bitmap image and display it at a specified location on the screen. This is handy for loading background images for games.

The FONT command can also be used to load custom designed graphic images and display them on the screen.

PEEK/POKE

With the PEEK and POKE commands you can now use constant keywords to access special sections of memory (like the video memory) and these keywords will be valid with future versions of MMBasic. This makes it easy to access internal MMBasic data structures in a portable manner.

Defined Subroutines and Functions

Defined subroutines and functions are useful features to help in organising programs so that they are easy to modify and read. A 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.

For example, assume that you would like to have the command FLASH added to MMBasic, its job would be to flash the power light on the Maximite. You could define a subroutine like this:

```
Sub FLASH
  Pin(0) = 1
  Pause 100
  Pin(0) = 0
End Sub
```

Then, in your program you just use the command FLASH to flash the power LED. For example:

```
IF A <= B THEN FLASH
```

If the FLASH subroutine was in program memory you could even try it out at the command prompt, just like any command in MMBasic. The definition of the FLASH subroutine can be anywhere in the program but typically it is at the start or end. If MMBasic runs into the definition while running your program it will simply skip over it.

Subroutine Arguments

Defined subroutines can have arguments (sometimes called parameter lists). 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 different from arguments defined for the subroutine (at the risk of making debugging harder).

When calling a subroutine you can supply less than the required number of values. For example:

```
MYSUB  23
```

In that case the missing values will be assumed to be either zero or an empty string. For example, in the above case `arg2$` will be set to `""` and `arg3` will be set to zero. This allows you to have optional values and, if the value is not supplied by the caller, you can take some special action.

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 `""`.

Local Variables

Inside a subroutine you will need to use variables for various tasks. In portable code you do not want the name you chose for such a variable to clash with a variable of the same name in the main program. To this end you can define a variable as LOCAL.

For example, this is our FLASH subroutine but this time we have extended it to take an argument (nbr) that specifies how many times to flash the LED.

```
Sub FLASH ( nbr )
  Local count
  For count = 1 To nbr
    Pin(0) = 1
    Pause 100
```

```
    Pin(0) = 0
    Pause 150
  Next count
End Sub
```

The counting variable (`count`) is declared as local which means that (like the argument list) it only exists within the subroutine and will vanish when the subroutine exits. You can have a variable called `count` in your main program and it will be different from the variable `count` in your subroutine.

If you do not declare the variable as local it will be created within your program and be visible in your main program and subroutines, just like a normal variable.

You can define multiple items with the one LOCAL command. If an item is an array the LOCAL command will also dimension the array (ie, you do not need the DIM command). For example:

```
LOCAL NBR, STR$, ARR(10, 10)
```

You can also use local variables in the target for GOSUBs. For example:

```
  GOSUB MySub
       ...
MySub:
   LOCAL X, Y
   FOR X = 1 TO ...
   FOR Y = 5 TO ...
  <statements>
   RETURN
```

The variables X and Y will only be valid until the RETURN statement is reached and will be different from variables with the same name in the main body of the program.

Defined Functions

Defined functions are similar to defined subroutines with the main difference being that the function is used to return a value in an expression. For example, if you wanted a function to select the maximum of two values you could define:

```
Function Max(a, b)
  If a > b
    Max = a
  Else
    Max = b
  EndIf
End Function
```

Then you could use it in an expression:

```
SetPin 1, 1 : SetPin 2, 1
Print "The highest voltage is" Max(Pin(1), Pin(2))
```

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 (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 $ the function will return a string, otherwise it will return a number. Within the function the function's name acts like a standard variable.

As another example, let us say that you need a function to format time in the AM/PM format:

```
Function MyTime$(hours, minutes)
  Local h
  h = hours
  If hours > 12 Then h = h - 12
  MyTime$ = Str$(h) + ":" + Str$(minutes)
  If hours <= 12 Then
    MyTime$ = MyTime$ + "AM"
  Else
    MyTime$ = MyTime$ + "PM"
  EndIf
End Function
```

As you can see, the function name is used as an ordinary local variable inside the subroutine. It is only when the function returns that the value assigned to `MyTime$` is made available to the expression that called it. This example also illustrates that you can use local variables within functions just like subroutines.

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 in your routine 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.

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 will be "magically" changed by your routine. It is much safer to assign the argument to a local variable and manipulate that instead.

Additional Notes

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.

You cannot use arrays in a subroutine or function's argument list however the caller can use them. 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.

The use of defined subroutines and functions should reduce the need to add specialised features to MMBasic. For instance, there have been a few requests to add bit shifting functions to the language. Now you can do that yourself… this is the right shift function:

```
Function RShift(nbr, bits)
   If nbr < 0 or bits < 0 THEN ERROR "Invalid argument"
   RShift = nbr\(2^bits)
End Function
```

You can now use this function as if it is a part of the language:

```
a = &b11101001
b = RShift(a, 3)
```

After running this fragment of code the variable b would have the binary value of `11101.`

The defined subroutine and function is intended to be a portable lump of code that you can insert into any program. This is why the full screen editor has the CTRL-F keys for inserting another file. The idea is that you can keep your defined routines in a file and whenever you need them you can quickly insert them using CTRL-F.

So, it would be easy to create a library of bit manipulation functions like that described above and insert them into any program when needed.

Implementation Details

Naming Conventions

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.

There are two types of variable: numeric which stores a floating point number (eg, 45.386), and string which stores a string of characters (eg, “Tom”). String variable names are terminated with a $ symbol (eg, name$) while numeric variables are not.

Variable names and labels 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. Eg, step = 5 is illegal as STEP is a keyword. In addition, a label cannot be the same as a command name.

Constants

Numerical 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.

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

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

Operators and Precedence

The following operators, in order of precedence, are recognised. 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
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction

Logical operators:

NOT	logical inverse of the value on the right
<> < > <= =< >= =>	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

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

The other logical operations result in the number 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 is highest in precedence so it will bind tightly to the next value. For normal use the expression to be negated should be placed in brackets. For example, 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

Implementation Characteristics

Maximum length of a command line is 255 characters.

Maximum length of a variable name or a label is 32 characters.

Maximum number of dimensions to an array is 8.

Maximum number of arguments to commands that accept a variable number of arguments is 50.

Maximum number of user defined subroutines and functions (combined): 64

Numbers are stored and manipulated as single precision floating point numbers. The maximum number that can be represented is 3.40282347e+38 and the minimum is 1.17549435e-38

The range of integers (whole numbers) that can be manipulated without loss of accuracy is ±16777100.

Maximum string length is 255 characters.

Maximum line number is 65000.

Maximum length of a file pathname (including the directory path) is 255 characters.

Maximum number of files simultaneously open is 10 on the SD card and one on the internal flash drive (A:).

Maximum SD card size is 2GB formatted with FAT16 or 2TB formatted with FAT32.

Size of the internal flash drive (A:) is 200KB.

Maximum size of a loadable video font is 64 pixels high x 255 pixels wide and 256 characters.

Compatibility

MMBasic implements a large subset of Microsoft's GW-BASIC. There are numerous small differences due to physical and practical considerations but most MMBasic 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 SUB/END SUB, the DO WHILE … LOOP and structured IF .. THEN … ELSE … ENDIF statements.

License

MMBasic is Copyright 2011, 2012 Geoff Graham - http://mmbasic.com.

The compiled object code (the .hex file) is free software: you can use or redistribute it as you please.

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.

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

Predefined Read Only Variables

The centre column specifies the platform (MM is Maximite family, DOS is the Windows version).

MM.HRES MM.VRES	MM	The horizontal and vertical resolution of the current video display screen in pixels.
MM.HPOS MM.VPOS	MM	The current horizontal and vertical position (in pixels) following the last graphics or print command.
MM.VER	MM DOS	The version number of the firmware in the form aa.bbcc where aa is the major version number, bb is the minor version number and cc is the revision number (normally zero but A = 01, B = 02, etc).
MM.DEVICE$	MM DOS	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. "DuinoMite" when running on one of the DuinoMite family. "DOS" when running on Windows XP/Vista/ "Generic PIC32" for the generic version of MMBasic on a PIC32.
MM.DRIVE$	MM	The current default drive returned as a string containing either “A:” or “B:”.
MM.FNAME$	MM DO	The name of the file that will be used as the default for the SAVE command. This is set by LOAD, RUN and SAVE.
MM.ERRNO	MM DOS	Is set to the error number if a statement involving the SD card fails or zero if the operation succeeds. This is dependent on the setting of OPTION ERROR. For the Maximite the possible values for MM.ERRNO are: 0 = No error 1 = No SD card found 2 = SD card is write protected 3 = Not enough space 4 = All root directory entries are taken 5 = Invalid filename 6 = Cannot find file 7 = Cannot find directory 8 = File is read only 9 = Cannot open file 10 = Error reading from file 11 = Error writing to file 12 = Not a file 13 = Not a directory 14 = Directory not empty 15 = Hardware error accessing the storage media 16 = Flash memory write failure

Commands

The centre column specifies the platform (MM is Maximite family and DOS is the Windows version).
Square brackets indicate that the parameter or characters are optional.

