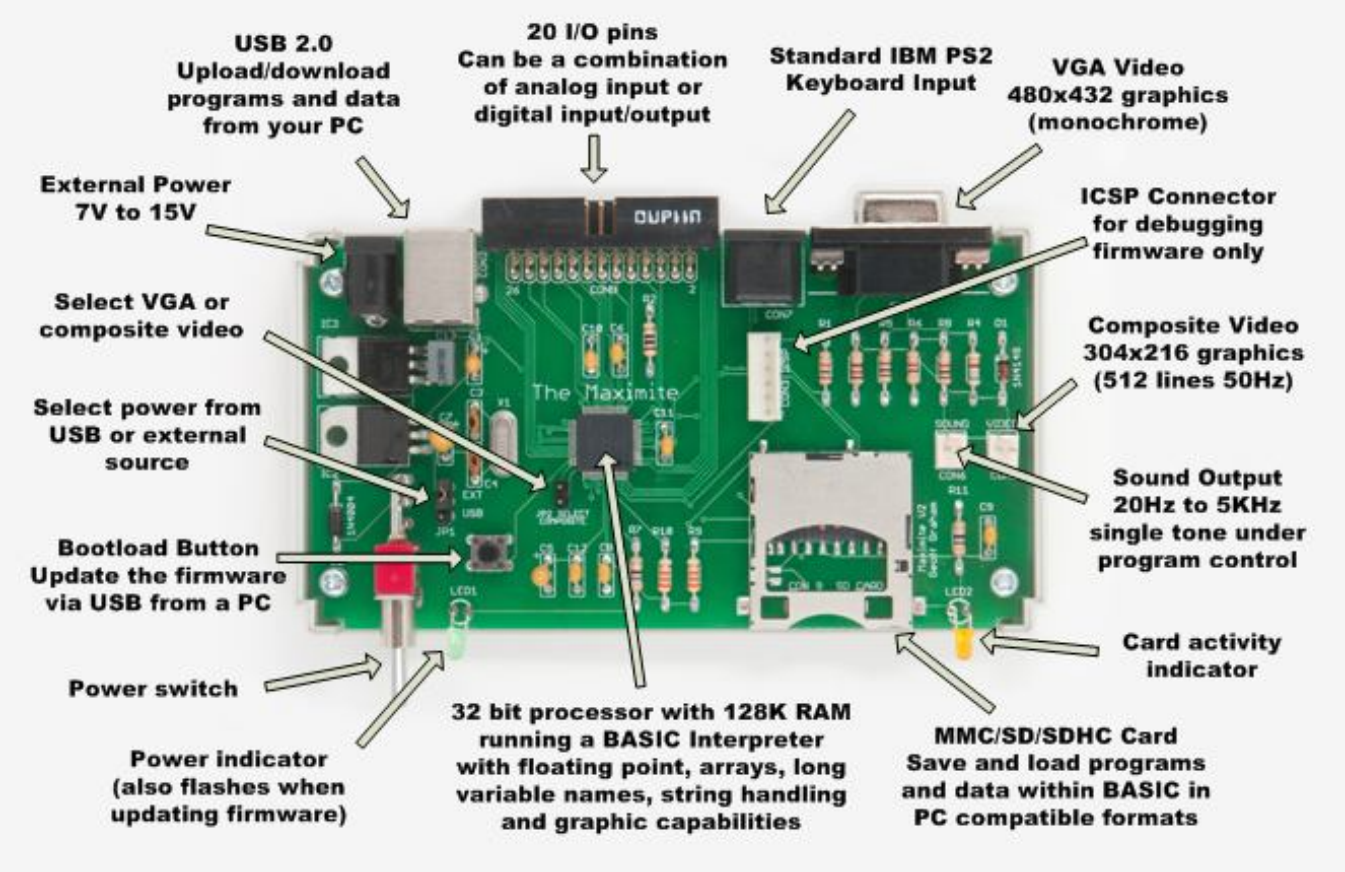
Maximite User Manual

For updates to this manual go to http://geoffg.net/maximite.html



Video Output

Placing a jumper on JP2 will select composite video output (eg, a TV set), removing it will select VGA video output (only one can be plugged in at a time).

