MMBasic for DTX modules

ICeemite family DTX2-4105C / 06C / 07C

Language Additions

to

MMBasic for Maximite (original document)

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

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

The ICeemite revision contains a large number of additions and improvements, some of which in the form of extra commands and functions, but the majority of them invisible to the end user.

This manual describes **only** the additions and changes introduced to the original MMBasic language for the support of the **ICeemite** family modules. It does not replace the original MMBasic manual but only completes it with the additional information.

For further details about the hardware of the ICeemite modules, please go to http://dimitech.com.
For general details and running MMBasic on other platforms please refer to http://mmbasic.com.

ICeemite embedded modules are designed and manufactured by Dimitech Pty Ltd

4/12/14 Document Revision 8

Contents

What's New

16 December 2014: Document Revision 9

Based on 4.5/R9.2. Minor release with fixed stability bugs and the handling of Ctrl-C

4 December 2014: Document Revision 8

Based on 4.5/R9.1. Minor release with fixed bugs in the TIME$ and DATE$ functions

15 November 2014: Document Revision 7

Based on 4.5/R9. Significantly improved program execution speed. Fixed known bugs from the previous release. Added description of:

- OPTION LOCK command for protection of sources from listing and editing
- OPTION UNLOCK command for temporary source access protection
- OPTION THREADS for enabling control over the number of serviceable threads

30 May 2014: Document Revision 6

Based on 4.5/R8. Major MMBasic update with a number of added features in MMBasic (see the link on the title page).

Changed the format of this document to describe only the changes and additions to the 'standard' MMBasic

3 May 2014: Document Revision 5

Based on 4.4/R7.2. Minor corrections. Added description of:

- SPEED function (see SPEED command and Appendix K)

27 April 2014: Document Revision 4

Based on 4.4/R7.1. Added description of:

- MSD command

18 October 2013: Document Revision 3

Based on 4.4/R7. Added description of:

- RESET command
- Changed some details in "Implementation Characteristics"
- Fixed obsolete LOCATE format to correspond to Microsoft's original format <row,column>

1 September 2013: Document Revision 2

Based on 4.4/R6. Added description of:

- SPEED command for low-power modes (including the new Appendix K)
- SIM$ function to calculate similarity of two strings

25 August 2013: Document Revision 1

Based on 4.4/R5. The original text by Geoff Graham edited to match the hardware description and the current unique to the ICeemite commands, functions and other features added:

- Hardware description of the ICeemite modules and relation between ports and MMBasic pins
- RTC-seeded RANDOMIZE on power on
- Inherent DRIVE command (can be omitted when switching drives)

- DIR supported along with FILES (plus the extended options)
- Hidden files and file attributes
- INIT command for formatting drives
- HELP command
- Changed editor keymap and ability to create files
- THREAD and PRIORITY commands (new Appendix J)

Functional Summary

This chapter describes the most basic aspects of MMBasic for ICeemite modules. It contains important information and you should familiarise yourself with it before taking on more serious work.

Command and Program Input

At the prompt (the symbol **]**) 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 and down arrow keys 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.

Program execution can be interrupted at any time by pressing **CTRL-C** and control will be returned to the prompt.

Keyboard and 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. Either can be attached or removed at any time.

Standard CRT or LCD displays with VGA or composite input can be used. Terminal connections via serial link (USB or UART) can one display text output while any graphics will be ignored.

Program and Data Storage

In DOS the drive letters are as supported by Windows. In MMBasic two "drives" are available for storing and loading programs and data:

- Drive "A:" is a virtual drive using part of the internal flash memory and has a size of about 180k with the current firmware revision
- Drive "B:" is the SD card on board (if installed). 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 (8 characters for file name and 3 characters for extension), 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 after power on and reset events is B: and this can be changed with the DRIVE command.