‘ (single quotation mark)	MM DOS	Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.
? (question mark)	MM DOS	Shortcut for the PRINT command.
AUTO or AUTO start or AUTO start, increment	MM DOS	Enter automatic line entry mode. To terminate this mode use Control-C. With no arguments this command will take lines of text from the keyboard or USB and append them to program memory without modification. This is useful for adding lines that do not have line numbers and when pasting a program into a terminal emulator. If 'start' is provided the lines will be prefixed with an automatically generated line number. 'start' is the starting line number and 'increment' is the step size (default 10). If the automatically generated number is the same as an existing line in memory it will be preceded by an asterisk (*). In this case pressing Enter without entering any text will preserve the line in memory and generate the next number.
BLIT x1, y1, x2, y2, w, h or BLIT x1, y1, x2, y2, w, h, RGB	MM	Copy one section of the video screen to another. 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'. All arguments are in pixels. The source and destination can overlap. Colour Maximite only: If the optional argument 'RGB' is specified then only the specified colour planes will be copied. For example, 'GB' will copy only the green and blue colour planes.
CHDIR dir$	MM DOS	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.
CIRCLE (x, y) ,r [,c [,F]]	MM	Draws a circle on the video output centred at 'x' and 'y' with a radius of 'r'. 'c' is the colour and defaults to the current foreground colour if not specified. 'c' can also be -1 which will invert the pixels. The F option will cause the circle to be filled according to the 'c' parameter. See the section "Graphics and Working with Colour" for a definition of the colours and graphics coordinates. Note that because the pixels are not exactly square the circle will be oval to some degree.
CLEAR	MM DOS	Delete all variables and recover the memory used by them. See ERASE for deleting specific array variables.
CLOSE [#]nbr [,[#]nbr] …	MM DOS	Close the file(s) or serial port(s) previously opened with the file number ‘nbr’. The # is optional. Also see the OPEN command.
CLOSE CONSOLE	MM	Close a serial port that had been previously opened as the console.

CLS	MM DOS	Clears the video display screen and places the cursor in the top left corner.
COLOUR fore [, back] or COLOR fore [, back]	MM	Colour Maximite only. Sets the default colour for commands that display on the screen (PRINT, LINE, etc). 'fore' is the foreground colour, 'back' is the background colour. The background is optional and if not specified will default to black. The actual colour displayed will depend on the current colour mode (see the MODE command). See "Working with Colour" at the start of this manual for more details.
CONFIG COMPOSITE NTSC | PAL or CONFIG VIDEO OFF | ON or CONFIG CASE UPPER | LOWER | TITLE Or CONFIG KEYBOARD US | FR | GR | BE | IT or CONFIG TAB 2 | 4 | 8	MM	The COMPOSITE setting will change the timing for the composite video output. Default is PAL. The VIDEO setting will switch the video output on or off. There is a performance improvement with the video off but the biggest benefit is that the unused memory is returned to the memory pool. Default is ON. The CASE setting will change the case used for listing command and function names when using the LIST command. The default is TITLE but the old standard of MMBasic can be restored using CONFIG CASE UPPER. The KEYBOARD setting will change the keyboard layout to suit standard keyboards (US), French (FR), German (GR), Belgium (BE) or Italian (IT) keyboards. Default is US. The TAB setting will set the spacing for the tab key. Default is 2. The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off). The power must be cycled after changing a setting for it to take effect.
CONTINUE	MM DOS	Resume running a program that has been stopped by an END statement, an error, or CTRL-C. The program will restart with the next statement following the previous stopping point.
COPY src$ TO dest$	MM DOS	Copy the file named 'src$' to another file named 'dest$'. 'dest$' can be just a drive designation (ie, A:) and this makes it convenient to copy files between drives.
COPYRIGHT	MM DOS	List all contributors to MMBasic and summarise the copyright.
DATA constant[,constant]...	MM DOS	Stores numerical and string constants to be accessed by READ. String constants do not need to be quoted unless they contain significant spaces, the comma or a keyword (such as THEN, WHILE, etc). Numerical constants can also be expressions such as 5 * 60.
DATE$ = "DD-MM-YY" or DATE$ = "DD/MM/YY"	MM	Set the date of the internal clock/calendar. DD, MM and YY are numbers, for example: DATE$ = “28-7-2012” Normally the date is set to “1-1-2000” on power up. If the real time clock option is fitted to the Colour Maximite the current date will be automatically set on power up using that facility.

DELETE line DELETE -lastline DELETE firstline - DELETE firstline - lastline	MM DOS	Deletes a program line or a range of lines. If '–lastline' is used it will delete from the start of the first line in the program to the end of 'lastline'. If 'startline-' is used it will delete from start of 'startline' to the end of the program. Also see the NEW command.
DIM variable(elements...) [variable(elements...)]...	MM DOS	Specifies variables that have more than one element in a single dimension, i.e., arrayed variables.
DO <statements> LOOP	MM DOS	This structure will loop forever; the EXIT command can be used to terminate the loop or control must be explicitly transferred outside of the loop by commands like GOTO or RETURN (if in a subroutine).
DO WHILE expression <statements> LOOP	MM DOS	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, even once.
DO <statements> LOOP UNTIL expression	MM DOS	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 false.
DRIVE drivespec$	MM	Change the default drive used for file operations that do not specify a drive to that specified in drivespec$. This can be the string “A:” or “B:”. See also the predefined read-only variable MM.DRIVE$.
EDIT or EDIT filename or EDIT line-number	MM	Invoke the full screen editor. This can be used to edit either the program currently loaded in memory or a program file. It can also be used to view and edit text data files. If EDIT is used on its own it will edit the program memory. If 'filename' is supplied the file will be edited leaving the program memory untouched. On entry the cursor will be automatically positioned at the last line edited or, if there was an error when running the program, the line that caused the error. If 'line-number' is specified on the command line the program in memory will be edited and cursor will be placed on the line specified. The editing keys are: Left/Right arrows — Moves the cursor within the line. Up/Down arrows — Moves the cursor up or down a line. Page Up/Down — Move up or down a page of the program. Home/End — Moves the cursor to the start or end of the line. A second Home/End will move to the start or end of the program. Delete — Delete the character over the cursor. This can be the line separator character and thus join two lines. Backspace — Delete the character before the cursor. Insert — Will switch between insert and overtype mode. Escape Key — Will close the editor without saving (confirms first). Function Key 1 — Will save the edited text and exit.

		Function Key 2 Will save, exit and run the program. Function Key 3 Will invoke the search function. SHIFT F3 Will repeat the search using the text entered at F3. Function Key 4 Will mark text for cut or copy (see below). Function Key 5 Will paste text previously cut or copied. CTRL-F Will insert a file into the program being edited. When in the mark text mode (entered with F4) the editor will allow you to use the arrow keys to highlight text which can be deleted, cut to the clipboard or simply copied to the clipboard. The status line will change to indicate the new functions of the function keys. While the full screen editor is running it will override the programmable function keys F1 to F5. When the editor exits all programmable functions will be restored. The editor will work with lines wider than the screen but characters beyond the screen edge will not be visible. You can split such a line by inserting a new line character and the two lines can be later rejoined by deleting the inserted new line character. All the editing keys work with a VT100 terminal emulator so editing can also be accomplished over a USB or serial link. The editor has been tested with Tera Term and this is the recommended software. Note that Tera Term must be configured for an 80 column by 36 line display.
ELSE	MM DOS	Introduces a default condition in a multiline IF statement. See the multiline IF statement for more details.
ELSEIF expression THEN	MM DOS	Introduces a secondary condition in a multiline IF statement. See the multiline IF statement for more details.
ENDIF	MM DOS	Terminates a multiline IF statement. See the multiline IF statement for more details.
END	MM DOS	End the running program and return to the command prompt.
END FUNCTION	MM DOS	Marks the end of a user defined function. See the FUNCTION command. Each sub must have one and only one matching END FUNCTION statement. Use EXIT FUNCTION if you need to return from a subroutine from within its body. Only one space is allowed between END and FUNCTION.
END SUB	MM DOS	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. Only one space is allowed between END and SUB.
ERASE variable [,variable]...	MM DOS	Deletes arrayed variables and frees up the memory. Use CLEAR to delete all variables including all arrayed variables.

ERROR [error_msg$]	MM DOS	Forces an error and terminates the program. This is normally used in debugging or to trap events that should not occur.
EXIT EXIT FOR EXIT FUNCTION EXIT SUB	MM DOS	EXIT by itself provides an early exit from a DO...LOOP EXIT FOR provides an early exit from a FOR...NEXT loop. EXIT FUNCTION provides an early exit from a defined function. EXIT SUB provides an early exit from a defined subroutine. Only one space is allowed between the two words.
FILES [fspec$]	MM DOS	Lists files in the current directory on the SD or internal drive (drive A:). The SD card (drive B:) may use an optional 'fspec $'. Question marks (?) will match any character and an asterisk (*) will match any number of characters. If omitted, all files will be listed. For example: *.* Find all entries *.TXT Find all entries with an extension of TXT E*.* Find all entries starting with E X?X.* Find all three letter file names starting and ending with X
FONT #nbr or FONT #nbr, scale or FONT #nbr, [scale], reverse	MM	Selects a font for the video output. 'nbr' is the font number in the range of 1 to 10. The # symbol is optional. 'scale' is the multiply factor in the range of 1 to 8 (eg, a scale of 2 will double the size of a pixel in both the vertical and horizontal). Default is 1. If 'reverse' is a number other than zero the font will be displayed in reverse video. Default is no reverse. There are three fonts built into MMBasic: #1 is the standard font of 10 x 5 pixels containing the full ASCII set. #2 is a larger font of 16 x 11 pixels also with the full ASCII set. #3 is a jumbo font of 30 x 22 pixels consisting of the numbers zero to nine and the characters plus, minus, space, comma and full stop. Examples: 10 FONT #3, 2, 1 ' double scale and reverse video 10 FONT #3, ,0 ' reset to normal video 10 FONT #2 ' just select font #2 Font #1 with a scale of one and no reverse is the default on power up and will be reinstated whenever control returns to the input prompt. Other fonts can be loaded into memory: see the FONT LOAD command.
FONT LOAD file$ AS #nbr	MM	Loads the font contained in 'file$' and install it as font 'nbr' which can be any number between 3 and 10. The # symbol is optional. Appendix E describes the format of the font file.
FONT UNLOAD #nbr	MM	Removes font 'nbr' and frees the memory used. The # symbol is optional. You cannot unload the built-in fonts.
FOR counter = start TO finish [STEP increment]	MM DOS	Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' equals 'finish'. The 'increment' must be an integer, but may be negative. See also the NEXT command.

