

## **FemtoBasic Manual - Version 3.006**

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 FemtoBasic 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 and IRBRECVR are handled specially so the "break key" can interrupt their execution.

Pre-compiled binaries are provided for both a VGA display and a TV display and for either the Propeller Demo Board, Propeller Proto Board, or a Hydra. The Hydra version cannot use an SD card with a VGA display because the SD card interface uses the same I/O pins as the VGA display. The names of the binary files indicate which display and board they are intended for.

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 P3
4	(PIN-2) DI	Pull-up P2
5	(PIN-3) GND	GND
6	(PIN-4) +3.3	VCC
7	(PIN-5) CLK	Pull-up P1
8	(PIN-6) GND	GND
9	(PIN-7) DO	Pull-up P0
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.

## 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.

### **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.

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 FemtoBasic.

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 FemtoBasic.

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.

### **BOE-BOT Extensions**



The BOE-BOT version of FemtoBasic is similar to the regular version except that it uses a full duplex serial port for its keyboard input and display output and several "pseudo-variables" and statements have been added to control servos, a PING distance sensor, IR distance sensors, and an HM55B compass connected via a PCA9554 I2C I/O Expander. A Propeller Proto Board or equivalent is assumed.

The left servo is connected to I/O pin 0, the right servo to pin 1, and the PING bracket servo to pin 2. An IR detector is connected to pin 3 and an IR LED is connected to pin 4. The PING control signal is connected to pin 5. The "console" receive line is pin 6 and the transmit line is pin 7. This "console" is implemented using a configured xBee transceiver. A PCA9554 I2C I/O Expander is connected to the boot EEPROM bus using I/O pins 28 (SCL) and 29 (SDA).

The HM55B Ena pin is connected to PCA9554 I/O pin 0. The Clk pin is connected to pin1. DI is connected to pin 2 and DO to pin 3.

Two different binary versions are provided. One uses the programming serial port for the "console" (BoeBotBasicUS.binary) and the other uses pins 6 and 7 for wireless operation via an xBee transceiver (BoeBotBasicXB.binary).

### **Expressions**

PING

This value is the distance in mm of the last PING reading (one-way). It will be zero if a new reading has been initiated, but a value isn't ready yet.

IRZONE [ <expr> , <expr> , <expr> ]

The first expression is the center frequency (in Hz). The second expression is the number of zones. The third expression is the width of a zone (in Hz). The IR emitter frequency is swept from the last zone to the center frequency with about 200 cycles of each frequency emitted per zone. This value is the number of the first zone where a response is detected (#zones-1 to 0) or -1 to indicate that no response was detected.

EXPAND [ <expr> ]

This is the value of the PCA9554 I2C I/O Expander's register whose address is supplied.

COMPASS

This value is the compass heading in brads (0 to 359 degrees is the same as 0-255).

### **Statements**

SRVLEFT {[ <expr> ]} = <expr>  
SRVRIGHT {[ <expr> ]} = <expr>  
SRVPING {[ <expr> ]} = <expr>

Send a pulse stream to the specified servo with a width (in us) given by the expression on the right side of the assignment. If a square bracketed expression is provided, this is the number of pulses to send (at 20ms intervals). If no pulse count is provided, the pulse train continues indefinitely. If either the pulse count is zero or the pulse width is zero, the pulse train will stop. The pulse width must lie between 500us and 2500us.

PING

Initiates a new PING cycle. The one-way path length will be zero until the new reading is complete.

COMPASS <var> , <var>

Reads the raw x and y values of the HM55B compass and assign them to the first and second variables specified respectively.

EXPAND [ <expr> ] = <expr>

The PCA9554 I2C I/O Expander's register whose address is supplied is set to the value on the right side of the assignment.

### **IR Buddy Extensions**

The IR Buddy version of FemtoBasic is identical to the regular version except that statements have been added to control one or more IR Buddy devices. These may be connected to any otherwise available I/O pin.

#### **Statements**

IRBSEND <expr>

The expression is the I/O pin number to be used. This resets the IR Buddy.

IRBSEND <expr> , <expr> {, <expr>}

The first expression is the I/O pin to be used. The remaining expressions are byte values to be sent to the IR Buddy (at 9600 Baud).

IRBRCV <expr> , <expr> , <var> {, <var>}

The first expression is the I/O pin to be used. The second expression is an initial timeout (in ms) to use. Subsequent timeouts are 10ms. The number of bytes specified are received, one in each variable. If a timeout occurs, that variable and all subsequent ones are set to -1.

### **uOLED-96-Prop Extensions**

UOLED SETUP

Initialize the uOLED-96-Prop. This must be done before any other operations are done.

UOLED START

Power up the screen electronics and display any data previously written to graphics RAM.

UOLED STOP

Power down the screen electronics without disturbing any data in graphics RAM.

UOLED LEVEL <expr>

Set the master contrast setting (range 0-15).

UOLED COLOR <R> , <G> , <B>

Set the individual color contrast values (range 0-255).

UOLED DIM <X Up Lt> , <Y Up Lt> , <X Lo Rt> , <Y Lo Rt>

Dim a designated screen window given the coordinates of the left upper corner and the

right lower corner.