During start up 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 the prompt ("]") and wait for input. If the module has been restarted because the watchdog timer has timed out and forced a restart, the file "RESTART.BAS" will be run instead (see the WATCHDOG command for details).

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

Drive A: does not support hierarchic organisation with folders. All files in A: only exist in the root folder. However a file in A: can be made "hidden" if its name starts with the '.' character. For example the file MYPROG.BAS in A: is a normally visible file, while the file .MYPROG.BAS will not be seen in the FILES command. Making files in A: hidden can be used to protect important system files from being accidentally deleted or corrupted in other way by the user.

When connected to a PC via USB, the B: drive (if a SD card is installed) appears as accessible data storage drive on the computer. It provides a very convenient way for data transfer from and to the module. This feature is always available and does not require any user interaction. In Windows-based PCs a special driver should be installed to allow proper operation simultaneously as USB terminal and storage device. For latest drivers and other information visit http://dimitech.com.

Date and Time

Current date and time can be set or read using the DATE$ and TIME$ functions.

Program execution can be stopped 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.

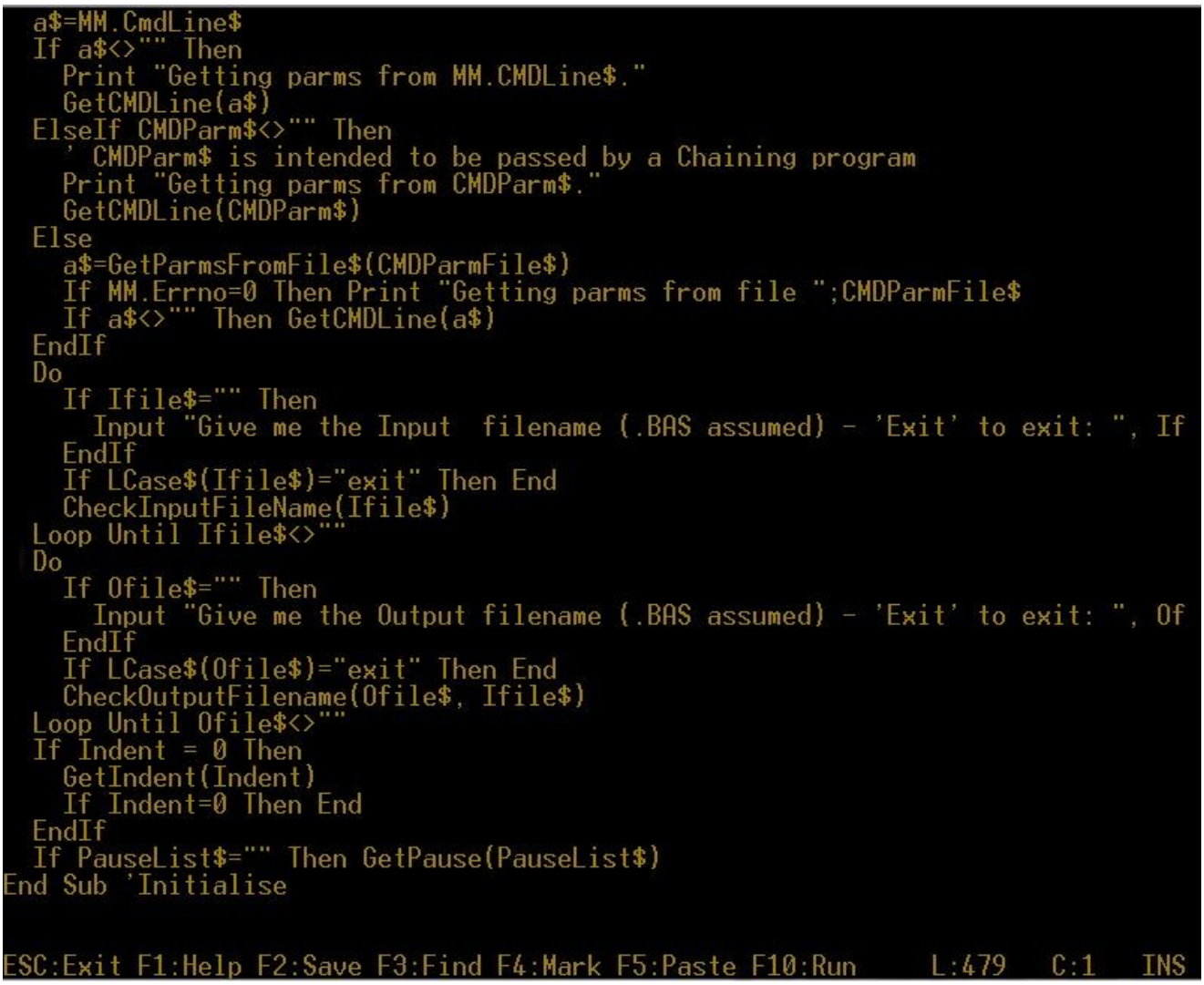
Using SETTICK you can setup a "tick" which will generate a regular interrupt with a period from one millisecond to over a month. See Interrupts below.

