MMBasic Language Manual
Ver 3.1

Geoff Graham

For updates to this manual and more details on MMBasic
go to http://geoffg.net/maximite.html

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 (all still based on the PIC32).

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

Maximite and mini Maximite: Maximite Hardware Manual   from: http://geoffg.net/maximite.html.
UBW32 experimenter board: UBW32 MMBasic Manual   from: http://geoffg.net/ubw32.html.
CGMMSTICK1 board: http://www.circuitgizmos.com/products/cgmmstick1/cgmmstick1.shtml
DuinoMite hardware: http://olimex.com

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).

To edit programs you use the EDIT command, which will invoke the full screen editor. You can use the arrow keys to move around the program and keys such as Home, End, Page Up and Down will do what you expect. The editor will allow you to insert blank lines and indent lines for improved readability. It will work using an attached video screen (VGA or composite) and over USB with a VT100 compatible terminal emulator (Tera Term is recommended).

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

Program and Data Storage

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 startup MMBasic will look for a file 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.

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"

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.

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. A high performance Pulse Width Modulation (PWM) output is also available by reusing the sound connector and specifying an optional duty cycle parameter to the SOUND command.

The Dallas 1-Wire protocol is also supported. See Appendix C for details.

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. If not set the calendar will start from midnight 1st Jan 2000 on power up.

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 you can setup a "tick" which will generate a regular interrupt with a period from one millisecond to over a month. See Interrupts below.

Interrupts

Any external I/O pin can be configured to generate an interrupt using the SETPIN command with up to 21 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.

Graphics

Graphics commands operate on the video output. 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 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 next PRINT output using LOCATE and the SAVEBMP command will save the current screen as a BMP file.

The colour parameter for these commands can be zero to turn a pixel off, 1 to turn the pixel on or -1 to invert a pixel. This last function is useful in animation.

Sound

The SOUND command will generate a simple square wave between 20Hz and 1MHz lasting for a specified duration. This command can also be used to generate a Pulse Width Modulation (PWM) signal.

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.

SUB and END sub make it easy to add your own commands to MMBasic. For example:

```
SUB REVERSE ( str$ )
   LOCAL x
   FOR x = LEN(str$) TO 0 STEP -1
   PRINT MID$(str$, x, 1);
   NEXT x
END SUB
```

You have now added the new command REVERSE to your program and if you used it in a program like this:

```
REVERSE "Hello World"
```