VGA

Standard monochrome VGA (31.5KHz horizontal scanning with 60Hz vertical refresh).

480x432 pixel graphic screen. 80 characters per line and 36 lines per screen

Composite

Standard monochrome PAL (15.625KHz horizontal scanning with 50Hz vertical refresh non interlaced).

304x216 pixel graphic screen. 50 characters per line and 18 lines per screen

USB

Implements the CDC (Communication Device Class) protocol over USB 2.0. This is a serial interface to the BASIC interpreter so, by using a terminal emulator on the host, programs can be entered, edited and run. Using this interface you can upload programs by streaming the text with a suitable terminal editor.

The Windows driver is available from http://geoffg.net/maximite.html There is native support for the CDC protocol in Linux (the cdc-acm driver) and Apple OS/X.

Keyboard

Standard IBM compatible PS2 keyboard with mini-DIN connector or a USB/mini-DIN adapter.

Non ASCII keys (such as the function keys) are mapped to ASCII characters. Use a command like `PRINT HEX$(ASC(INKEY$))` to check the actual mapping.

SD/MMC Card Interface

Will accept MMC, SD or SDHC memory cards formatted as FAT16 or FAT32. File names must be in 8.3 format (long file names are not supported). Note that there is no advantage in using a fast SD card as the card is clocked at a fixed 20MHz, regardless of its speed rating.

Electrical Characteristics

Power Supply

Via External Power:	7V to 12V (14V if no significant current is drawn from the I/O pins). The centre pin of the external power connector is positive.
Via USB Connector:	4.5V to 5.5V (JP1 placed in the USB position)
Current Draw:	140mA typical (plus current draw from the I/O pins)

Digital Inputs

Logic Low:	0 to 0.65V
Logic High:	2.5V to 3.3V (I/O pins 1 to 10) 2.5V to 5.5V (I/O pins 11 to 20)
Input Impedance:	>1MΩ. All digital inputs are Schmitt Trigger buffered.
Frequency Response:	Up to 200KHz (pulse width 10nS or more) on the counting inputs (pins 11 to 14).

Analog Inputs (I/O pins 1 to 10)

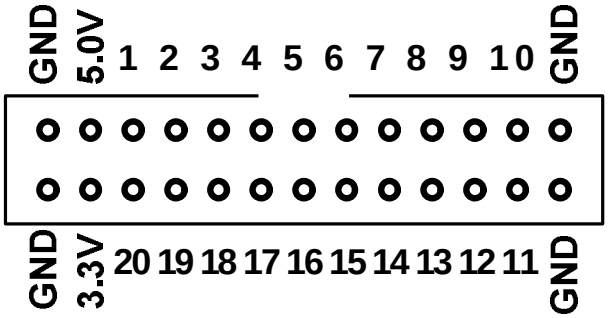
Voltage Range:	0 to 3.3V
Accuracy:	Typically better than ±1% although this can be considerably improved by using a correction factor in the BASIC program.
Input Impedance:	>1MΩ (for accurate readings the source impedance should be <10K)

Digital Outputs

Typical current draw or sink ability on any I/O pin:	10mA
Maximum current draw or sink on any I/O pin:	25mA
Maximum current draw or sink for all I/O pins combined:	150mA
Maximum open collector voltage (I/O pins 11 to 20):	5.5V

External Input/Output Connector

Rear panel connector with the pin numbers as used in MMBasic (external view looking at the back panel):



Loading New Firmware

The Maximite has the ability to reprogram itself with a new version of its firmware – also known as re-flashing or a boot loading. To do this, hold down the PROGRAM button on the PC board while you apply power and the front panel power LED will flash to indicate that the Maximite is in the reprogramming mode. Firmware upgrades can be downloaded from http://geoffg.net/maximite.html and will come with a program called "BootLoader.exe", run this program and follow the instructions included in the upgrade package to re program the Maximite with the new version.