Graphics

The ICeemite modules use monochrome graphics and don't support colours as per the concept of the Colour Maximite as it offers limited benefits for embedded systems while consuming significant amount of RAM. Such functionality can be easily achieved and exceeded in an embedded system wherever needed, by using a proper TFT LCD module with SPI or other suitable control bus.

Full Screen Editor

An important productivity feature of MMBasic is the full screen editor which allows programs to be conveniently developed directly on the device. It will work using an attached video screen (VGA or composite) and over USB with a VT100 compatible terminal emulator (using "Tera Term" is highly recommended).



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

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

You can also run the editor with a file name (eg, EDIT "file.ext") and the editor will edit that file while leaving program memory untouched. If the file doesn't exist, an option to create one with this name, will be offered. This is handy for examining or changing files on the disk without disturbing your program.

If you are used to an editor like Notepad you will find that the operation of the full screen editor is familiar. The arrow keys will move your cursor around in the text, home and end will take you to the beginning or end of the line. Page up and page down will do what their titles suggest. The delete

key will delete the character at the cursor and backspace will delete the character before the cursor. The insert key will toggle between insert and overtype modes.

About the only unusual key combination is that two home key presses will take you to the start of the program and two end key presses will take you to the end.

The editor also allows HELP command to be invoked by pressing the F1 key. You will be asked for a keyword for help. The help system implements a simple smart search which will allow in many cases the needed information to be found on the basis of partial or mistyped keyword.

At the bottom of the screen the status line will list the various function keys used by the editor and their action. In more details these are:

ESC	This will cause the editor to abandon all changes and return to the command prompt with the program memory unchanged. If you have changed the text you will be asked if this is really what you want to do.
F1: HELP	Will invoke the built-in HELP system.
F2: SAVE	This will save the program to program memory and return to the command prompt. If you are editing a disk file it will save that file to the disk.
F3: FIND	This will prompt for the text that you want to search for. When you press enter the cursor will be placed at the start of the first entry found.
F10: RUN	This will save the program to program memory and immediately run it.
SHIFT-F3	Once you have used the search function you can repeatedly search for the same text by pressing SHIFT-F3.
F4: MARK	This is described in detail below.
F5: PASTE	This will insert (at the current cursor position) the text that had been previously cut or copied (see below).
CTRL-F	This will insert (at the current cursor position) a file residing on the disk. Note that while this key is not listed on the status line it is always available.