FUNCTION xxx (arg1 [,arg2, …])
<statements>
<statements>
xxx = <return value>
END FUNCTION
	MM
	DOS
		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. 'arg1', 'arg2', etc are the arguments or parameters to the function.
		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 (ie, not an expression or constant) 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.
		You must not jump into or out of a function using commands like GOTO, GOSUB, interrupts, etc. Doing so will have undefined side effects.

GOSUB target	MM DOS	Initiates a subroutine call to the target, which can be a line number or a label. The subroutine must end with RETURN.
GOTO target	MM DOS	Branches program execution to the target, which can be a line number or a label.
IF expr THEN statement or IF expr THEN statement ELSE statement	MM DOS	Evaluates the expression ‘expr' and performs the THEN statement if it is true or skips to the next line if false. The optional ELSE statement is the reverse of the THEN test. This type of IF statement is all on one line. The ‘THEN statement’ construct can be also replaced with: GOTO linenumber | label’.

IF expression THEN
<statements>
[ELSE
<statements>]
[ELSEIF expression THEN
<statements>]
ENDIF
	MM
	DOS
		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.

INPUT ["prompt string$";] list of variables	MM DOS	Allows input from the keyboard to a list of variables. The input command will prompt with a question mark (?). The input must contain commas to separate each data item if there is more than one variable. For example, if the command is: INPUT a, b, c And the following is typed on the keyboard: 23, 87, 66 Then a = 23 and b = 87 and c = 66 If the "prompt string$" is specified it will be printed before the question mark. If the prompt string is terminated with a comma (,) rather than the semicolon (;) the question mark will be suppressed.
INPUT #nbr, list of variables	MM DOS	Same as above except that the input is read from a file previously opened for INPUT as 'nbr'. See the OPEN command.
IRETURN	MM	Returns from an interrupt. The next statement to be executed will be the one that was about to be executed when the interrupt was detected.
KILL file$	MM DOS	Deletes the file specified by 'file$'. Quote marks are required around a string constant and the extension, if there is one, must be specified. Example: KILL "SAMPLE.DAT"
LET variable = expression	MM DOS	Assigns the value of 'expression' to the variable. LET is automatically assumed if a statement does not start with a command.
LINE [(x1 , y1)] - (x2, y2) [,c [,B[F]]]	MM	Draws a line or box on the video screen. x1,y1 and x2,y2 specify the beginning and end points of a line. 'c' specifies the colour and defaults to the default foreground colour if not specified. It can also be -1 to invert the pixels. (x1, y1) is optional and if omitted the last drawing point will be used. The optional B will draw a box with the points (x1,y1) and (x2,y2) at opposite corners. The optional BF will draw a box (as ,B) and fill the interior. See the section "Graphics and Working with Colour" for a definition of the colours and graphics coordinates.
LINE INPUT [prompt$,] string-variable$	MM DOS	Reads entire line from the keyboard into 'string-variable$'. If specified the 'prompt$' will be printed first. Unlike INPUT, LINE INPUT will read a whole line, not stopping for comma delimited data items. A question mark is not printed unless it is part of 'prompt$'.
LINE INPUT #nbr, string-variable$	MM DOS	Same as above except that the input is read from a file previously opened for INPUT as 'nbr'. See the OPEN command.
LIST LIST line LIST -lastline LIST firstline - LIST firstline - lastline	MM DOS	Lists all lines in a program line or a range of lines. If –lastline is used it will start with the first line in the program. If startline- is used it will list to the end of the program.

LOAD file$	MM DOS	Loads a program called ‘file$’ from the current drive into program memory. Quote marks are required around a string constant. Example: LOAD “TEST.BAS” If an extension is not specified “.BAS” will be added to the file name.
LOADBMP file$ [, x, y]	MM	Load a bitmapped image and display it on the video screen. "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 “.BMP” will be added to the file name. The file must use either a monochrome or 16 colours (4 bit) colour depth and be in the uncompressed BMP format. Microsoft's Paint program is recommended for creating suitable images. See also SAVEBMP.
LOCAL variable [, variables]	MM DOS	Defines a list of variable names as local to the subroutine or function. 'variable' can be an array and the array will be dimensioned just as if the DIM command had been used. A local variable will only be visible within the procedure and will be deleted (and the memory reclaimed) when the procedure returns. If the local variable has the same name as a global variable (used before any subroutines or functions were called) the global variable will be hidden by the local variable while the procedure is executed.
LOOP [UNTIL expression]	MM DOS	Terminates a program loop: see DO.

MEMORY	MM
	DOS
		List the amount of memory currently in use. For example:
		`15kB (18%) Program (528 lines)`
		`23kB (28%) 52 Variables`
		`17kB (21%) General`
		`28kB (33%) Free`
		Program memory is cleared by the NEW command. Variable and the general memory spaces are cleared by many commands (eg, NEW, RUN, LOAD, etc) as well as the specific commands CLEAR and ERASE.
		General memory is used by fonts, file I/O buffers, etc.

MERGE file$	MM DOS	Adds program lines from 'file$' to the program in memory. Unlike LOAD, it does not clear the program currently in memory.
MKDIR dir$	MM DOS	Make, or create, the directory ‘dir$’ on the SD card.

MODE mode [, palette]	MM	Colour Maximite only. Sets the number of colours that can be displayed on the screen. 'mode' can be: 1 Monochrome mode. 'palette' will select the colour to use and can be 0 to 7 representing the colours black to white. This mode provides complete compatibility with programs written for the monochrome Maximite 2 Four colour mode. 'palette' can be a number from 1 to 6 and will select the range of colours available (see table below). 3 Eight colour mode. In this mode all eight colours (including black and white) can be used. 'palette' can be supplied but will be ignored. 4 240x216 pixel resolution with all eight colours (including black and white) available. 'palette' can be supplied but will be ignored. In mode 2 the colours available in each palette are: palette = 1 Black, Red, Green, Yellow palette = 2 Black, Red, Blue, Purple palette = 3 Black, Red, Cyan, White palette = 4 Black, Green, Blue, Cyan palette = 5 Black, Green, Purple, White palette = 6 Black, Blue, Yellow, White The MODE command allows the programmer to trade the number of colours used and resolution against the amount of memory required by the video driver. Modes 1 and 4 use the least amount of memory while mode 3 uses the most. Also see "Graphics and Working with Colour" at the start of this manual.
NAME old$ AS new$	MM DOS	Rename a file or a directory from ‘old$’ to ‘new$’ Unlike the other commands that work with file names the NAME command cannot accept a full pathname (with directories).
NEW	MM DOS	Deletes the program in memory and clears all variables.
NEXT [counter-variable] [, counter-variable], etc	MM DOS	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 counter-variables as in: NEXT x, y, z
ON nbr GOTO | GOSUB target[,target, target,...]	MM DOS	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.

OPEN fname$ FOR mode AS [#]fnbr	MM DOS	Opens a file for reading or writing. 'fname' is the filename (8 chars max) with an optional extension (3 chars max) separated by a dot (.). It can be prefixed with a directory path. For example: "B:\DIR1\DIR2\FILE.EXT". 'mode' is INPUT or OUTPUT or APPEND. 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). Note: APPEND is not supported on the flash file system (drive A:). '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$() functions all use 'fnbr' to identify the file being operated on. See also OPTION ERROR and MM.ERRNO for error handling.
OPEN comspec$ AS [#]fnbr or OPEN comspec$ AS console	MM	Will open a serial port for reading and writing. Two ports are available (COM1: and COM2:) and both can be open simultaneously. For a full description with examples see Appendix A. 'comspec$' is the serial port specification and has the form: "COMn: baud, buf, int, intlevel" with an optional ",FC" and/or ",OC" appended. COM1: uses pin 15 for receive data and pin 16 for transmit data and if flow control is specified pin 17 for RTS and pin 18 for CTS. COM2: uses pin 19 for receive data and pin 20 for transmit data on the monochrome Maximite and D0 (receive) and D1 (transmit) on the Colour Maximite. For the DuinoMite see the "DuinoMite MMBasic ReadMe.pdf" document for details. If the port is opened using 'fnbr' the port can be written to and read from using any commands or functions that use a file number. A serial port can be opened with "AS CONSOLE". In this case any data received will be treated the same as keystrokes received from the keyboard and any characters sent to the video output will also be transmitted via the serial port. This enables remote control of MMBasic via a serial interface.
OPTION BASE 0 or OPTION BASE 1	MM DOS	Set the lowest value for array subscripts to either 0 or 1. The default is 0. This must be used before any arrays are declared.