Maximite BASIC V2.6

Functional Summary

Keyboard/Display

Input can come from either a keyboard or from a computer using a terminal emulator via the USB interface. 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.

Startup

On startup MMBasic looks for a file called "AUTORUN.BAS" in the root directory of the SD card and will automatically load and run it if found.

If it is not found MMBasic will print a prompt (">") and wait for input.

Command and Program Input

At the prompt 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. If the line is preceded with a number it will be saved in memory along with other lines in the program. The program can be listed with LIST and run using the RUN command.

You can interrupt MMBasic at any time by typing CTRL C on either the keyboard or USB interface and control will be returned to the prompt.

A program line held in memory can be changed by using the EDIT command or by entering a new line with the same number thereby overwriting it. A line can be deleted by entering its number without any commands. All program lines can be cleared from working memory with the NEW command.

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

SD Card Storage

A program can be saved to the SD card 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. Both data and programs are stored using standard text and can be read and edited in Windows, Apple Mac, Linux, etc.

You can list the programs stored on the SD card with FILES, delete them using KILL and rename them using NAME. 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"

Expressions

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

FNAME$ = "TEST": RUN FNAME$ + ".BAS"

Or, as an extreme (and not recommended) example:

NBR = 100 : GOTO NBR * 3 + 20

Structured Statements

MMBasic supports a number of modern structured statements. The IF… THEN command can span many lines with ELSEIF … THEN, ELSE and ENDIF statements as required and also spaced over many lines. The DO WHILE … LOOP command and its variants make it easy to build loops without using the GOTO statement.

Timing

You can get the current date/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 using TIMER as a command.

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.

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). The Maximite 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 C 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.

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 (similar to a GOSUB). The target line number 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 Maximite 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 on the SD card.

Sound

The SOUND command will generate a simple square wave between 20Hz and 5KHz lasting for a specified duration. This command can also be used to generate a Pulse Width Modulation (PWM) signal.

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). These include the DO WHILE … LOOP and structured IF .. THEN … ELSE … ENDIF statements.

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

Naming Conventions

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

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”.

Implementation Characteristics

Maximum length of a command line is 255 characters.

Maximum length of a variable name 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) than can be manipulated without loss of accuracy is ±16777100.

Maximum string length is 255 characters.

Maximum line number is 65000.

Maximum number of files simultaneously open is 10.

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

Predefined 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.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 15 = Directory not empty 15 = Hardware error accessing the storage media