You can also use control keys instead of the functions keys listed above. These control keystrokes are:

LEFT	Ctrl-S	RIGHT	Ctrl-D	UP	Ctrl-E	DOWN	Ctrl-X
HOME	Ctrl-U	END	Ctrl-K	PageUp	Ctrl-P	PageDn	Ctrl-L
DEL	Ctrl-]	INSERT	Ctrl-N				
F1	Ctrl-?	F2	Ctrl-Q	F3	Ctrl-R	F4	Ctrl-T
F5	Ctrl-Y	F10	Ctrl-W	ShiftF3	Ctrl-G		

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

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

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

The editor is a very productive method of writing a program. Using the OPTION Fnn command you can program a function key to generate the command "EDIT" when pressed. So, with one key press you can jump into the editor where you can make a change. Then by pressing the F10 key you can save and run the program. If your program stops with an error you can press the edit function key and be back in the editor with the cursor positioned at the line that caused the error. This edit/run/edit cycle is very fast.

If the full screen editor is used over USB, screen size of 80 characters by 36 lines should be configured for best results. See Appendix G for details.

Note that a terminal emulator like "Tera Term" can lose its position in the text with multiple fast keystrokes (like the up and down arrows). If this happens you can press the HOME key twice which will force the editor to jump to the start of the program and redraw the display.

