

## **QSBASIC Manual - Version 1.001**

Program lines consist of a line number from 1 to 65535, then a space, then one or more statements separated by a colon (":"). Most statements can be given without a line number and will be executed immediately if so.

If a line is entered with the same line number as that of an existing line in the current program, the existing line is deleted and the new line is stored.

A line number by itself may be entered. Any existing line in the current program with the same line number will be deleted. No new line will be stored.

When QSBASIC is first started after a Propeller reset, if an SD card is provided and it has a file "autoexec.bas" in its root directory, then the line 'LOAD "autoexec.bas" : RUN' is automatically executed.

The ESC key is a "break key". If it is entered, the running program will stop after the current statement finishes executing. PAUSE is handled specially so the "break key" can interrupt their execution.

The basic SD card wiring is shown below. "Pull-up" refers to a 20K pull up resistor from the pin indicated to +3.3V. For the Demo Board version, the pins are as indicated below. For the Proto Board version, P0 = I/O pin 8, P1 = I/O pin 9, P2 = I/O pin 10, and P3 = I/O pin 11. For the Hydra version, P0 = I/O pin 16, P1 = I/O pin 17, P2 = I/O pin 18, and P3 = I/O pin 19.

### **SD CARD Socket Pin-out:**

PIN	SD CARD	Propeller
1 (NC)		
2	(PIN-9) DAT2	Pull-up
3	(PIN-1) CS	Pull-up P15
4	(PIN-2) DI	Pull-up P14
5	(PIN-3) GND	GND
6	(PIN-4) +3.3	VCC
7	(PIN-5) CLK	Pull-up P13
8	(PIN-6) GND	GND
9	(PIN-7) DO	Pull-up P12
10	(PIN-8) DAT1	Pull-up
11 (CD SW)		

## **Expressions**

Expressions consist of variables, constants, and "pseudo-variables" that function like variables, but may have complex actions like FILE or EEPROM[5]. Constants may be decimal, hexadecimal (prefixed with "\$"), or binary (prefixed with "%"). All expressions use 32-bit integer values.

<var>

There are 26 variables designated by the letters A through Z. Upper and lower case letters are equivalent.

INA [ <expr> {.. <expr>} ]

This has the value of the specified input pin or pins. If two pin values are given, the first is the most significant bit number and the second is the least significant bit number of the value. The pin or pins specified are changed to input mode (the default).

BYTE [ <expr> ]

This has the value of the main memory byte at the address provided.

WORD [ <expr> ]

This has the value of the main memory word at the address provided. The least significant bit of the address is ignored.

LONG [ <expr> ]

This has the value of the main memory long word at the address provided. The least significant two bits of the address are ignored.

EEPROM [ <expr> {, <expr>} ]

This has the value of the byte in the EEPROM attached to the boot I2C bus (on pins 28-29) at the address specified. If two expressions are provided, the first gives the pin number of the SCL line of the EEPROM while the second expression is the address. The address may be any value from zero to \$7FFFF and the upper 3 bits are used to form the device select code.

FILE

This has the value of the next byte in the currently open SD card file or -1 at the end of the file. The file must be open for reading.

MEM

This has the value of the amount of space available for program storage.

CNT

The current system clock value.

PHSA

The cog counter phase register A value.

PHSB

The cog counter phase register B value.

FRQA

The cog counter frequency register A value.

FRQB

The cog counter frequency register B value.

CTRA

The cog counter control register A value.

CTRB

The cog counter control register B value.

KEYCODE

The value of the next keyboard key value or zero if the keyboard buffer is empty. This syntax is for QSBASIC only.

KEYCODE [ <expr> ]

This returns the value of the next keyboard key value or zero if the keyboard buffer is empty and a new character is not received before the timeout expires. The timeout in milliseconds is given as the expression in brackets. This syntax is used by DongleBASIC, BoeBotBASIC, and uOLED96PropBASIC.

RND <expr>

The value is a pseudo-random number in the range zero to one less than that of the expression given.

- <expr>

! <expr>

"-" is negate. "!" is bit-wise logical not.