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.
CHDIR dir$	Change the current working directory on the SD card to ‘dir$’ The special entry “..” represents the parent of the current directory and “.” represents the current directory.
CIRCLE (x, y) ,r [,c [,F]]	Draws a circle on the video output centred at x and y with a radius of r. If c is zero the pixels are turned off, if it is non zero or not specified the pixels are turned on (ie, the circle is drawn). The F option will cause the circle to be filled according to the c parameter. See page 4 for a definition of the graphics coordinates. Note that because the Maximite’s pixels are not exactly square the circle will be oval to some degree.
CLEAR	Will delete all variables and recover the memory used by them. See ERASE for deleting specific array variables.
CLOSE [#]nbr [,[#]nbr] …	Will close the file(s) or serial port(s) previously opened with the file number ‘nbr’. The # is optional. Also see the OPEN command.
CLOSE CONSOLE	Will close a serial port that had been previously opened as the console.
CLS	Clears the video display screen and places the cursor in the top left corner.
CONTINUE	Will resume running a program that has been stopped by an END statement, an error, or CTRL-C. The program will restart with the next statement following the previous stopping point.
COPYRIGHT	List all contributors to the Maximite firmware including MMBasic and summarise the copyright statement.
DATA constant[,constant]...	Stores numerical and string constants to be accessed by READ. String constants do not need to be quoted unless they contain significant spaces, the comma or a keyword (such as THEN, WHILE, etc). Numerical constants can also be expressions such as 5 * 60.
DATE$ = "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 start with the first line in the program. If startline- is used it will delete 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 simply loops; the only way out is by EXIT or GOTO.
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.
EDIT [line-number]	Edit the line 'line-number'. If no line number is used this command will edit the previous entry typed at the command prompt. On entering the edit mode the line will be displayed and the cursor placed at the beginning of the line. The editing commands are: Space — Step the cursor one character to the right Backspace — Step the cursor one character to the left D — Delete the character over the cursor nnD — Delete nn characters at the cursor and to the right X — Jump to the end of the line and enter insert mode I — Enter insert mode. All keystrokes will be inserted before the cursor. Use escape to exit insert mode. Use Enter (or Return) to finish editing (even in insert mode). When Enter is pressed the line is added to the program just as if it had been typed in at the command prompt. If the line number had been changed a new (edited) copy of the line will be added to memory, if it is unchanged the line will replace 'line-number'. The maximum line length that can be edited is 79 chars in VGA mode and 49 chars in composite mode.
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	Will end the running program and return to the command prompt.
ERASE variable [,variable]...	Deletes arrayed variables and frees up the memory. Use CLEAR to delete all variables including all arrayed variables.
ERROR [error_msg$]	Will force an error and terminate the program. This is normally used in debugging or to trap events that should not occur.

EXIT [FOR]	EXIT by itself exits from a DO...LOOP EXIT FOR exits from a FOR...NEXT loop.
FILES [search_pattern$]	List files in the current directory on the SD card. The optional 'search_pattern$' may contain question marks (?) to match any character. An asterisk (*) as the first character of the filename or extension will match any file or any extension. If omitted, all files will be listed.
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 line	Initiates a subroutine call to the line specified. The subroutine must end with RETURN.
GOTO line	Branches program execution to the specified line.
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. The 'statement' can be just a line number and in that case a GOTO is assumed. For Microsoft compatibility the 'THEN statement' construct can be also replaced with 'GOTO linenumber'. This type of IF statement is all on one line.

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	Will return 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$ from the SD card. 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 line does not start with a command.
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, non zero = on) 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 4 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 SD card 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.
LOCATE x, y	Positions the cursor to a location in pixels and the next PRINT command will place its output at this location. See page 4 for a definition of the graphics coordinates. Only affects the video output.
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%) Array and string memory used`
	Program memory is cleared by the NEW command. Variable, array and string memory spaces are cleared by many commands (eg, NEW, RUN, LOAD, etc) as well as the specific commands CLEAR and ERASE.

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$	Will rename a file or a directory on the SD card from ‘old$’ to ‘new$’
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 line[,line,line,...]	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of variable; if it is 1, the first line is called, if 2, the second line is called, etc.
OPEN fname$ FOR mode AS [#]fnbr	Will open a file on the SD card for reading or writing. ‘fname’ is the filename (8 chars max) with an optional extension (3 chars max) separated by a dot (.). ‘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. ‘comspec’ is the serial port specification and has the form: “COMn: baud, buf, FC” with an optional “,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. 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 the remote control of the Maximite via a serial interface. For a full description with examples see Appendix A.