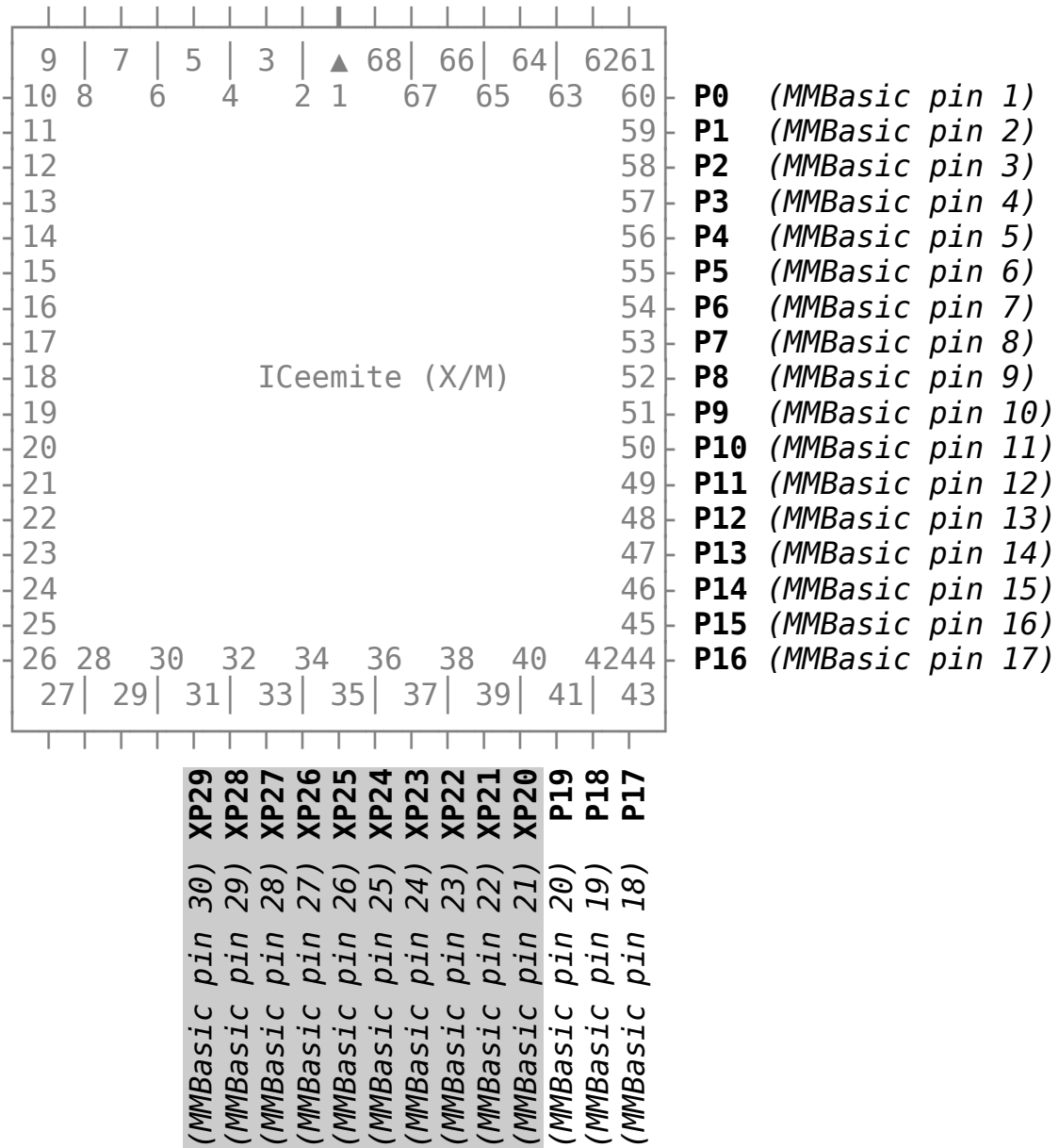
Input/Output

Pin References in Hardware and MMBasic

IMPORTANT! Read carefully the following paragraph

In the hardware I/O pins are referred to as "PortX" (where 'X' is a decimal number). They always start from 0. Hence ICeemite modules have 20 I/O pins numbered from 0 through 19: ***P0...P19***.
MMBasic however, following the standard BASIC language conventions, enumerates everything with indexes starting from **1**.

From MMBasic's point of view the same hardware I/O ports P0 through P19 are named as '***pin 1***' through '***pin 20***'. This creates a possible confusion in the end user, and when reading the text you should carefully note whether the text says "P.." or "pin ..." as those two are not the same thing.
This relation between hardware I/O and MMBasic "pins" is demonstrated below:



NOTE: Greyed pins are only available in ICeemite X

IMPORTANT!

Everywhere in this document *pin **x*** (x=1…20) in MMBasic refers to ***Px-1*** (x=0…19) in the hardware I/O ports and ***Px*** (x=0…19) refers to *pin **x+1*** (x=1…20) in MMBasic.

External I/O Pins

An external I/O pin can be configured to a specific function using the SETPIN command, set its output using the PIN command and read the current input value using the PIN() function. Digital I/O pins use positive logic where the number 0 to represent a low voltage and any non-zero number (in programs typically 1) represents a high voltage. An analogue input will report the measured voltage as a floating point number.

The DTX2-4105C and DTX2-4107C have 20 I/O pins numbered in MMBasic from 1 to 20 (referring to P0…P19 in the hardware schematic).

DTX2-4106C has additional 10 digital I/O pins numbered in MMBasic from 21 to 30 (referring to XP20…XP29 in the hardware schematic).

Pins 1 to 10 can be used for analogue input and digital input/output with a maximum input voltage of 3.3V. Pins 11 to 20 are digital only but support input voltages up to 5V and can be set to open collector. Expansion pins 21 to 30 are also digital only with voltages up to 3.3V.

Note that pin 0 actually exists in MMBasic, but not as an available to the user I/O port but an output only and hard-wired to the on-board LED indicator. It can be used in programs as PIN(0)=<value> to turn the LED on or off.

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

Implementation Details

Maximum length of a command line is 255 characters.
Maximum length of a variable name or a label is 32 characters.
Maximum number of dimensions to an array is 8.
Maximum number of arguments to commands that accept a variable number of arguments is 50.
Maximum number of nested FOR…NEXT loops is 33.
Maximum number of nested DO…LOOP commands is 33.
Maximum number of nested GOSUBs is 100.
Maximum number of nested multi-line IF…ELSE…ENDIF commands is 33.
Maximum number of user defined subroutines and functions (combined): 250
Maximum number of simultaneously running threads: 11
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 65500.
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:) varies with the firmware release and currently is around 180KB.
Maximum size of a loadable video font is 64 pixels high x 255 pixels wide and 107 characters.
Maximum number of library files that can be loaded simultaneously is 20.
Maximum number of background pulses launched by the PULSE command is 5.