<9> SHL <9>

<9> SHR <9>

<9> ROL <9>

<9> ROR <9>

<9> SAR <9>

<9> REV <9>

"SHL" is logical shift left. "SHR" is logical shift right. "ROL" is rotate left. "ROR" is rotate right. "SAR" is arithmetic shift right. "REV" is bit reverse. In all cases, the value to be shifted is the left operand and the shift count is the right operand. "REV" reverses the order of the specified number of least significant bits with the most significant bits set to zero.

<8> & <8>

"&" is bit-wise logical and.

<7> | <7>

"|" is bit-wise logical or

<6> \* <6>

<6> / <6>

<6> // <6>

"\*" is a 32 bit unsigned multiplication. "/" is the quotient of a 32 bit unsigned division. "//" is the remainder of a 32 bit unsigned division.

<5> + <5>

<5> - <5>

"+" is a 32 bit addition. "-" is a 32 bit subtraction

<4> = <4>

<4> < <4>

<4> > <4>

<4> <= <4>

<4> >= <4>

<4> <> <4>

"=" is equal to. "<" is less than. ">" is greater than.

"<=" is less than or equal to. ">=" is greater or equal to.

"<>" is not equal to.

NOT <3>

"NOT" is logical not.

<2> AND <2>

"AND" is logical and.

<1> OR <1>

"OR" is logical or.

Note that the numbers in the brackets (<n>) are used to indicate the operator precedence.

I2C [ <expr> , <expr> , <expr> ]  
I2C [ <expr> , <expr> , <expr> , <expr> ]

This returns 1 to 4 bytes (in LSB..MSB order) from an I2C device. The 1st <expr> is the pin number of the clock (SCL) signal. The data signal is connected to the next numerically higher pin. For the boot EEPROM I2C bus, this would be 28 (and 29). The 2nd <expr> is the device select code. The read/write bit is ignored and will be set appropriately for the operation involved. If four <expr> are provided, the 3rd <expr> is the 8-bit address. The last <expr> in the brackets is the number of bytes to be read.

## **Statements**

Note that multiple statements may be given on a line separated by a colon (":"). There are some restrictions on what can be combined on a line. These are described in the individual statements' descriptions.

{LET} <var> = <expr>

Set the variable <var> to the value of the expression <expr>.

INPUT { "<prompt>"; } <var> { ,<var> }

If given, the <prompt> is displayed and an input line may be entered. For each variable given, an expression is evaluated from the input line. The expressions may be separated by commas (",") or, if unambiguous, by spaces. These expressions may contain variable names, operators, or "pseudo-variables". If more expressions are given than variables, the excess are ignored. An error is treated as if it occurred on the line where the INPUT statement is given.

PRINT {"<string>" | <expr> } {, | ;}

A series of expressions or string constants are given, separated by commas (",") or semicolons (";"). If a comma is used, enough spaces are inserted to display the next item at the next display column divisible by 8. If a semicolon is used, the next item begins at the next column. If the statement ends with a comma or semicolon, an end of line is not inserted. A PRINT statement by itself causes an end of line to be displayed

GOTO <expr>

Go to the label whose value is equal to the expression

GOSUB <expr>

Call (as a subroutine) the label whose value is equal to the expression. Note that a GOSUB must be the only or last statement on a line.

RETURN

Return from a GOSUB call

REM <comment>

The rest of the line in the program is considered part of the comment and is otherwise ignored.

NEW

Erase the current program and clear all the variables.

LIST {<expr> {,<expr>}}

List the current program. If no expressions are given, the whole program is listed. If one expression is given, that line is listed. If two expressions are given, all lines between those two values are listed.

RUN

Clear all the variables (to zero) and start executing the current program from the first line.

OPEN "<file>", {R | W | A }

Open the specified file on the SD card in the mode requested (R - read, W - write, A - append). If a file is already open, it is closed first. Only one

file may be open at a time.

READ <var> {,<var>}

Read a line from the currently opened SD card file and set the variables specified to the expressions supplied on that line. The expressions may be separated by commas or, if unambiguous, may be separated by spaces. These expressions may be any expression including operators, variables, pseudo-variables (like CNT). Effectively, this is as if "<var> = <expr>" were executed for each variable given and each expression in the SD card file.