OPTION ERROR CONTINUE or OPTION ERROR ABORT	MM DOS	Set the treatment for errors in file input/output. The option CONTINUE will cause MMBasic to ignore file related errors. The program must check the variable MM.ERRNO to determine if and what error has occurred. The option ABORT sets the normal behaviour (ie, stop the program and print an error message). The default is ABORT. Note that this option only relates to errors reading from or writing to drives A: and B:. It does not affect the handling of syntax and other program errors.
OPTION PROMPT string$	MM DOS	Sets the command prompt to the contents of 'string$' (which can also be an expression which will be evaluated when the prompt is printed). For example: OPTION PROMPT "Ok " or OPTION PROMPT TIME$ + ": " or OPTION PROMPT CWD$ + ": " Maximum length of the prompt string is 48 characters. The prompt is reset to the default ("> ") on power up but you can automatically set it by saving the following example program as "AUTORUN.BAS" on the internal flash drive A:. 10 OPTION PROMPT "My prompt: " 20 NEW
OPTION Fnn string$	MM	Sets the programmable function key 'Fnn' to the contents of 'string$'. 'Fnn' is the function key F1 to F12. Maximum string length is 12 characters. 'string$' can also be an expression which will be evaluated at the time of running the OPTION command. This is most often used to append the ENTER key (chr$(13)), or double quotes (chr$(34)). For example: OPTION F1 "RUN" + CHR$(13) OPTION F6 "SAVE " + CHR$(34) OPTION F10 "ENDIF" Normally these commands are included in an AUTORUN.BAS file (see OPTION PROMPT for an example).
OPTION USB OFF or OPTION USB ON	MM	Turn the USB output off and on. This disables/enables the output from the PRINT command from being sent out on the USB interface. It does not affect the reception of characters from the USB interface. Normally this is used when a program wants to separately display data on the USB and video interfaces. This option is always reset to ON at the command prompt.
OPTION VIDEO OFF or OPTION VIDEO ON	MM	VIDEO OFF prevents the output from the PRINT command from being displayed on the video output (VGA or composite). The VIDEO ON option will revert to the normal action. Normally this is used when a program wants to separately display data on the USB and video outputs. This option is always reset to ON at the command prompt.

PAUSE delay	MM DOS	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 4294967295 mS (about 49 days).
PIN(pin) = value	MM	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. 'pin' zero is a special case and will always control the LED on the front panel of the Maximite, the yellow LED on the UBW32 or the green LED on the DuinoMite. A 'value' of non-zero will turn the LED on, or zero for off. See the function PIN() for reading from a pin and the command SETPIN for configuring it.
PIXEL(x,y) = colour	MM	Set a pixel on the VGA or composite screen to a colour or inverted (if the value is -1). See the section "Graphics and Working with Colour" for a definition of the colours and graphics coordinates. See the function PIXEL(x,y) for obtaining the value of a pixel.
PLAYMOD file [, dur] or PLAYMOD STOP	MM CMM	Play synthesised music or sound effects. 'file' specifies a file which must be located on the internal drive A: and must be in the .MOD format. 'dur' specifies the duration in milliseconds that the audio will play for - if not specified it will play until explicitly stopped or the program terminates. The audio is synthesised in the background and is not disturbed by the running program. The command PLAYMOD STOP will immediately halt any music or sound effect that is currently playing. NOTE: The file needs to be located on the internal drive A: for performance reasons, it will not play from the SD card.
POKE hiword, loword, val or POKE keyword, offset, val	MM DOS	Will set a byte within the PIC32 virtual memory space. The address is specified by 'hiword' which is the top 16 bits of the address while 'loword' is the bottom 16 bits. Alternatively 'keyword' can be used and 'offset' is the ±offset from the address of the keyword. The keyword can be VIDEO (monochrome video buffer) or RVIDEO, GVIDEO, BVIDEO (red, blue and green video buffers), PROGMEM (program memory) or VARTBL (the variable table). The input keystroke buffer is KBUF, the head of the keystroke queue is KHEAD and the tail is KTAIL (the buffer is 256 bytes long). This command is for expert users only. The PIC32 maps all control registers, flash (program) memory and volatile (RAM) memory into a single address space so there is no need for INP or OUT commands. The PIC32MX5XX/6XX/7XX Family Data Sheet lists the details of this address space. Note that MMBasic stores most data (including video) as 32 bit integers and the PIC32 uses little endian format. WARNING: No validation of the parameters is made and if you use this facility to access an invalid memory address the MIPS CPU will throw an exception which causes the processor to reset and clear all memory. To see this effect try POKE 0,0,0.

PRINT expression [[,;]expression] … etc	MM DOS	Outputs text to the screen. 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 numbers (greater than six digits) are printed in scientific format. The function FORMAT$() can be used to format numbers. 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. A single question mark (?) can be used as a shortcut for the PRINT keyword.
PRINT @(x, y) expression Or PRINT @(x, y, m) expression	MM	Same as the PRINT command except that the cursor is positioned at the coordinates x, y. 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)
PRINT #nbr, expression [[,;]expression] … etc	MM DOS	Same as above except that the output is directed to a file previously opened for OUTPUT or APPEND as ‘nbr’. See the OPEN command.
PULSE pin, width	MM	Will generate a pulse on 'pin' with duration of 'width' mS. 'width' can be a fraction. For example, 0.01 is equal to 10 µS 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: For a pulse of less than 1 mS the accuracy is ± 1 µS. For a pulse of 1 mS or more the accuracy is ± 0.5 mS. A pulse of more than 1 mS will run in the background. 'pin' must be configured as an output.

PWM freq, ch1, ch2 or PWM STOP	MM	Generate a pulse width modulated (PWM) output for driving analogue circuits. ‘freq’ is the output frequency (between 20 Hz and 1 MHz) . The frequency can be changed at any time by issuing a new PWM command. The output will run continuously in the background while the program is running and can be stopped using the PWM STOP command. 'ch1' and 'ch2' are the output duty cycles for channel 1 and 2 as a percentage. If the percentage 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. Both are optional and if not specified will default to the previously used duty cycle for that channel. The PWM output is generated on the PWM/sound connector and that assumes that the connector has been wired for PWM output. The frequency of the output is locked to the PIC32 crystal and is very accurate and for frequencies below 100 KHz the duty cycle will be accurate to 0.1%. The original monochrome Maximite has only one PWM/sound output so only 'ch1' can be set on that model.
QUIT	DOS	Will exit MMBasic and return control to the operating system.
RANDOMIZE nbr	MM DOS	Seeds the random number generator with ‘nbr’. To generate a different random sequence each time you must use a different value for ‘nbr’. One good way to do this is use the TIMER function. For example 100 RANDOMIZE TIMER
READ variable[, variable]...	MM DOS	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. See also DATA and RESTORE.
REM string	MM DOS	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.
RENUMBER or RENUMBER first or RENUMBER first, incr or RENUMBER first, [incr], start	MM DOS	Renumber the program currently held in memory including all references to line numbers in commands such as GOTO, GOSUB, ON, etc. ‘first’ is the first number to be used in the new sequence. Default is 10. ‘incr’ is the increment for each line. Default is 10. ‘start’ is the line number in the old program where renumbering should commence from. The default is the first line of the program. This command will first check for errors that may disrupt the renumbering process and it will only change the program in memory if no errors are found. However, it is prudent to save the program before running this command in case there are some errors that are not caught.
RESTORE	MM DOS	Resets the line and position counters for DATA and READ statements to the top of the program file.
RETURN	MM DOS	RETURN concludes a subroutine called by GOSUB and returns to the statement after the GOSUB.

RMDIR dir$	MM DOS	Remove, or delete, the directory ‘dir$’ on the SD card.
RUN [line] [file$]	MM DOS	Executes the program in memory. If a line number is supplied then execution will begin at that line, otherwise it will start at the beginning of the program. Or, if a file name (file$) is supplied, the current program will be erased and that program will be loaded from the current drive and executed. This enables one program to load and run another. Example: RUN “TEST.BAS” If an extension is not specified “.BAS” will be added to the file name.
SAVE [file$]	MM DOS	Saves the program in the current working directory as ‘file$’. The file name is optional and if omitted the last filename used in SAVE, LOAD or RUN will be automatically used. Example: SAVE “TEST.BAS” If an extension is not specified “.BAS” will be added to the file name.
SAVEBMP file$	MM	Saves the current VGA or composite screen as a BMP file in the current working directory on the current drive. The Colour Maximite will save the file as a 16 colour (4 bit) file. Example: SAVEBMP “IMAGE.BMP” If an extension is not specified “.BMP” will be added to the file name. Note that Windows 7 Paint has trouble displaying monochrome images. This appears to be a bug in Paint as all other software tested (including Windows XP Paint) can display the image without fault. See also LOADBMP.
SCANLINE colour, start [, end]	MM	Colour Maximite only. This command can be used to set the colour of individual horizontal scan lines on the VGA monitor when in MODE 1,7 (monochrome with white foreground). This applies to all video output displayed before and after the SCANLINE command. 'colour' can be any colour specified by name or number from 0 to 7. 'start' and 'end' specify the range of scan lines to set. If 'end' is not specified only one line will be set. Multiple calls to SCANLINE can be used to set the colour of other scan lines or to change the colour of lines already set (ie, the settings accumulate). This command is valid only when the colour mode is set to MODE 1,7 (monochrome with the colour set to white). All settings made by SCANLINE are automatically cancelled whenever the MODE command is used or when MMBasic returns to the command prompt. See the section "Graphics and Working with Colour" for more details.