Compatibility

MMBasic implements a large subset of Microsoft's GW-BASIC. There are numerous small differences due to physical and practical considerations but most MMBasic commands and functions are essentially the same. An online manual for GW-BASIC is available at http://www.antonis.de/qbebooks/gwbasman/index.html and this provides a more detailed description of the commands and functions.
MMBasic also implements a number of modern programming structures documented in the ANSI Standard for Full BASIC (X3.113-1987) or ISO/IEC 10279:1991. These include SUB/END SUB, the DO WHILE … LOOP and structured IF .. THEN … ELSE … ENDIF statements.

License

Additional and Changed Commands

Note: Square brackets indicate that the parameter or characters are optional.

DRIVE drivespec$	Change the default drive used for file operations that do not specify a drive to that specified in 'drivespec$'. This can be the string "A:" or "B:". ICeemite modules allow execution of an implied DRIVE command. Hence for example DRIVE A: can be written just as A: See also the predefined read-only variable MM.DRIVE$.
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 program file. It can also be used to view and edit text data files. If EDIT is used on its own it will edit the program memory. If 'filename' is supplied the file will be edited leaving the program memory untouched. If the supplied filename does not exist, the editor will offer the creation of a new file. 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. 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.
END	Ends the current thread. If END is executed from within a thread, it will terminate the execution of the thread but will not affect the execution of any other running threads or the main thread. Generally END is an equivalent of PRIORITY 0, with one exception. If END is executed from within the main thread, it will end the whole program with all of its threads and will return back to the command prompt, while executing PRIORITY 0 in the main thread will terminate it leaving other threads running, i.e. the program will not stop. See the PRIORITY command.
FILES [fspec$] [A] or DIR [fspecs$] [A]	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 An optional parameter '-A' displays the file attributes when executed on drive B: and ALL files (including the hidden ones) on drive A:

HELP [keyword$]	Displays help information about the provided keyword. From the command line keyword$ without enclosing quotes will work for almost all system words, however it is recommended that keyword$ is always enclosed by quotes or at least the opening one is present. Examples: HELP HELP CIRCLE HELP 'CIRCLE or HELP 'CIRCLE' HELP "CIRCLE or HELP "CIRCLE" The command is also able to find in many cases the most suitable match for a partial or mistyped keyword. As in the examples above HELP CICLE will lead to displaying help information about the CIRCLE command as a closest match to the parameter 'CICLE'.
MSD ON | OFF	Enable (parameter ON) or disable (parameter OFF) file access to the internal microSD card in drive B: by a host PC via USB connection By default this function is activated. Note that the access to B: may not work on MacOS powered machines. MSD OFF is recommended in such cases.
PRIORITY pr	Assigns thread priority. The parameter 'pr' is an integer number greater than 0. By default every thread starts with priority 1 which means the switching is 'chopped' at one command level. This can be changed by assigning the thread higher priority (ie. executing more commands at once). Priority can be changed at any time, but a thread can change its own priority only. PRIORITY 0 effectively ends the thread and it will never receive the execution again, similar to an END command. There are some differences between PRIORITY 0 and END though. PRIORITY also can be invoked as a function to return the current thread's priority. See the END command and PRIORITY function.
RANDOMIZE nbr	Seed the random number generator with 'nbr'. On power up the random number generator is seeded from the real time clock, which provides a good base for the generation of random numbers each time and makes the use of this command not necessary. RANDOMIZE can be used to re-seed the random generator with the value 'nbr' when needed. One good way to do seed for random values is to use the TIMER function. For example 100 RANDOMIZE TIMER
RESET	Resets the system from within the software
SPEED mode [,wumask]	Changes the clock frequency of PIC32 or enables sleep mode. See Appendix K for full details about how to use the SPEED command.
THREAD target	Initiates the execution of a new parallel thread. The parameter 'target' can be a line number or label. THREAD acts somewhat similar to a GOTO command, however it runs the code **in parallel** with caller. The command itself does not take time and will continue immediately the execution further. The new thread will also start immediately with default priority 1. See Appendix J for more details about threads.