It would print the following on the screen:

```
dlroW olleH
```

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
```

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 that are available.

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.

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.

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 212KB.

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

License

MMBasic is Copyright 2011, 2012 Geoff Graham - http://geoffg.net

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 for version 2.7 is available for download under the GNU GPL 3. The source to versions 3.0 and later is available via subscription (free of charge) to individuals for personal use. In both cases go to http://geoffg.net/maximite.html for details.

Predefined Read Only Variables

MM.HRES	The horizontal resolution of the current video display screen in pixels.
MM.VRES	The vertical resolution of the current video display screen in pixels.
MM.VER	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.DRIVE$	The current default drive returned as a string containing either "A:" or "B:".
MM.FNAME$	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	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. 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

' (single quotation mark)	Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.
? (question mark)	Shortcut for the PRINT command.
AUTO or AUTO start or AUTO start, increment	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. 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.
CHDIR dir$	Change the current working directory on the SD card to 'dir$' The special entry ".." represents the parent of the current directory and "." represents the current directory.
CIRCLE (x, y) ,r [,c [,F]]	Draws a circle on the video output centred at x and y with a radius of r. If c is zero the pixels are turned off, if it is 1 or not specified the pixels are turned on (ie, the circle is drawn) and -1 will invert the pixels. The F option will cause the circle to be filled according to the c parameter. See page 5 for a definition of the graphics coordinates. Note that because the pixels are not exactly square the circle will be oval to some degree.
CLEAR	Delete all variables and recover the memory used by them. See ERASE for deleting specific array variables.
CLOSE [#]nbr [,[#]nbr] …	Close the file(s) or serial port(s) previously opened with the file number 'nbr'. The # is optional. Also see the OPEN command.
CLOSE CONSOLE	Close a serial port that had been previously opened as the console.
COPY src$ TO dest$	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. Note that it is not possible to have both the source and destination on the internal flash drive A:.
CLS	Clears the video display screen and places the cursor in the top left corner.

CONFIG COMPOSITE NTSC | PAL or CONFIG VIDEO OFF | ON or CONFIG CASE UPPER|LOWER|TITLE or CONFIG KEYBOARD US|FR|GR or CONFIG TAB 2 | 4 | 8	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 slight performance improvement with the video off but the biggest benefit is that the unused memory is returned to the general memory pool used by strings and arrays. 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) or German (GR) 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	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.
COPYRIGHT	List all contributors to the MMBasic and summarise the copyright.
DATA constant[,constant]...	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"	Set the date of the internal clock/calendar. DD, MM and YY are numbers, for example: DATE$ = "28-2-2011" The date is set to "1-1-2000" on power up.
DELETE line DELETE -lastline DELETE firstline - DELETE firstline - lastline	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...)]...	Specifies variables that have more than one element in a single dimension, i.e., arrayed variables.
DO <statements> LOOP	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	Loops while "expression" is true (this is equivalent to the older WHILE-WEND loop, also implemented in MMBasic).

DO <statements> LOOP UNTIL expression	Loops until the expression following UNTIL is true.
DRIVE drivespec$	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	Invoke the full screen editor. This can be used to edit either the program currently loaded in memory or a file on either drive A: or B:. 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:

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	Introduces a default condition in a multiline IF statement. See the multiline IF statement for more details.
ELSEIF expression THEN	Introduces a secondary condition in a multiline IF statement. See the multiline IF statement for more details.
ENDIF	Terminates a multiline IF statement. See the multiline IF statement for more details.
END	End the running program and return to the command prompt.
END SUB	Marks the end of a user defined subroutine. See the SUB command. Each sub must have one and only one matching END SUB statement. Use EXIT SUB if you need to return from a subroutine from within its body. Only one space is allowed between END and SUB.
ERASE variable [,variable]...	Deletes arrayed variables and frees up the memory. Use CLEAR to delete all variables including all arrayed variables.
ERROR [error_msg$]	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 SUB	EXIT by itself will terminate a DO...LOOP EXIT FOR will terminate a FOR...NEXT loop. EXIT SUB will exit a subroutine (same effect as an END SUB). Only one space is allowed between the two words.

FILES [fspec$]	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	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, comma and full stop.

	Examples: 10 FONT #3, 2, 1 ' double scale and reverse 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	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. The font is loaded into the general memory area used by arrays and strings. Use the MEMORY command to check the usage of this area.
FONT UNLOAD #nbr	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]	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.
GOSUB target	Initiates a subroutine call to the target, which can be a line number or a label. The subroutine must end with RETURN.
GOTO target	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	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
	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$";] list of variables	Allows input from the keyboard to a list of variables. The input command will prompt with a question mark (?). The input must contain commas to separate each data item if there is more than one variable. For example, if the command is: INPUT a, b, c And the following is typed on the keyboard: 23, 87, 66 Then a = 23 and b = 87 and c = 66 If the "prompt string$" 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	Same as above except that the input is read from a file previously opened for INPUT as 'nbr'. See the OPEN command.