SETPIN pin, cfg	MM	Will configure the external I/O 'pin' according to 'cfg'. The original Maximite has 20 I/O pins numbered 1 to 20, the Colour Maximite adds another 20 I/O pins on the Arduino connector. These are labelled D0 to D13 and A0 to A5. The possible configurations are: 0 Not configured or inactive 1 Analog input (pins 1 to 10, A0 to A5) 2 Digital input (all pins) 3 Frequency input (pins 11 to 14) 4 Period input (pins 11 to 14) 5 Counting input (pins 11 to 14) 8 Digital output (all pins) 9 Open collector digital output (pins 11 to 20, D0 to D13) In this mode the function PIN() will also return the output value. Pins 11 to 20 and D0 to D13 can accept 5V inputs and 5V pull-ups in open collector mode. The remainder have a maximum input voltage of 3.3V. For the DuinoMite see "DuinoMite MMBasic ReadMe.pdf" See the function PIN() for reading inputs and the statement PIN()= for outputs. See the command below if an interrupt is configured.
SETPIN pin, cfg , target	MM	Will configure 'pin' to generate an interrupt according to 'cfg': 0 Not configured or inactive 6 Interrupt on low to high input (pins 1 to 20, D0 to D8) 7 Interrupt on high to low input (pins 1 to 20, D0 to D8) The starting line number of the interrupt routine is specified in the third parameter 'target', which can be a line number or label. This mode also configures the pin as a digital input so the value of the pin can always be retrieved using the function PIN(). See also IRETURN to return from the interrupt.
SETTICK period, target	MM	This will setup a periodic interrupt (or "tick"). The time between interrupts is 'period' milliseconds and 'target' is the line number or label of the interrupt routine. See also IRETURN to return from the interrupt. The period can range from 1 to 4294967295 mSec (about 49 days). This interrupt can be disabled by setting 'target' to zero (ie, SETTICK 0, 0).
SPRITE LOAD file or SPRITE ON n, x, y or SPRITE MOVE n, x, y or SPRITE OFF n or SPRITE UNLOAD	MM	Load and manipulate sprites on the screen. Sprites are 16x16 pixel objects that can be moved about on the screen without erasing or disturbing text or other underlying graphics. This command is mostly used in animated games. 'n' is the sprite number, 'x' and 'y' are the sprite's coordinates on the screen. SPRITE LOAD will load a sprite file into memory. This file defines the graphic images of one or more sprites. See Appendix H for details. SPRITE ON will display an individual sprite contained in the sprite file. SPRITE OFF will remove the sprite from the screen and restore the background graphics that was obscured when the sprite was turned on. SPRITE MOVE will move the sprite to a new location and restore the background at the old location.

		SPRITE UNLOAD will disable the sprites, unload the file and reclaim the memory used. The sprite file can contain many individual sprites which can be simultaneously displayed and independently manipulated at the same time. The number of sprites is limited only by the available memory. See the section "Game Playing Features" for more details.
SUB xxx (arg1 [,arg2, …]) <statements> <statements> END SUB	MM DOS	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. 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 (ie, not an expression or constant) 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. Brackets around the argument list in both the caller and the definition are optional.
SYSTEM command$	DOS	Submit 'command$' to the operating system. It can be any command recognised by the command window in Windows XP/Vista/7. The available commands are listed here: http://ss64.com/nt For example, this will set the window to blue lettering on a yellow background: SYSTEM "COLOR 1E" Note that the command is executed in a different instance of the command processor to MMBasic so some commands (like "CD ..") will have no effect.
TIME$ = “HH:MM:SS” or TIME$ = “HH:MM” or TIME$ = “HH”	MM	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. Normally the time is set to “00:00:00” on power up. If the real time clock option is fitted to the Colour Maximite the current time will be automatically set using that facility.
TIMER = msec	MM DOS	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.

TONE left [, right [, dur]] or TONE STOP	MM	Generates a continuous sine wave on the sound output. 'left' and 'right' are the frequencies 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. If the duration is not specified the tone will continue until explicitly stopped or the program terminates. The command TONE STOP will immediately halt the tone output. The frequency can be from 1Hz to 20KHz and is very accurate (it is based on the PIC32 crystal oscillator). The frequency can be changed at any time by issuing a new TONE command. In the monochrome Maximite and compatibles only the left frequency will play but a dummy or empty value is still required for the right channel.
TROFF	MM DOS	Turns the trace facility off; see TRON.
TRON	MM DOS	Turns on the trace facility. This facility will print each line number in square brackets as the program is executed. This is useful in debugging programs.
XMODEM SEND file$ or XMODEM RECEIVE file$	MM	Transfers a file to or from a remote computer using the XModem protocol. The transfer is done over the USB connection or, if a serial port is opened as console, over that serial port. 'file$' is the file (on the SD card or internal flash) to be sent or received. 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. Download Tera Term from http://ttssh2.sourceforge.jp/

Functions

The centre column specifies the platform (MM is Maximite family, DOS is the Windows version).
Square brackets indicate that the parameter or characters are optional.

ABS(number)	MM DOS	Returns the absolute value of the argument 'number' (ie, any negative sign is removed and the positive number is returned).
ASC(string$)	MM DOS	Returns the ASCII code for the first letter in the argument ‘string$’.
ATN(number)	MM DOS	Returns the arctangent value of the argument 'number' in radians.
BIN$(number)	MM DOS	Returns a string giving the binary (base 2) value for the 'number'.
CHR$(number)	MM DOS	Returns a one-character string consisting of the character corresponding to the ASCII code indicated by argument 'number'.
CINT(number)	MM DOS	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().
COS(number)	MM DOS	Returns the cosine of the argument 'number' in radians.
CWD$	MM DOS	Returns the current working directory as a string.
DATE$	MM DOS	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$ =.
DEG(radians)	MM DOS	Converts 'radians' to degrees.
DIR$(fspec, type) or DIR$(fspec) or DIR$( )	MM	Will search an SD card for files and return the names of entries found. 'fspec' is a file specification using wildcards the same as used by the FILES command. Eg, "*.*" will return all entries, "*.TXT" will return text files. 'type' is the type of entry to return and can be one of: VOL Search for the volume label only DIR Search for directories only FILE Search for files only (the default if 'type' is not specified) The function will return the first entry found. To retrieve subsequent entries use the function with no arguments. ie, DIR$( ). 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$(*.*, FILES)`
		`DO WHILE f$ <> ""`
		`  PRINT f$`
		`  f$ = DIR$()`
		`LOOP`
		This function only operates on the SD card and you must change to the required directory before invoking it.

EOF([#]nbr)	MM DOS	Will return true if the file previously opened for INPUT with the file number 'nbr' is positioned at the end of the file. If used on a file number opened as a serial port this function will return true if there are no characters waiting in the receive buffer. The # is optional. Also see the OPEN, INPUT and LINE INPUT commands and the INPUT$ function.
EXP(number)	MM DOS	Returns the exponential value of 'number'.
FIX(number)	MM DOS	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$(nbr [, fmt$])	MM DOS	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 **
HEX$(number)	MM DOS	Returns a string giving the hexadecimal (base 16) value for the 'number'.
INKEY$	MM DOS	Checks the keyboard and USB input buffers 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 the input buffer is empty this function will immediately return with an empty string (ie, "").
INPUT$(nbr, [#]fnbr)	MM DOS	Will return a string composed of ‘nbr’ characters read from a file previously opened for INPUT with the file number ‘fnbr’. This function will read all characters including carriage return and new line without translation. When reading from a serial communications port this will return as many characters as are waiting in the receive buffer up to ‘nbr’. If there are no characters waiting it will immediately return with an empty string. The # is optional. Also see the OPEN command.
INSTR([start-position,] string-searched$, string-pattern$)	MM DOS	Returns the position at which 'string-pattern$' occurs in 'string-searched$', beginning at 'start-position'.
INT(number)	MM DOS	Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behaviour is for Microsoft compatibility, the FIX() function provides a true integer function. See also CINT() .
LEFT$(string$, number-of-chars)	MM DOS	Returns a substring of ‘string$’ with ‘number-of-chars’ from the left (beginning) of the string.
LEN(string$)	MM DOS	Returns the number of characters in 'string$'.
LOC([#]nbr)	MM	Will return the number of bytes waiting in the receive buffer of a serial port (ie, COM1: or COM2:) that has been opened as #nbr. The # is optional.
LOF([#]nbr)	MM	Will return the space (in bytes) remaining in the transmit buffer of a serial port (ie, COM1: or COM2:) that has been opened as #nbr. The # is optional.

LOG(number)	MM DOS	Returns the natural logarithm of the argument 'number'.
LCASE$(string$)	MM DOS	Returns ‘string$’ converted to lowercase characters.
MID$(string$, start-position-in-string[, number-of-chars])	MM DOS	Returns a substring of ‘string$’ beginning at ‘start-position-in-string’ and continuing for ‘number-of-chars’ bytes. If ‘number-of-chars’ is omitted the returned string will extend to the end of ‘string$’
OCT$(number)	MM DOS	Returns a string giving the octal (base 8) representation of 'number'.
PEEK(hiword, loword) or PEEK(keyword, ±offset)	MM DOS	Will return a byte within the PIC32 virtual memory space. The address is specifies by ‘hiword’ which is the top 16 bits of the address while ‘loword’ is the bottom 16 bits. Alternatively 'keyword' can be used and 'offset' is the ±offset from the address of the keyword. The keyword can be VIDEO (monochrome video buffer) or RVIDEO, GVIDEO, BVIDEO (red, blue and green video buffers), PROGMEM (program memory) or VARTBL (the variable table). The input keystroke buffer is KBUF, the head of the keystroke queue is KHEAD and the tail is KTAIL (the buffer is 256 bytes long). See the POKE command for notes and warnings related to memory access.
PI	MM DOS	Returns the value of pi.
PIN(pin)	MM	Returns the value on the external I/O ‘pin’. Zero means digital low, 1 means digital high and for analogue inputs it will return the measured voltage as a floating point number. Frequency inputs will return the frequency in Hz (maximum 200 kHz). 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). ‘pin’ zero (ie, = PIN(0)) is a special case which will always return the state of the bootload or program push button on the PC board (non zero means that the button is down). Also see the SETPIN and PIN() = commands.
POS	MM DOS	Returns the current cursor position in the line in characters.
PIXEL(x, y)	MM	Returns the colour of a pixel on the VGA or composite screen. See the section "Graphics and Working with Colour" for a definition of the colours and graphics coordinates.. See the statement PIXEL(x,y) = for setting the value of a pixel.
RAD(degrees)	MM DOS	Converts 'degrees' to radians.