OPTION THREADS th	Sets the maximum number of services threads in the system. The default value is 1. Setting a higher value enabled the multitasking capabilities on the expense of some reduced execution speed.
OPTION LOCK key	Permanently locks with a key the execution of some commands: LIST, DIR (in drive A: only), EDIT, and SAVE. Thus programs and data stored in A: can be completely protected from being accessed and viewed by the user. Key can be any valid integer number, except -1. **<u>WARNING!</u> Since the key is programmed into the microcontroller's flash memory, locking is <u>PERMANENT</u>. Once set, the key cannot be changed or removed (only temporary unlocking is possible with the OPTION UNLOCK command).** **Be careful with this command!**
OPTION UNLOCK key	Temporarily unlocks a protected system. Unlocking lasts only until the next reset cycle. To prevent the use of brute force key recovery algorithms, an attempt with wrong key enforces a mandatory 5 seconds delay.

Additional and Changed Functions

<u>Note:</u> Square brackets indicate that the parameter or characters are optional.

PRIORITY	Returns the priority of the current thread.
SIM$(string1$, string2$)	Returns the calculated similarity of two strings as a number between 0 and 1.
SPEED	Returns the current SPEED setting. See the SPEED command and Appendix K

Appendix Z1
Multitasking with MMBasic on ICeemite

MMBasic for the ICeemite family implements simple multitasking abilities to run several threads simultaneously. This feature is achieved with the use of only two commands – THREAD and PRIORITY. MMBasic does not take any precautions to prevent the value of a variable used in one thread to be changed or corrupted by another thread. The user should organise this process in the most convenient fashion suiting the particular application.

The format of the THREAD command is:

THREAD <statement>

A thread is a piece of executable code ending with an END command. On power up and RUN command a default thread is created (also called "main thread") and the execution starts with it only. The user may create more threads at any time when that is needed.

The following example demonstrates running three separate threads (two additional plus the main one):

```
Thread MyThread1
Thread MyThread2

For count0 = 1 To 40
  Print "Executing MAIN thread"
Next
End

MyThread1:
  For count1 = 1 to 20
    Print "Executing THREAD1"
  Next
End

MyThread2:
  For count2 = 1 to 10
    Print "Executing THREAD1"
  Next
End
```

This program will run these extra two threads in parallel to the main one, producing a result in the form:

```
Executing MAIN thread
Executing THREAD1
Executing THREAD2
```

Executing MAIN thread
Executing THREAD1
Executing THREAD2
Executing MAIN thread
Executing THREAD1
Executing THREAD2
… and so on…

There are some limitations in the way threads can be used. Execution blocking commands (such as INPUT) will stop the execution of all threads until the command produces result. Another limitation is with the use of local variables in shared threads – that will produce a misleading error message about re-defined variable.

Another limitation is the lack of system tools to end or change the priority of another thread. Functions like those must be organised at user program level by passing values to dedicated global variables.

A thread can be started and stopped at any time at will. The command THREAD will initiate its execution and will return the control back to the caller immediately. The command END will stop the thread and remove it from the execution stack.

A small, but important detail about using the END command is when it is used from within a thread, it will stop the thread only, but if it is used from within the main program body, all threads including the main one will be stopped altogether.