OPTION BASE 0 or OPTION BASE 1	Sets 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	Sets 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 or writing from the SD card, it does not affect the handling of syntax and other program errors.
OPTION PROMPT string$	Sets the command prompt to the contents of 'string$' (which can also be an expression which is evaluated when the prompt is printed). For example: OPTION PROMPT "Ok " or OPTION PROMPT TIME$ + ": " 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 SD card. 10 OPTION PROMPT "My prompt: " 20 NEW
PAUSE nbr	Will halt execution of the running program for 'nbr' milliseconds. The maximum value of 'nbr' is 4294967295 (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. 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 non zero). See page 4 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 #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.
PRESET (x, y)	Turn off a pixel on the video screen. This statement is included for Microsoft compatibility. New programs should use the PIXEL(x,y) = statement.
PSET (x, y)	Turn on a pixel on the video screen. This statement is included for Microsoft compatibility. New programs should use the PIXEL(x,y) = statement.
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 begins at that line. Or, if a file name (file$) is supplied, the current program will be erased and that program will be loaded from the SD card and executed. This enables one program to load and run another. Example: RUN "TEST.BAS" If an extension is not specified ".BAS" will be added to the file name.
SAVE file$	Saves the program in the current working directory on the SD card as 'file$'. Example: SAVE "TEST.BAS" If an extension is not specified ".BAS" will be added to the file name.
SAVEBMP file$	Saves the current VGA or composite screen as a BMP file in the current working directory on the SD card. 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 and 5V tolerant on pins 11 to 20) 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 to 20) In this mode the function PIN() will also return the output value. 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 ,line	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 'line'. 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, line	This will setup a periodic interrupt (or "tick"). The time between interrupts is 'period' milliseconds and 'line' is the line number 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 frequency, duration or SOUND frequency, duration, dutycycle	Generate a single tone of 'frequency' (between 20Hz and 1MHz) for 'duration' milliseconds. The sound is played in the background and does not stop program execution. If 'duration' is zero, any active SOUND statement is turned off. If no SOUND statement is running, a 'duration' of zero has no effect. 'dutycycle' 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 100KHz the duty cycle will be accurate to 0.1%.
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 "0:0:0" 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.
WEND	WEND concludes a WHILE-WEND loop; see WHILE.
WHILE expression	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 … LOOP construct.
WRITE [#nbr,] expression [,expression] …	Outputs the value of each ‘expression’ separated by commas (,). If the ‘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 on the SD card previously opened for OUTPUT or APPEND as ‘#nbr’. See the OPEN command. WRITE is useful for writing data that will later be read by the INPUT command or by programs that can read CSV (comma separated variables) format files (such as Microsoft’s Excel).

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.
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 on the SD card 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 time use the command DATE$ =.
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$	Reads the status of the keyboard, and returns a single character if available. If a character is not available, this function will immediately return with a null string ("").
INPUT$(nbr, [#]fnbr)	Will return a string composed of ‘nbr’ characters read from a file previously opened for INPUT with the file number ‘fnbr’. This function will read all characters including carriage return and new line without translation. The # is optional. Also see the OPEN command.
INSTR([start-position,] string-searched$, 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.
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 200KHz). 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 is a special case which will always return the state of the bootload 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.
PIXEL(x,y)	Returns the value of a pixel on the VGA or composite screen. Zero is off, 1 is on. See page 4 for a definition of the graphics coordinates. See the statement PIXEL(x,y) = for setting the value of a pixel.
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.
SPC(number)	Returns a string of blank spaces 'number' bytes long. This function is similar to the SPACE$() function and is only included for Microsoft compatibility.
SPI(rx, tx, clk[, data[, speed]])	Sends and receives a byte using the SPI protocol with the Maximite 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 the Maximite (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 500KHz, M is 50KHz and L is 5KHz. Default is H. See Appendix C for a full description.
SQR(number)	Returns the square root of the argument 'number'.

STR$(number)	Returns a string in the decimal (base 10) representation of the argument 'number'.
STRING$(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$)	Returns the numerical value of the ‘string$’.

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 on the SD card. 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, one for transmit and one for receive. The default size is 256 bytes.
- 'int' is the line number 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, 10000, 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.

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 a high voltage. 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).

For RS232 you need a chip like the MAX232 to invert the signal and generate the ±12V signal levels required by the standard. Because the MAX232 is a 5V chip you will also need to set the outputs to open collector and use pull up resistors to 5V.