IRETURN	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$	Deletes the file specified by 'file$'. Quote marks are required around a string constant and the extension, if there is one, must be specified. Example: KILL "SAMPLE.DAT"
LET variable = expression	Assigns the value of 'expression' to the variable. LET is automatically assumed if a statement does not start with a command.
LOCAL, variable [, variables]	Defines a list of variable names as local to the subroutine or function. 'variable' can be an array and the array will be dimensioned just as if the DIM command had been used. A local variable will only be visible within the procedure and will be deleted (and the memory reclaimed) when the procedure returns. If the local variable has the same name as a global variable (used before any subroutines or functions were called) the global variable will be hidden by the local variable while the procedure is executed.
LINE [(x1 , y1)] - (x2, y2) [,c [,B[F]]]	Draws a line or box on the video screen. x1,y1 and x2,y2 specify the beginning and end points of a line. c specifies the displayed pixel (0 = off, 1 = on, -1 = invert) and defaults to on if not specified. (x1, y1) is optional and if omitted the last drawing point will be used. The optional B will draw a box with the points (x1,y1) and (x2,y2) at opposite corners. The optional BF will draw a box (as ,B) and fill the interior. See page 5 for a definition of the graphics coordinates.
LINE INPUT [prompt$,] string-variable$	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$'.
LINE INPUT #nbr, string-variable$	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	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$	Loads a program called 'file$' from the current drive into working 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.
LOOP [UNTIL expression]	Terminates a program loop: see DO.

MEMORY	List the amount of memory currently in use. For example:
	
	  5kB (17%) Program memory used
	  3kB (16%) Variable memory used
	12kB (30%) General memory used
	
	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 for arrays, strings, fonts, file I/O buffers, etc.

MERGE file$	Adds program lines from 'file$' to the program in memory. Unlike LOAD, it does not clear the program currently in memory.
MKDIR dir$	Make, or create, the directory 'dir$' on the SD card.
NAME old$ AS new$	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	Deletes the program in memory and clears all variables.
NEXT [counter-variable] [, counter-variable], etc	NEXT comes at the end of a FOR-NEXT loop; see FOR. The 'counter-variable' specifies exactly which loop is being operated on. If no 'counter-variable' is specified the NEXT will default to the innermost loop. It is also possible to specify multiple counter-variables as in: NEXT x, y, z
ON variable GOTO|GOSUB target[,target, target,...]	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of variable; 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	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 option will act the same as the OUTPUT mode (i.e. the file is created then opened for writing). '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	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 (13 and 14 in the DuinoMite).
	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	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	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$	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$	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	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	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	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	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) = value	Set a pixel on the VGA or composite screen off (if value is zero) or on (if value is 1) or inverted (if the value is -1). See page 5 for a definition of the graphics coordinates. See the function PIXEL(x,y) for obtaining the value of a pixel.
POKE hiword, loword, val	Will set a byte within the PIC32 virtual memory space to 'val'. 'hiword' is the top 16 bits of the address while 'loword' is the bottom 16 bits. 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 while the source code will provide the symbolic names used in the firmware and the Maximite.map file (produced after a successful compile) will list the addresses of these symbols. These addresses will change with each version of the firmware so programs should use the predefined variable MM.VER to determine the currently running version. WARNING: 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	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
	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 (ie, black letters, white background)
	  m = 5 Current pixels will be inverted (transparent background)

PRINT #nbr, expression [[,;]expression] … etc	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	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.

RANDOMIZE nbr	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]...	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	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	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	Resets the line and position counters for DATA and READ statements to the top of the program file.
RETURN	RETURN concludes a subroutine called by GOSUB.
RMDIR dir$	Remove, or delete, the directory 'dir$' on the SD card.
RUN [line] [file$]	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$) is supplied, the current program will be erased and that program will be loaded from the current drive and executed. This enables one program to load and run another. Example: RUN "TEST.BAS" If an extension is not specified ".BAS" will be added to the file name.
SAVE [file$]	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$	Saves the current VGA or composite screen as a BMP file in the current working directory on the current drive. 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 the image. This appears to be a bug in Paint as all other software tested (including Windows XP Paint) can display the image without fault.

SETPIN pin, cfg	Will configure the external I/O 'pin' according to 'cfg':
	
	  0 Not configured or inactive
	  1 Analog input (pins 1 to 10)
	  2 Digital input (all pins, 5V tolerant on pins 11 and above)
	  3 Frequency input (pins 11 to 14)
	  4 Period input (pins 11 to 14)
	  5 Counting input (pins 11 to 14)
	  6 Interrupt on low to high input change (all pins)
	  7 Interrupt on high to low input change (all pins)
	  8 Digital output (all pins)
	  9 Open collector digital output to 5V (pins 11 and above)
	         In this mode the function PIN() will also return the output value.
	
	In the Maximite implementation the highest pin number is 20 and in the UBW32 version it is 50. Note that on the UBW32 pin 48 is shared with the white LED, pin 49 is shared with the green LED and the program pushbutton is connected to pin 50.
	