Normally each new thread gets exactly one command system time. That can be changed with the command PRIORITY. It will assign a desired number of commands uninterrupted execution on every turn of this thread. The format is:

```
PRIORITY <level>
```

A thread can only change its own priority. In the example above if you change MyThread2 like this:

```
MyThread2:
  Priority 3
  For count2 = 1 to 10
    Print "Executing THREAD1"
  Next
End
```

It will produce the following output:

Executing MAIN thread
Executing THREAD1
Executing THREAD2
Executing THREAD2
Executing THREAD2
Executing MAIN thread
Executing THREAD1
Executing THREAD2

Executing THREAD2
Executing THREAD2
Executing MAIN thread
Executing THREAD1
Executing THREAD2
Executing THREAD2
Executing THREAD2
… and so on…

A thread can change its own priority at any time. The default priority, assigned by the THREAD command is always 1.

Self-assigning priority 0 will effectively end the thread just like the execution of END command. If the main thread sets its own priority at 0, that will not stop the execution of any other threads unlike the END command.

The current priority of a thread can be read by using the same PRIORITY command in the form variable = PRIORITY.

Appendix Z2
Low-power modes

The SPEED command allows the user to utilise much better the ICeemite especially in battery-powered systems by reducing the amount of consumed current depending on the current needs. In some cases the system can work on a much slower clock rate while consuming only a fraction of the power normally needed for full-speed operation. In other cases the system needs to be completely stopped and drawing the possible minimum of power (commonly known as "sleep" mode), and waiting for a specific event to resume its work.

MMBasic for ICeemite provides the SPEED command to address the low-power needs of an embedded system.

The command has format:

SPEED mode [,wumask]

where "mode" is a number and "wumask" only has meaning in mode=0 and defines the wake-up mask as a bitmask where every bit specifies event on certain hardware. If "wumask" is missing, its default value is assumed to be 0.

The following table describes the wake up bits in "wumask" and only applies to speed mode 0.
Note that the PS2 keyboard can wake up the ICeemite unconditionally.

Bit in WUMASK	Wake-up event
0	activated on raising or falling edge of P0
1	activated on raising or falling edge of P1
2	activated on raising or falling edge of P9
3	activated on falling edge of P10
4	activated on falling edge of P11
5	activated on falling edge of P12
6	activated on falling edge of P13
7	activated on raising or falling edge of LDFW

Examples:

```
SPEED 0,0          ' sleep and configure wake up up the PS2 keyboard only
SPEED 0,1          ' sleep and configure wake up from P0 (and keyboard)
SPEED 0,&H84       ' sleep and configure wake up from P9 or LDFW (and keyboard)
SPEED 0,&HFF       ' sleep and configure wake up from all the possible sources
```

Once awake from sleep, ICeemite will restore its work in the last active mode.

SPEED changes dynamically the clock frequency according to several pre-defined modes. The command can be executed at any moment in a program and the changes take place immediately without restarting the whole device. Thus the user can monitor the needs in real time and change the power consumption accordingly for best utilisation of the power source.

In some modes not all peripheral interfaces are available due to their own hardware requirements. Some of the lowest power modes can only work as a deeply embedded system – without support of a console or video. If the user need to have some interaction in a console with a system working in one of those low-power modes, an external signalling mechanism (such as a I/O port, monitored by the program and switching to a higher SPEED mode upon a need) should be used.

Note that the video composite mode most probably will not work properly in all modes due to the much stricter limitations for that a TV imposes on the video signal, compared to a VGA-compatible display.

The following table shows the pre-defined SPEED (parameters showing ICeemite) command modes.

Mode	Clock Frequency	Current Consumption	USB Console	Video
0	stopped	0.32mA	no	no
1	1MHz	2.5mA	no	no
2	8MHz	16.0mA	no	no
3	24MHz	47.0mA	yes	no
4	32MHz	60.4mA	yes	yes
5	**40MHz (default)**	**73.2mA**	**yes**	**yes**
6	48MHz	86.1mA	yes	yes
7	60MHz	107.0mA	yes	yes
8	72MHz	121.1mA	yes	yes
9	80MHz	133.1mA (absolute maximum 142mA)	yes	yes