RIGHT$(string$, number-of-chars)	MM DOS	Returns a substring of ‘string$’ with ‘number-of-chars’ from the right (end) of the string.
RND(number)	MM DOS	Returns a pseudo-random number in the range of 0 to 0.99999. The 'number' value is ignored if supplied. The RANDOMIZE command reseeds the random number generator.
SGN(number)	MM DOS	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN(number)	MM DOS	Returns the sine of the argument 'number' in radians.
SPACE$(number)	MM DOS	Returns a string of blank spaces 'number' bytes long.
SPI(rx, tx, clk[, data[, speed]])	MM	Sends and receives a byte using the SPI protocol with MMBasic as the master (ie, it generates the clock). ‘rx’ is the pin number for the data input (MISO) ‘tx’ is the pin number for the data output (MOSI) ‘clk’ is the pin number for the clock generated by MMBasic (CLK) ‘data’ is optional and is an integer representing the data byte to send over the data output pin. If it is not specified the ‘tx’ pin will be held low. ‘speed’ is optional and is the speed of the clock. It is a single letter either H, M or L where H is 500 kHz, M is 50 kHz and L is 5 kKHz. Default is H. See Appendix D for a full description.
SQR(number)	MM DOS	Returns the square root of the argument 'number'.
STR$(number)	MM DOS	Returns a string in the decimal (base 10) representation of 'number'.
STRING$(number, ascii-value\|string$)	MM DOS	Returns a string 'number' bytes long consisting of either the first character of string$ or the character representing the ASCII value ascii-value.
TAB(number)	MM DOS	Outputs spaces until the column indicated by 'number' has been reached.
TAN(number)	MM DOS	Returns the tangent of the argument 'number' in radians.
TIME$	MM DOS	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”. To set the current time use the command TIME$ = .

TIMER	MM DOS	Returns the elapsed time in milliseconds (eg, 1/1000 of a second) since reset. If not specifically reset this count will wrap around to zero after 49 days. The timer is reset to zero on power up and you can also reset it by using TIMER as a command.
UCASE$(string$)	MM DOS	Returns ‘string$’ converted to uppercase characters.
VAL(string$)	MM DOS	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 commands in MMBasic should be used.

The centre column specifies the platform (MM is Maximite family, DOS is the Windows version).
Square brackets indicate that the parameter or characters are optional.

IF condition THEN linenbr	MM DOS	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
LOCATE x, y	MM	Positions the cursor to a location in pixels and the next PRINT command will place its output at this location. This construct is included for Microsoft compatibility. New programs should use the PRINT @(x,y) construct (see the PRINT command).
PRESET (x, y) PSET (x, y)	MM	Turn off (PRESET) or on (PSET) a pixel on the video screen. These statements are included for Microsoft compatibility. New programs should use the PIXEL(x,y) = statement.
SOUND freq or SOUND freq, dur	MM	Generate a single tone of 'freq' (between 20 Hz and 1 MHz) for 'dur' milliseconds. The sound is played in the background and does not stop program execution. If 'dur' is not specified the sound will play forever until turned off. If 'dur' is zero, any active SOUND statement is turned off. The command has been replaced with the TONE command that generates a pure sine wave (not a square wave).
SPC(number)	MM DOS	This function returns a string of blank spaces 'number' bytes long. It is similar to the SPACE$() function and is only included for Microsoft compatibility.
WHILE expression WEND	MM DOS	WHILE initiates a WHILE-WEND loop. The loop ends with WEND, and execution reiterates through the loop as long as the 'expression' is true. This construct is included for Microsoft compatibility. New programs should use the DO WHILE … LOOP construct.
WRITE [#nbr,] expression [,expression] …	MM DOS	Outputs the value of each 'expression' separated by commas (,). If 'expression' is a number it is outputted without preceding or trailing spaces. If it is a string it is surrounded by double quotes ("). The list is terminated with a new line. If '#nbr' is specified the output will be directed to a file previously opened for OUTPUT or APPEND as '#nbr'. See the OPEN command. WRITE can be replaced by the PRINT command.

Appendix A
Serial Communications

Maximite family only (not DOS or Generic PIC32 versions).

Two serial ports are available for asynchronous serial communications (four on the DuinoMite). They are labelled COM1:, COM2:, etc and are opened in a manner similar to opening a file. After being opened they will have an associated file number (like an opened disk file) and you can use any commands that operate with a file number to read and write to/from the serial port. 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
Data$ = 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
```

The transmission format is fixed at 8 data bits, no parity and one stop bit.

'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.

It has the form `"COMn: baud, buf, int, intlevel, FC, OC"` where:

- 'n' is the serial port number for either COM1: or COM2: (plus COM3: and COM4: on the DuinoMite).
- 'baud' is the baud rate, either 19200, 9600, 4800, 2400, 1200, 600, 300 bits per second. For COM3 and COM4 (DuinoMite only) it can be any number from 300 to 460800. Default is 9600.
- 'buf' is the buffer sizes in bytes. Two of these buffers will be allocated from memory, one for transmit and one for receive. The default size is 256 bytes.
- 'int' is the line number or label of an interrupt routine to be invoked when the serial port has received some data. The default is no interrupt.
- 'intlevel' is the number of characters that must be waiting in the receive queue before the receive interrupt routine is invoked. The default is 1 character.

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.

Two options can be optionally added, these are FC and OC. They must be at the end of 'comspec' and, if both are specified, must be in this order.

- 'FC' will enable hardware flow control. Flow control can only be specified on COM1: and it enables two extra signals, Request To Send (receive flow control) and Clear To Send (transmit flow control). Default is no flow control.
- 'OC' will force the output pins (Tx and optionally RTS) to be open collector and can be used on both COM1: and COM2:. The default is normal (0 to 3.3V) output.

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 buffer size (1KB) but no flow control:

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

The same as above but with receive flow control (RTS) and transmit flow control (CTS) enabled:

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

An example specifying everything including an interrupt, an interrupt level, flow control and open collector:

```
OPEN "COM1:19200, 1024, ComIntLabel, 256, FC, OC" AS #5
```

Input/Output Pin Allocation

COM1: uses pin 15 for receive data (data in) and pin 16 for transmit data (data out). If flow control is specified pin 17 will be used for RTS (receive flow control – it is an output) and pin 18 will be CTS (transmit flow control – it is an input).

COM2: uses pin 19 for receive data (data in) and pin 20 for transmit data (data out) in the monochrome Maximite and D0 for receive data and pin D1 for transmit data on the Colour Maximite.

For the DuinoMite see the "DuinoMite MMBasic ReadMe.pdf" document for details.

When a serial port is opened the pins used by the port are automatically set to input or output as required and the SETPIN and PIN commands are 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 signal polarity is standard for devices running at TTL voltages (not RS232). 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. The flow control pins (RTS and CTS) use a low voltage to signal stop sending data and high as OK to send. These signal levels allow you to directly connect to devices like the EM-408 GPS module (which uses TTL voltage levels).

Reading and Writing

Once a serial port has been opened you can use any commands or functions that use a file number to write and read from the port. Generally the PRINT command is the best method for transmitting data and the INPUT$() function is the most convenient way of getting data that has been received. When using the INPUT$() 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 number characters that can be retrieved by the INPUT$() function). The EOF() function will return true if there are no characters waiting. The LOF() function will return the space (in characters) remaining in the transmit buffer.

When outputting to a serial port (ie, using PRINT #n, data) the command will pause if the output buffer is full and wait until there is sufficient space to place the new data in the buffer before returning. If the receive buffer overflows with incoming data the serial port will automatically discard the oldest data to make room for the new data.

Serial ports can be closed with the CLOSE command. This will discard any characters waiting in the buffers, return the buffer memory to the memory pool 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 routine (if specified) will operate the same as a general interrupt on an external I/O pin (see page 4 for a description) and must be terminated with an IRETURN command to return control to the main program when completed.

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. So, for example, if you have specified the interrupt level as 200 characters and a buffer of 256 characters then quite easily the buffer will have overflowed by the time the interrupt routine can read the data. In this case the buffer should be increased to 512 characters or more.

Opening a Serial Port as the Console

A serial port can be opened as the console for MMBasic. The command is:

```
OPEN comspec AS CONSOLE
```

In this case any characters received from the serial port will be treated the same as keystrokes received from the keyboard and any characters sent to the video output will also be transmitted via the serial port. This enables a user with a terminal at the end of the serial link to exercise remote control of MMBasic. For example, via a modem.

Note that only one serial port can be opened “AS CONSOLE” at a time and it will remain open until explicitly closed using the CLOSE CONSOLE command. It will not be closed by commands such as NEW and RUN.

Appendix B
I²C Communications

Maximite family only (not DOS or Generic PIC32 versions).

The Inter Integrated Circuit (I²C) bus was developed by the electronics giant Philips for the transfer of data between integrated circuits. It has been adopted by many manufacturers and can be used to communicate with many devices including memories, clocks, displays, speech module, etc. Using the I²C bus is complicated and if you do not need to communicate with these devices you can safely ignore this section.

This implementation was developed by Gerard Sexton and fully supports master and slave operation, 10 bit addressing, address masking and general call, as well as bus arbitration (ie, bus collisions in a multi-master environment).

In the master mode, there is a choice of 2 modes: interrupt and normal. In normal mode, the I2C send and receive commands will not return until the command completes or a timeout occurs (if the timeout option has been specified). In interrupt mode, the send and receive commands return immediately allowing other MMBasic commands to be executed while the send and receive are in progress. When the send/receive transactions have completed, an MMBasic interrupt will be executed. This allows you to set a flag or perform some other processing when this occurs.

When enabled the I²C function will take control of the external I/O pins 12 and 13 and override SETPIN and PIN() commands for these pins. Pin 12 becomes the I²C data line (SDA) and pin 13 the clock (SCL). Both of these pins should have external pullup resistors installed (typical values are 10KΩ for 100KHz or 2KΩ for 400 kHz). For the DuinoMite see the "DuinoMite MMBasic ReadMe.pdf" document for details.

Be aware that when running the I²C bus at above 150 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.

There are four commands for master mode; I2CEN, I2CDIS, I2CSEND and I2CRCV. For slave mode the commands are; I2CSEN, I2CSDIS, I2CSSEND and I2CSRCV. The master and slave modes can be enabled simultaneously; however, once a master command is in progress, the slave function will be "idle" until the master releases the bus. Similarly, if a slave command is in progress, the master commands will be unavailable until the slave transaction completes.

Both the master and slave modes use an MMBasic interrupt to signal a change in status. These interrupt routines operate the same as a general interrupt on an external I/O pin (see page 4 for a description) and must be terminated with an IRETURN command to return control to the main program when completed.

The automatic variable MM.I2C will hold the result of a command or action.

I2C Master Mode Commands

I2CEN speed, timeout [, int]	Enables the I²C module in master mode. 'speed' is a value between 10 and 400 (for bus speeds 10 kHz to 400 kHz). '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). 'int' is optional. It specifies the line number or label of an interrupt routine to be run when the send or receive command completes. If this is not supplied, the send and receive command will only return when they have completed or timed out. If it is supplied then the send and receive will complete immediately and the command will execute in the background.
I2CDIS	Disables the slave I²C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN. It will also send a stop if the bus is still held.

I2CSEND addr, option, sendlen, senddata [,sendata ....]	Send data to the I^2C slave device. 'addr' is the slave i2c address. 'option' is a number between 0 and 3 1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 2 = treat the address as a 10 bit address 3 = combine 1 and 2 (hold the bus and use 10 bit addresses). '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 in the command as individual bytes. Example: I2CSEND &H6F, 1, 3, &H23, &H43, &H25 • The data can be in a one dimensional array. The subscript does not have to be zero and will be honoured; also bounds checking is performed. Example: I2CSEND &H6F, 1, 3, ARRAY(0) • The data can be a string variable (not a constant). Example: I2CSEND &H6F, 1, 3, STRING$ The automatic variable MM.I2C will hold the result of the transaction.
I2CRCV addr, bus_hold, rcvlen, rcvbuf [,sendlen, senddata [,sendata ....]]	Receive data from the I^2C slave device with the optional ability to send some data first. 'addr' is the slave i2c address (note that 10 bit addressing is not supported). 'option' is a number between 0 and 3 1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 2 = treat the address as a 10 bit address 3 = combine 1 and 2 (hold the bus and use 10 bit addresses). 'rcvlen' is the number of bytes to receive. 'rcvbuf' is the variable to receive the data - this is a one dimensional array or if rcvlen is 1 then this may be a normal variable. The array subscript does not have to be zero and will be honoured; also bounds checking is performed. Optionally you can specify data to be sent first using 'sendlen' and 'senddata'. These parameters are used the same as in the I2CSEND command (ie, senddata can be a constant, an array or a string variable). Examples: I2CRCV &h6f, 1, 1, avar I2CRCV &h6f, 1, 5, anarray(0) I2CRCV &h6f, 1, 4, anarray(2), 3, &h12, &h34, &h89 I2CRCV &h6f, 1, 3, anarray(0), 4, anotherarray(0) The automatic variable MM.I2C will hold the result of the transaction.

I2C Slave Mode Commands

I2CSEN addr, mask, option, send_int, rcv_int	Enables the I^2C module in slave mode. 'addr' is the slave i2c address 'mask' is the address mask (bits set as 1 will always match) 'option' is a number between 0 and 3 1 = allows MMBasic to respond to the general call address. When this occurs the value of MM.I2C will be set to 4. 2 = treat the address as a 10 bit address 3 = combine 1 and 2 (respond to the general call address and use 10 bit addresses). 'send_int' is the line number or label of a send interrupt routine to be invoked when the module has detected that the master is expecting data. 'rcv_int is the line number or label of a receive interrupt routine to be invoked when the module has received data from the master.
I2CSDIS	Disables the slave I^2C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN.
I2CSSEND sendlen, senddata [,sendata ....]	Send the data to the I^2C master. This command should be used in the send interrupt (ie in the 'send_int_line' when the master has requested data). Alternatively a flag can be set in the send interrupt routine and the command invoked from the main program loop when the flag is set. 'sendlen is the number of bytes to send. 'senddata' is the data to be sent. This can be specified in various ways, see the I2CSEND commands for details.
I2CSRCV rcvlen, rcvbuf, rcvd	Receive data from the I^2C master device. This command should be used in the receive interrupt (ie in the 'rcv_int_line' when the master has sent some data). Alternatively a flag can be set in the receive interrupt routine and the command invoked from the main program loop when the flag is set. 'rcvlen' is the maximum number of bytes to receive. 'rcvbuf' is the variable to receive the data - this is a one dimensional array or if rcvlen is 1 then this may be a normal variable. The array subscript does not have to be zero and will be honoured; also bounds checking is performed. 'rcvd' will contain actual number of bytes received by the command.

I2C Automatic Variable

MM.I2C	Is set to indicate the result of an I2C operation. 0 = The command completed without error. 1 = Received a NACK response 2 = Command timed out 4 = Received a general call address (when in slave mode)

I2C Utility Command

NUM2BYTE number, array(x) or NUM2BYTE number, variable1, variable2, variable3, variable4	Convert 'number' to four numbers containing the four separate bytes of 'number' (MMBasic numbers are stored as the C float type and are four bytes in length). The bytes can be returned as four separate variables, or as four elements of 'array' starting at index 'x'. See the function BYTE2NUM() for the reverse of this command.

I2C Utility Function

BYTE2NUM(array(x)) or BYTE2NUM(arg1, arg2, arg3, arg4)	Return the number created by storing the four arguments as consecutive bytes (MMBasic numbers are stored as the C float type and are four bytes in length). The bytes can be supplied as four separate numbers (arg1 - arg4) or as four elements of 'array' starting at index 'x'. See the command NUM2BYTE for the reverse of this function.

Appendix C
1-Wire Communications

Maximite family only (not DOS or Generic PIC32 versions).

The 1-Wire protocol was invented by Dallas Semiconductor to communicate with chips using a single signalling line. It is mostly used in communicating with the DS18B20 and DS18S20 temperature measuring chips. This implementation was developed for MMBasic by Gerard Sexton.

There are four commands that you can use:

OWRESET pin [,presence]

OWWRITE pin, flag, length, data [, data…]

OWREAD pin, flag, length, data [, data…]

OWSEARCH pin, srchflag, ser [,ser…]

Where:

pin - the MMBasic I/O pin to use

presence - an optional variable to receive the presence pulse (1 = device response, 0 = no device)

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 receive

srchflag - a combination of the following options:

- 1 - start a new search
- 2 - only return devices in alarm state
- 4 - search for devices in the requested family (first byte of ser)
- 8 - skip the current device family and return the next device
- 16 - verify that the device with the serial number in ser is available

If srchflag = 0 (or 2) then the search will return the next device found

ser - serial number (8 bytes) will be returned (srchflag 4 and 16 will also use the values in ser)

After the command is executed, the pin will be set to the not configured state unless flag option 8 is used. The data and ser arguments can be a string, array or a list of variables.

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

There are two utility functions available:

OWCRC8(len, cdata [, cdata…]) Processes the cdata and returns the 8 bit CRC

OWCRC16(len, cdata [, cdata…]) Processes the cdata and returns the 16 bit CRC

Where:

len - length of data to process

cdata - data to process

The cdata can be a string, array or a list of variables

Appendix D
SPI Communications

Maximite family only (not DOS or Generic PIC32 versions).

The Serial Peripheral Interface (SPI) communications protocol is used to send and receive data between integrated circuits.

The SPI function in MMBasic acts as the master (ie, MMBasic generates the clock).

The syntax of the function is:

```
received_data = SPI( rx, tx, clk, data_to_send, speed, mode, bits )
```

Data_to_send, speed, mode and bits are all optional. If not required they can be represented by either empty space between the commas or left off the end of the list.

Where:

- 'rx' is the pin number for the data input (MISO)
- 'tx' is the pin number for the data output (MOSI)
- 'clk' is the pin number for the clock generated by MMBasic (CLK)
- 'data_to_send' is optional and is an integer representing the data byte to send over the output pin. If it is not specified the 'tx' pin will be held low.
- 'speed' is optional and is the speed of the clock. It is a single letter either H, M or L where H is 3 MHz, M is 500 KHz and L is 50 KHz. Default is H.
- 'mode' is optional and is a single numeric digit representing the transmission mode – see Transmission Format below. The default mode is 3.
- 'bits' is optional and represents the number of bits to send/receive. Range is 1 to 23 (this limit is defined by how many bits can be stored in a floating point number). The default is 8.

The SPI function will return the data received during the transaction as an integer. Note that a single SPI transaction will send data while simultaneously receiving data from the slave (which is often discarded).

Examples

Using all the defaults:

```
A = SPI(11, 12, 13)
```

Specifying the data to be sent:

```
A = SPI(11, 12, 13, &HE4)
```

Setting the mode but using the defaults for data to send and speed:

```
A = SPI(11, 12, 13, , , 2)
```

An example specifying everything including a 12 bit data transfer:

```
A = SPI(11, 12, 13, &HE4, M, 2, 12)
```

Transmission Format

The most significant bit is sent and received first. The format of the transmission can be specified by the 'mode' as follows:

Mode	CPOL	CPHA	Description
0	0	0	Clock is active high, data is captured on the rising edge and output on the falling edge
1	0	1	Clock is active high, data is captured on the falling edge and output on the rising edge
2	1	0	Clock is active low, data is captured on the falling edge and output on the rising edge
3	1	1	Clock is active low, data is captured on the rising edge and output on the falling edge

For a more complete explanation see: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

I/O Pins

Before invoking this function the 'rx' pin must be configured as an input using the SETPIN command and the 'tx' and 'clk' pins must be configured as outputs (either normal or open collector) again using the SETPIN command. The clock pin should also be set to the correct polarity (using the PIN function) before the SETPIN command so that it starts as inactive.

The SPI enable signal is often used to select a slave and "prime" it for data transfer. This signal is not generated by this function and (if required) should be generated using the PIN function on another pin.

The SPI function does not "take control" of the I/O pins like the serial and I^2C protocols and the PIN command will continue to operate as normal on them. Also, because the I/O pins can be changed between function calls it is possible to communicate with many different SPI slaves on different I/O pins.

Example

The following example will send the command &H80 and receive two bytes from the slave SPI device. Because the mode, speed and number of bits are not specified the defaults are used.