WRITE {"<string>" | <expr> } {, | ;}

This works just like the PRINT statement except that the characters produced are written to the currently opened SD card file. An end of line is written as a carriage return / line feed pair (ASCII CR/LF ... 13, 10).

CLOSE

Close the currently opened SD card file, if any.

DELETE "<file>"

Delete the specified SD card file. Any opened SD card file will be closed.

RENAME "<file>", "<file>"

Rename the specified SD card file. Any opened SD card file will be closed. This is not currently implemented and will produce an error message.

FILES

List all files on the SD card (at the root level). Neither subdirectories nor the files within them are included in this listing. Any opened SD card file will be closed.

SAVE

Save the current program to an otherwise unused area in the boot EEPROM. Note that downloading a program to the EEPROM using the Propeller Tool or an equivalent downloading program will erase any saved program.

SAVE [ <expr> {, <expr>} ]

This saves the current program in the EEPROM attached to the boot I2C bus (on pins 28-29) at the address specified. If two expressions are provided, the first gives the pin number of the SCL line of the EEPROM while the second expression is the address. The address may be any value from

zero to \$7FFFF and the upper 3 bits are used to form the device select code. The address is adjusted to 2 bytes below the next 64 byte boundary and the total program length is stored in those two bytes followed by the program itself. If the address is adjusted upwards, only the last two bytes of that 64 byte block are changed.

SAVE "<file>"

Save the current program to the specified file on a SD card. Any existing file by that name will be overwritten with the program which will be saved in text file format, as if they were displayed with the LIST statement.

LOAD

Erase the current program and load in a program previously saved with the SAVE statement.

LOAD [ <expr> {, <expr>} ]

Erase the current program and load in a program previously saved with the SAVE [ <expr> {, <expr>} ] statement.

LOAD "<file>"

Erase the current program and load in a program from an SD card file. This program must be in text format, just as if it were to be typed in. All lines must be numbered (with a line number) except lines that are completely blank.

FOR <var> = <expr> TO <expr> {STEP <expr>}

This sets up a standard Basic FOR/NEXT loop. The variable is set to the value of the first expression and tested against the limit given by the value of the second expression (which is evaluated only once). The optional step size may be positive or negative. If negative, the limit test is appropriately changed. The FOR statement must be the last statement on a multi-statement line and improperly nested FOR/NEXT statement pairs may cause incorrect execution without an error message. Default STEP value is +1.

NEXT <var>

This terminates a standard Basic FOR/NEXT loop. The STEP value is added to the variable and the result is tested against the limit value. If still within the limit, program execution continues with the statement after the matching FOR statement. If not, execution continues with the next statement.

OUTA [ <expr> {.. <expr>} ] = <expr>



This sets the specified output pin or pins to the expression to the right of the assignment. If one pin value is given, that is the pin to be changed. If two pin values are given, the first is the most significant bit number and the second is the least significant bit number of the value. The pin or pins specified are changed to output mode.

The LEDs on the QuickStart board are connected to I/O pins 16 through 23. You can turn them on and off by using the OUTA statement on those bits. For example, to turn on the right-most LED, you'd use "OUTA[16] = 1".

PAUSE <expr> {, <expr>}

The program is paused for the number of milliseconds given by the first (or only) value given. If two values are given, the first is in milliseconds while the second is in microseconds and they're added together for the total pause time. The minimum pause time is 50us. If the pause time is more than 10ms, the pause statement is interrupted after 10ms and reexecuted with a 10ms shorter pause time. This is to allow for the interruption of the program using a "break key". The PAUSE statement must be the first or only statement on a line.

BYTE [ <expr> ] = <expr>

This sets the value of the main memory byte at the address provided to the expression on the right side of the assignment.

WORD [ <expr> ] = <expr>

This sets the value of the main memory word at the address provided to the expression on the right side of the assignment. The least significant bit of the address is ignored.

LONG [ <expr> ] = <expr>

This sets the value of the main memory long word at the address provided to the expression on the right side of the assignment. The least significant two bits of the address are ignored.

PHSA = <expr>

Set the cog counter phase register A to the expression.

PHSB = <expr>

Set the cog counter phase register B to the expression.