	In the DuinoMite version pins 8 to 12 as well as 19 and 20 are unavailable.
	
	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	Will configure 'pin' to generate an interrupt according to 'cfg':
	
	  0 Not configured or inactive
	  6 Interrupt on low to high input change (all pins)
	  7 Interrupt on high to low input change (all pins)
	
	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	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 'line' to zero (ie, SETTICK 0, 0).
SOUND freq, dur or SOUND freq, dur, duty	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 zero, any active SOUND statement is turned off. If no SOUND statement is running, a 'dur' of zero has no effect. 'duty' is the optional duty cycle of the waveform in percent. If it is close to zero the output will be a narrow positive pulse, if 50 a square wave will be

	generated and if close to 100 it will be a very wide positive pulse. If it is not specified the duty cycle will default to 50%.
	Setting the duty cycle allows the sound output to be used as a Pulse Width Modulation (PWM) output for driving analogue circuits. The signal will be available at the sound connector and the voltage divider on this output should be removed so that the full signal level is available. 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%.
SUB xxx (arg1 [,arg2, …]) <statements> <statements> END SUB	Defines a callable subroutine. This is the same as adding a new command to MMBasic while it is running your program. 'xxx' is the subroutine name and it must meet the specifications for naming a variable. 'arg1', 'arg2', etc are the arguments or parameters to the subroutine. You use the subroutine by using its name and arguments in a program just as you would a normal MMBasic command. For example: xxx carg1, carg2, etc 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. When the END SUB or EXIT SUB statement is reached the program returns to the next statement after the call to 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.
TIME$ = "HH:MM:SS" or TIME$ = "HH:MM" or TIME$ = "HH"	Sets the time of the internal clock. MM and SS are optional and will default to zero if not specified. For example TIME$ = "14:30" will set the clock to 14:30 with zero seconds. The time is set to "00:00:00" on power up.
TIMER = msec	Resets the timer to a number of milliseconds. Normally this is just used to reset the timer to zero but you can set it to any positive integer. See the TIMER function for more details.
TROFF	Turns the trace facility off; see TRON.
TRON	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$	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 30 seconds. Download Tera Term from http://ttssh2.sourceforge.jp/

Functions

ABS(number)	Returns the absolute value of the argument 'number' (ie, any negative sign is removed and the positive number is returned).
ASC(string$)	Returns the ASCII code for the first letter in the argument 'string$'.
ATN(number)	Returns the arctangent value of the argument 'number' in radians.
BIN$(number)	Returns a string giving the binary (base 2) value for the 'number'.
CHR$(number)	Returns a one-character string consisting of the character corresponding to the ASCII code indicated by argument 'number'.

CINT(number)	Round numbers with fractional portions up or down to the next whole number or integer.
	
	For example, 45.47 will round to 45
	                   45.57 will round to 46
	                   -34.45 will round to -34
	                   -34.55 will round to -35
	
	See also INT() and FIX().

COS(number)	Returns the cosine of the argument 'number' in radians.
CWD$	Returns the current working directory as a string.
DATE$	Returns the current date based on MMBasic's internal clock as a string in the form "DD-MM-YYYY". For example, "28-02-2011". The internal clock/calendar will keep track of the time and date including leap years. The date is set to "1-1-2000" on power up. To set the date use the command DATE$ =.
DEG(radians)	Converts 'radians' to degrees.

DIR$(fspec, type)
or
DIR$(fspec)
or
DIR$( )
	Will search an SD card for files and return the names of entries found.
	
	'fspec' is a file specification using wildcards the same as used by the FILES command. Eg, "*.*" will return all entries, "*.TXT" will return text files.
	
	'type' is the type of entry to return and can be one of:
	    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 function only operates on the SD card and you must change to the required directory before invoking it.

EOF([#]nbr)	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)	Returns the exponential value of 'number'.
FIX(number)	Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point. For example 9.89 will return 9 and -2.11 will return -2. The major difference between FIX and INT is that FIX provides a true integer function (ie, does not return the next lower number for negative numbers as INT() does). This behaviour is for Microsoft compatibility. See also CINT() .

FORMAT$(nbr [, fmt$])	Will return a string representing 'nbr' formatted according to the specifications in the string 'fmt$'.
	
	The format specification starts with a % character and ends with a letter. Anything outside of this construct is copied to the output as is.
	
	The structure of a format specification is:
	
	       % [flags] [width] [.precision] type
	
	Where 'flags' can be:
	
	    - Left justify the value within a given field width
	    0 Use 0 for the pad character instead of space
	    + Forces the + sign to be shown for positive numbers
	    space Causes a positive value to display a space for the sign. Negative values still show the – sign
	
	'width' is the minimum number of characters to output, less than this the number will be padded, more than this the width will be expanded.
	
	'precision' specifies the number of fraction digits to generate with an e, or f type or the maximum number of significant digits to generate with a g type. If specified, the precision must be preceded by a dot (.).
	
	'type' can be one of:
	
	    g Automatically format the number for the best presentation.
	    f Format the number with the decimal point and following digits
	    e Format the number in exponential format
	
	If uppercase G or F is used the exponential output will use an uppercase E. If the format specification is not specified "%g" is assumed.
	
	Examples:
	       format$(45) will return 45
	       format$(45, "%g") will return 45
	       format$(24.1, "%g") will return 24.1
	       format$(24.1,"%f") will return 24.100000
	       format$(24.1, "%e") will return 2.410000e+01
	       format$(24.1,"%09.3f") will return 00024.100
	       format$(24.1,"%+.3f") will return +24.100
	       format$(24.1,"**%-9.3f**") will return **24.100 **

HEX$(number)	Returns a string giving the hexadecimal (base 16) value for the 'number'.