```
SETPIN 18, 2                    ' set rx pin as a digital input
SETPIN 19, 8                    ' set tx pin as an output
PIN(20) = 1 : SETPIN 20, 8      ' set clk pin high then set it as an output
PIN(11) = 1 : SETPIN 11, 8      ' pin 11 will be used as the enable signal

PIN(11) = 0                     ' assert the enable line (active low)
junk = SPI(18, 19, 20, &H80)    ' send the command and ignore the return
byte1 = SPI(18, 19, 20)         ' get the first byte from the slave
byte2 = SPI(18, 19, 20)         ' get the second byte from the slave
PIN(11) = 1                     ' deselect the slave
```

Appendix E
Loadable Fonts

Maximite family only (not DOS or Generic PIC32 versions).

This section describes the format of a font file that can be loaded using the FONT LOAD command.

A font file is just a text file containing ordinary characters which are loaded line by line to build the bitmap of each character in the font. Each character can be up to 64 pixels high and 255 pixels wide. Up to 255 characters can be defined.

The first non-comment line in the file must be the specifications for the font as follows:

```
height, width, start, end
```

Where 'height' and 'width' are the size of each character in pixels, 'start' is the number in the ASCII chart where the first character sits and 'end' is the last character. Each number is separated by a comma. So, for example, `16, 11, 48, 57` means that the font is 16 pixels high and 11 wide. The first character is decimal 48 (the zero character) and the last is 57 (number nine character).

The remainder of the lines specify the bitmap for each character.

Each line represents a horizontal row of pixels. A space means the pixel is not illuminated and any other character will turn the pixel on. If the font is 11 pixels wide there must be 11 characters in the line although trailing spaces can be omitted. The first line is the top row of pixels in the character, the next is the second and so on. If the character is 16 pixels high there must be 16 lines to define the character. This repeats until each character is drawn. Using the above example of a font 16x11 with 10 characters there must be a total of 160 lines with each line 11 characters wide. This is in addition to the specification line at the top.

A comment line has an apostrophe (') as the first character and can occur anywhere. A comment line is completely ignored; all other lines are significant.

The following example creates two small icons; a smiley face and a frowning face. Each is 11x11 pixels with the first (the smiley face) in the position of the zero character (0) and the frowning face in the position of number one (1). To display a smiley face your program would contain this:

```
40 FONT LOAD "FACES.FNT" AS #6  ' load the font
50 FONT #6                      ' select the font
60 PRINT "0"                    ' print a smiley face
```

```
' example
' FACES.FNT
11,11,48,49

     XXX
   XX   XX
  XX     XX
XX  X X  XX
X         X
XX X   X XX
 X  XXX  X
   XX   XX
     XXX


     XXX
   XX   XX
  XX     XX
XX  X X  XX
X         X
XX  XXX  XX
 X X   X X
   XX   XX
     XXX
```

Appendix F
Special Keyboard Keys

Maximite family only (not DOS or Generic PIC32 versions).
MMBasic generates a single unique character for the function keys and other special keys on the keyboard. These are shown in the table as hexadecimal and decimal numbers:

Keyboard Key	Key Code (Hex)	Key Code (Decimal)
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
Num Lock	8C	140
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

If the control key is simultaneously pressed then 20 (hex) is added to the code (this is the equivalent of setting bit 5). If the shift key is simultaneously pressed then 40 (hex) is added to the code (this is the equivalent of setting bit 6). If both are pressed 60 (hex) is added. For example Control-PageDown will generate A9 (hex). The shift modifier only works with the function keys F1 to F12; it is ignored for the other keys.

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. This is particularly useful when using the EDIT command.

Appendix G
Tera Term Setup

Maximite family only.

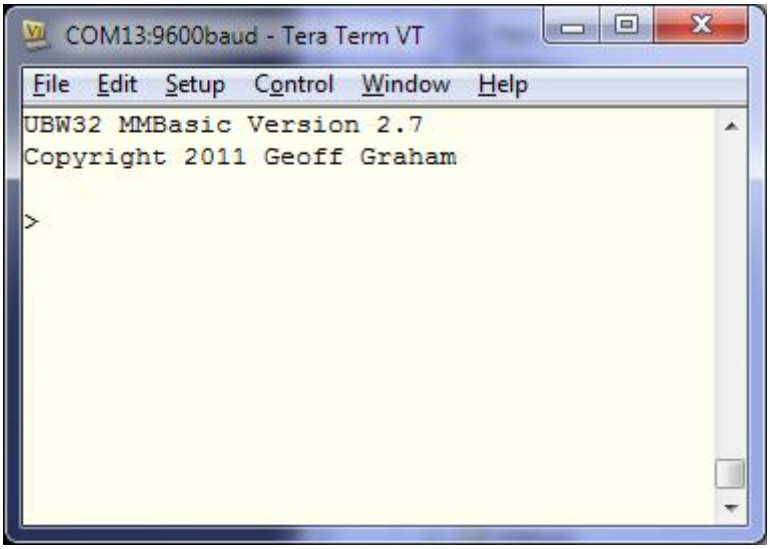
MMBasic creates a virtual serial port over USB so that you can communicate with it from a Windows, Linux or Macintosh computer using nothing more than the USB port.

The communications protocol used is the CDC (Communication Device Class) protocol and there is native support for this in Linux (the cdc-acm driver) and Apple OS/X. Macintosh users can refer to the document "Using Serial Over USB on the Macintosh" on http://geoffg.net/maximite.html. The rest of this tutorial assumes that you are using a computer running Windows XP, Vista or 7.

First you need to install the Windows Serial Port Driver (available from http://geoffg.net/maximite.html). Full instructions are included in the download and when you have finished you should see the connection in Device Manager as a numbered communications port (ie, COM13).

To communicate with MMBasic over this virtual serial port you need to use a terminal emulator. This is a program that emulates the old fashioned VT100 terminal over a serial communications link. There are quite a few free emulators that you can use but I recommend Tera Term.

1. You should download Tera Term from http://en.sourceforge.jp/projects/ttssh2/releases/ and install it. These instructions are based on version 4.71.
2. Make sure that the USB cable is plugged into your PC and that you know the COM number.
3. When you run Tera Term for the first time you will get a dialog box asking you to select the type of connection. Select serial and select the correct COM number. You should then see the MMBasic prompt as shown below:



4. Before you start using Tera Term you need to make a few changes to the setup:

   Select **Setup -> Terminal…**

   Set the terminal size to 80 x 36.

   Untick the tick box labelled "term size = win size".

   Tick the box labelled "auto window resize".

   Select **Setup -> Serial Port…**

   Make sure that the port matches the COM number representing the Maximite.

   In the box for the "transmit delay msec/line" enter 50. Leave the other box with zero in it.

   You do not have to bother with the baud rate or any other settings.

   Select **Setup -> Save Setup…**

   Save the setup as TERATERM.INI in the Tera Term installation directory overwriting the file there.

Appendix H
Sprite Definition Files

Maximite family only (not DOS or Generic PIC32 versions).

This section describes the format of a sprite file that can be loaded using the SPRITE LOAD command.

A sprite file is similar to a font file except that it contains the definition of sprites which are 16x16 bit graphical objects that can be moved about on the video screen without disturbing background text of graphics. The sprite file is just a text file containing ordinary characters which are loaded line by line to build the bitmap of each sprite. Currently the dimensions of each sprite are fixed at 16x16 bits although alternative sizes may be allowed in the future.

The first non-comment line in the file must be the specifications for the sprite file as follows:

```
dimension, number
```

Where 'dimension' is the height and width of the sprites in pixels. At this time it must be the number 16. 'number' is the number of sprites in the file and is limited only by the amount of free memory available. The remainder of the lines specify the bitmap for each sprite.

Each line represents a horizontal row of pixels with each character in the line defining the colour of the pixel. The character can be a single numeric digit in the range of 0 to 7 representing the colours black to white or it can be a space which means that that particular pixel will be transparent (ie, the background will show through).

Each sprite must immediately follow the preceding sprite in the file and be defined by 16 lines each of 16 characters wide (although trailing spaces can be omitted and will be assumed to be transparent pixels).

A comment line has an apostrophe (') as the first character and can occur anywhere. A comment line is completely ignored; all other lines are significant.

The following example is of a file that contains a single sprite consisting of a red ball with a white border and a blue centre dot:

```
' example sprite
' TEST.SPR
16, 1
      7777
    74444447
  744444444447
 7444444444447
 7444444444447
7444444444444447
7444444114444447
7444441111444447
7444441111444447
7444444114444447
7444444444444447
 7444444444447
 7444444444447
  744444444447
    74444447
      7777
```