FRQA = <expr>

Set the cog counter frequency register A to the expression.

FRQB = <expr>

Set the cog counter frequency register B to the expression.

CTRA = <expr>

Set the cog counter control register A to the expression.

CTRB = <expr>

Set the cog counter control register B to the expression.

DISPLAY <expr> {, <expr> }

Send the specified byte values to the display driver. The specific control codes, their parameters, and their meaning depend on the display driver. See the display driver documentation for descriptions.

STOP

Stop execution of the program.

END

Stop execution of the program (works like STOP).

EEPROM [ <expr> {, <expr>} ] = <expr>

This sets the value of the byte in the EEPROM attached to the boot I2C bus (on pins 28-29) at the address specified. If two expressions are provided, the first gives the pin number of the SCL line of the EEPROM while the second expression is the address. The address may be any value from zero to \$7FFFF and the upper 3 bits are used to form the device select code.

FILE = <expr>

This sets the value of the next byte in the currently open SD card file. The file must be open for writing or appending.

SPIN [ <expr> {, <expr>} ]

This causes a Spin program to be loaded into the Propeller's main memory from a 32K EEPROM "page". If only one expression is provided, it is the starting address in a 512K byte address space made up of one or more

EEPROMs attached to the I2C bus on Propeller pins 28 (SCL) and 29 (SDA). The boot EEPROM is the first 32K of this address space. The lower order 15 bits of the address are ignored so the loading process always begins on a 32K byte boundary. If two expressions are provided, the first gives the pin number of the SCL line of the EEPROM while the second expression gives the starting address. The address may be any value from zero to \$7FFFF and the upper 3 bits are used to form the device select code. Once the Spin program has been successfully loaded, it begins execution. The loaded Spin program completely replaces the running QSBASIC.

SPIN "<file>"

This causes a Spin program to be loaded into the Propeller's main memory from a specified file on an attached SD card. This file should be a copy of the binary form of a Spin program as saved from the Propeller Tool. Once the Spin program has been successfully loaded, it begins execution. The loaded Spin program completely replaces the running QSBASIC.

DUMP <expr> , <expr>

This displays a portion of the Propeller's main memory. The first expression gives the starting address and the second expression gives the number of bytes to be displayed. The information is formatted 8 bytes per line with both hexadecimal and ASCII displayed.

DUMP [ <expr> {, <expr> } ] , <expr>

This displays a portion of the EEPROM. The last expression gives the number of bytes to be displayed. The first portion describes a starting address in EEPROM. See the SPIN statement for a description of the values.

COPY [ <expr> {, <expr> } ] , [ <expr> {, <expr>} ]

This copies a Spin program from the first 32K byte EEPROM "page" specified to the second. As with the SPIN statement, if only one expression is supplied, it provides the starting EEPROM address. If two are supplied, the first is the pin number of the SCL line while the second is the EEPROM address. The amount of data copied is taken from the beginning of the Spin program binary file.

COPY "<file>" , [ <expr> {, <expr>} ]

This copies a Spin program from an SD card file to a 32K byte EEPROM "page".

COPY [ <expr> {, <expr>} ] , "<file>"

This copies a Spin program from a 32K byte EEPROM "page" to an SD card file.

## BUTTONS

This has the value of the touch buttons on the QuickStart board as an 8 bit binary value with button 0 as the least significant bit and button 7 as the most significant bit.

```
I2C [ <expr> , <expr> , <expr> ] = <expr>  
I2C [ <expr> , <expr> , <expr> , <expr> ] = <expr>
```

This writes 1 to 4 bytes (in LSB..MSB order) to an I2C device. The 1st <expr> is the pin number of the clock (SCL) signal. The data signal is connected to the next numerically higher pin. For the boot EEPROM I2C bus, this would be 28 (and 29). The 2nd <expr> is the device select code. The read/write bit is ignored and will be set appropriately for the operation involved. If four <expr> are provided, the 3rd <expr> is the 8-bit address. The last <expr> in the brackets is the number of bytes to be written. The data to be written is taken from the <expr> on the right of the equal sign.

-----  
Try the following test program:

```
100 OUTA[23..16] = BUTTONS  
110 GOTO 100
```