INKEY$	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$(nbr, [#]fnbr)	Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number 'fnbr'. This function will read all characters including carriage return and new line without translation. When reading from a serial communications port this will return as many characters as are waiting in the receive buffer up to 'nbr'. If there are no characters waiting it will immediately return with an empty string. The # is optional. Also see the OPEN command.
INSTR([start-position,] string-searched$, string-pattern$)	Returns the position at which 'string-pattern$' occurs in 'string-searched$', beginning at 'start-position'.
INT(number)	Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behaviour is for Microsoft compatibility, the FIX() function provides a true integer function. See also CINT() .
LEFT$(string$, number-of-chars)	Returns a substring of 'string$' with 'number-of-chars' from the left (beginning) of the string.
LEN(string$)	Returns the number of characters in 'string$'.
LOC([#]nbr)	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)	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)	Returns the natural logarithm of the argument 'number'.
LCASE$(string$)	Returns 'string$' converted to lowercase characters.
MID$(string$, start-position-in-string[, number-of-chars])	Returns a substring of 'string$' 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)	Returns a string giving the octal (base 8) representation of 'number'.

PEEK(hiword, loword)	Will return a byte within the PIC32 virtual memory space. 'hiword' is the top 16 bits of the address while 'loword' is the bottom 16 bits.
	See the POKE command for notes and warnings related to memory access.
PI	Returns the value of pi.
PIN(pin)	Returns the value on the external I/O 'pin'. Zero means digital low, 1 means digital high and for 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	Returns the current cursor position in the line in characters.
PIXEL(x, y)	Returns the value of a pixel on the VGA or composite screen. Zero is off, 1 is on. See page 7 for a definition of the graphics coordinates.
	See the statement PIXEL(x,y) = for setting the value of a pixel.
RAD(degrees)	Converts 'degrees' to radians.
RIGHT$(string$, number-of-chars)	Returns a substring of 'string$' with 'number-of-chars' from the right (end) of the string.
RND(number)	Returns a pseudo-random number in the range of 0 to 0.99999. The 'number' value is ignored if supplied. The RANDOMIZE command reseeds the random number generator.
SGN(number)	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN(number)	Returns the sine of the argument 'number' in radians.
SPACE$(number)	Returns a string of blank spaces 'number' bytes long.
SPI(rx, tx, clk[, data[, speed]])	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)	Returns the square root of the argument 'number'.
STR$(number)	Returns a string in the decimal (base 10) representation of 'number'.
STRING$(number, ascii-value\|string$)	Returns a string 'number' bytes long consisting of either the first character of string$ or the character representing the ASCII value ascii-value.
TAB(number)	Outputs spaces until the column indicated by 'number' has been reached.
TAN(number)	Returns the tangent of the argument 'number' in radians.
TIME$	Returns the current time based on MMBasic's internal clock as a string in the form "HH:MM:SS" in 24 hour notation. For example, "14:30:00". The internal clock/calendar will keep track of the time and date including leap years. The time is set to midnight on power up. To set the time use the command TIME$.
TIMER	Returns the elapsed time in milliseconds (eg, 1/1000 of a second) since reset. If not specifically reset this count will wrap around to zero after 49 days. The timer is reset to zero on power up and you can also reset it by using TIMER as a command.
UCASE$(string$)	Returns 'string$' converted to uppercase characters.
VAL(string$)	Returns the numerical value of the 'string$'. If 'string$' is an invalid number the function will return zero. This function will recognise the &H prefix for a hexadecimal number, &O for octal and &B for binary.

Obsolete Commands and Functions

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

IF condition THEN linenbr	For Microsoft compatibility a GOTO is assumed if the THEN statement is followed by a number. A label is invalid in this construct. New programs should use: IF condition THEN GOTO linenbr | label
LOCATE x, y	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)	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.
SPC(number)	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	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] …	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

Two serial ports are available for asynchronous serial communications. They are labelled COM1: and COM2: and are opened in a manner similar to opening a file. After they have been 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. Finally the serial port can be 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:
- 'baud' is the baud rate, either 19200, 9600, 4800, 2400, 1200, 600, 300, 150 or 75 bits per second. Default is 1200.
- 'buf' is the buffer sizes in bytes. Two of these buffers will be allocated from the general memory area, 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 and pin 16 for transmit data. If flow control is specified pin 17 will be used for RTS (receive flow control) and pin 18 will be CTS (transmit flow control).

COM2: uses pin 19 for receive data and pin 20 for transmit data (13 and 14 in the DuinoMite).

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

The Inter Integrated Circuit (I^{2}C) 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 I^{2}C 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 I^{2}C 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 I^{2}C 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). Refer to the diagrams at the start of this manual for the location of pins 12 and 13 on the external I/O connector.

Be aware that when running the I^{2}C 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 I^{2}C 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 I^{2}C 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 I^{2}C 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^{2}C 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^{2}C 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^{2}C 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^{2}C 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^{2}C 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^{2}C 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

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 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

The Serial Peripheral Interface (SPI) communications protocol is used to send and receive data between integrated circuits. As implemented this function is suitable for moving small amounts of data to and from a chip like an accelerometer, but not for shifting large amounts of data from EEPROMS, etc. The SPI function in MMBasic acts as the master (ie, MMBasic generates the clock).

The syntax of the SPI function is:

```
received_data = SPI( rx, tx, clk [, data_to_send [, speed ] ] )
```

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 500 kHz, M is 50 kHz and L is 5 kHz. Default is H.

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

Transmission Format

The format of the transmission matches the most common standard. The clock is high when inactive and the data is valid on the clock's trailing edge (ie, low to high transition). Data bytes are 8 bits, high voltage is logic 1 and the most significant bit is sent first.

In SPI parlance the MMBasic implementation of SPI has CPOL = 1 and CPHA = 1 or it operates in mode 3.

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). The clock pin should also be set high (using the PIN function) before the SETPIN command so that it starts as inactive (ie, high).

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 <u>not</u> "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 speed is not specified it defaults to high (500KHz):

```
 10 SETPIN 18, 2              ' set rx pin as a digital input
 20 SETPIN 19, 8              ' set tx pin as an output
 30 PIN(20) = 1 : SETPIN 20, 8   ' set clk pin high then set it as an output
 40 PIN(11) = 1 : SETPIN 11, 8   ' pin 11 will be used as the enable signal
 50 '
 60 PIN(11) = 0               ' assert the enable line (active low)
 70 junk = SPI(18, 19, 20, &H80)  ' send the command and ignore the return
 80 byte1 = SPI(18, 19, 20)   ' get the first byte from the slave
 90 byte2 = SPI(18, 19, 20)   ' get the second byte from the slave
100 PIN(11) = 1              ' deselect the slave
```

Appendix E
Loadable Fonts

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:

<div align="center">

`height, width, start, end`

</div>

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

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

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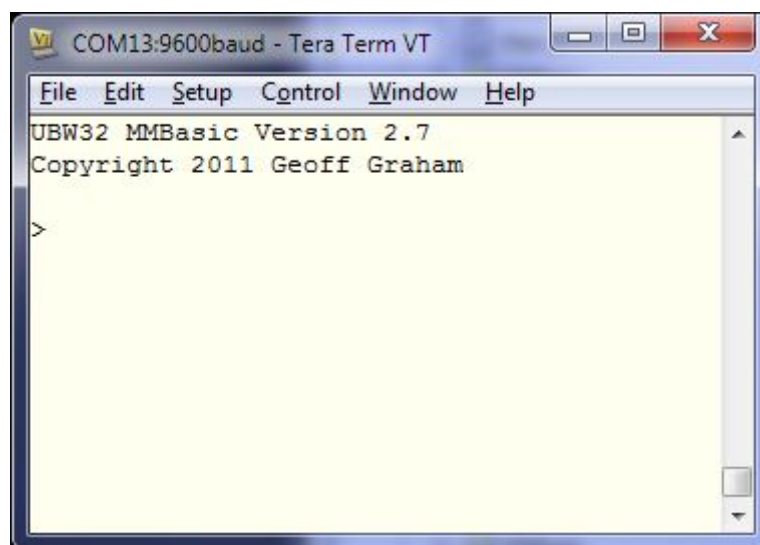
4. Before you start using Tera Term you need to make a few changes to the setup:

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.