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.

Note that the RENUMBER command does not renumber the interrupt line number in ‘comspec’.

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 the Maximite. 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^2C) 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^2C bus is complicated and if you do not need to connect your Maximite to 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^2C 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^2C data line (SDA) and pin 13 the clock (SCL). Both of these pins should have external pullup resistors installed (a typical value is 4.7K connected to +5V). Refer to the diagram on page 2 of this manual for the location of pins 12 and 13 on the external I/O connector.

Be aware that when running the I^2C bus at above 150KHz you may need to vary the value of the pullup resistors and that the cabling between the devices becomes very 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^2C 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 100kHz 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_line]	Enables the I^2C module in master mode. 'speed' is a value between 10 and 400 (for bus speeds 10kHz to 400kHz). '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_line' is optional. It specifies the line number 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^2C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN. It will also send a stop if the bus is still held.

I2CSEND addr, option, sendlen, senddata [,sendata ....]	Send data to the I^2C slave device. 'addr' is the slave i2c address. 'option' is a number between 0 and 3 1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 2 = treat the address as a 10 bit address 3 = combine 1 and 2 (hold the bus and use 10 bit addresses). 'sendlen' is the number of bytes to send. 'senddata' is the data to be sent - this can be specified in various ways (all values sent will be between 0 and 255): • The data can be supplied in the command as individual bytes. Example: I2CSEND &H6F, 1, 3, &H23, &H43, &H25 • The data can be in a one dimensional array (the subscript does not have to be zero and will be honoured also bounds checking is performed). Example: I2CSEND &H6F, 1, 3, ARRAY(0) • The data can be a string variable (not a constant). Example: I2CSEND &H6F, 1, 3, STRING$ The automatic variable MM.I2C will hold the result of the transaction.
I2CRCV addr, bus_hold, rcvlen, rcvbuf [,sendlen, senddata [,sendata ....]]	Receive data from the I^2C slave device with the optional ability to send some data first. 'addr' is the slave i2c address (note that 10 bit addressing is not supported). 'option' is a number between 0 and 3 1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command) 2 = treat the address as a 10 bit address 3 = combine 1 and 2 (hold the bus and use 10 bit addresses). 'rcvlen' is the number of bytes to receive. 'rcvbuf' is the variable to receive the data - this is a one dimensional array or if rcvlen is 1 then this may be a normal variable. The array subscript does not have to be zero and will be honoured, also bounds checking is performed. Optionally you can specify data to be sent first using 'sendlen' and 'senddata'. These parameters are used the same as in the I2CSEND command (ie, senddata can be a constant, an array or a string variable). Examples: I2CRCV &h6f, 1, 1, avar I2CRCV &h6f, 1, 5, anarray(0) I2CRCV &h6f, 1, 4, anarray(2), 3, &h12, &h34, &h89 I2CRCV &h6f, 1, 3, anarray(0), 4, anotherarray(0) The automatic variable MM.I2C will hold the result of the transaction.

I²C Slave Mode Commands

I2CSEN addr, mask, option, send_int_line, rcv_int_line	Enables the I²C 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 the Maximite 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_line' is the line number of a send interrupt routine to be invoked when the module has detected that the master is expecting data 'rcv_int_line is the line number of a receive interrupt routine to be invoked when the module has received data from the master.
I2CSDIS	Disables the slave I²C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN.
I2CSSEND sendlen, senddata [,sendata ....]	Send the data to the I²C 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 I²C 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
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, the Maximite 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 the Maximite (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 500KHz, M is 50KHz and L is 5KHz. 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 not "take control" of the I/O pins like the serial and I^2C protocols and the PIN command will continue to operate as normal on them. Also, because the I/O pins can be changed between function calls it is possible to communicate with many different SPI slaves on different I/O pins.

Example

The following example will send the command &H80 and receive two bytes from the slave SPI device. Because the 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
```