UOLED PIXEL <X> , <Y> , <R> , <G> , <B>

Writes 2 bytes of color data to the pixel at the coordinate specified. The color information range is 0-255.

UOLED SCROLL SETUP <X> , <Y> , <Address> , <# Lines> , <Interval>

X is the number of columns of horizontal offset. Y is the number of lines of vertical offset. Address is the starting line address. # Lines is the number of lines to be scrolled horizontally. Interval is the time interval between scroll steps. 0 = 6 frames, 1 = 10 frames, 2 = 100 frames, and 3 = 200 frames.

UOLED SCROLL START

Activate the scrolling function as set up previously

UOLED SCROLL STOP

Deactivate the scrolling function

UOLED LINE <X Up Lt> , <Y Up Lt> , <X Lo Rt> , <Y Lo Rt> , <R> , <G> , <B>

Display a line from the specified left upper corner to the specified right lower corner in the color specified.

UOLED RECT <X Up Lt> , <Y Up Lt> , <X Lo Rt> , <Y Lo Rt> , <R> , <G> , <B> {, <R> , <G> , <B>}

Display a rectangle from the specified left upper corner to the specified right lower corner of the display. The first set of color values is used for the outline color. If the second set of color values is given, it's used for the fill color. If not, the outline color is used for the fill color as well.

UOLED COPY <X Up Lt> , <Y Up Lt> , <X Lo Rt> , <Y Lo Rt> , <X Up Lt Dest> , <Y Up Lt Dest>

Copy one area of the display screen to another. The left upper corner and the right lower corner of the source area is supplied followed by the left upper corner of the destination area.

UOLED TEXT <X> , <Y> , <R> , <G> , <B> , " ... "

UOLED TEXT <X> , <Y> , <R> , <G> , <B> , <expr>

Display text using the current font starting at the coordinates provided. These are in terms of character positions, not pixels (range X: 0-11/15, Y: 0-7). Wraparound occurs at the right and bottom of the display. The first form displays the contents of the string while the second form displays the decimally formatted value of the expression given with a leading minus sign if negative. With the 5x7 font, there are 16 characters per line. With the 8x8 font, there are 12 characters per line.

UOLED TEXT <R> , <G> , <B>

Set the default background color for text. It's set to black during the initialization of FemtoBasic.

UOLED TEXT <Font>

Set the current font. 0 - 5x7 font. 1 - 8x8 font (default).

UOLED ERASE

Erase the screen (to black).

UOLED RESET

Resets the display. You must do a UOLED SETUP afterwards.

UOLED CIRCLE <X> , <Y> , <Rad> , <R> , <G> , <B>

Display a circle whose center is at X,Y and whose radius is Rad using the color specified. If Rad is negative, the circle is filled with the color specified and the radius is the absolute value of Rad.

UOLED CIRCLE <X> , <Y> , <Rad> , <Arc> , <R> , <G> , <B>

Display a one eighth circle arc whose center is at X,Y and whose radius is Rad using the color specified. If Rad is negative, the one eighth circle pie slice is filled with the color specified and the radius is the absolute value of Rad. Arc indicates which one eighth circle is to be displayed. 1 - 0 to 45 degrees, 2 - 45 to 90 degrees, 3 - 90 to 135 degrees, 4 - 135 to 180 degrees.

### **HC-OSD Extensions**

The Hitt Consulting's Overlay Screen Display version uses the PS/2 keyboard for input and the overlay screen driver for output. None of the commands that use SD card files are present.

### **Expressions**

SERIAL [ <timeout> ]

The parameter is a timeout in milliseconds. This returns the character received from the 19.2Kbps serial interface or a -1 if the timeout occurs.

SERCHK [ <break> ]

The parameter is a character. This searches the entire buffered serial input stream for the specified character and returns true (-1) if the character is present and false (0) otherwise. The serial buffer is not changed.

TIME [ <expr> ]

If the value is between 0 and 6, this returns the binary value of the most recently read time unit (0 - Seconds, 1 - Minutes, 2 - Hours, 3 - Day of Week, 4 - Day, 5 - Month, 6 - Year). If the value is 7, this returns the control register value read from the DS1307. For values from 8 to 63, this returns the value stored in the DS1307's RAM at that address.

GPS [ <expr> ]

If the value is negative, this returns the next character from the GPS serial buffer with the absolute value of the expression used as a timeout in milliseconds. It returns a -1 if the timeout is exceeded. If the value is positive, this returns a character from the saved GPS phrase whose index is the value provided. It returns a -1 if the value is out of range or if there's no saved GPS phrase.

### **Statements**

TIME

This reads the current time from the DS1307 into an internal buffer used by TIME [ <expr> ].

TIME [ <address> ] = <expr>

This writes the expression on the right side of the "=" into the control register or RAM whose address is given. Addresses less than 7 or greater than 63 are not allowed.

TIME <Sec> , <Min> , <Hrs> , <DOW> , <Day> , <Mth> , <Yr>

This writes the time / date indicated to the appropriate locations in the DS1307 clock. The values are given in binary and are translated to BCD. The BCD values are also stored in the internal buffer used by TIME [ <expr> ]. The BCD values are written to the DS1307 in a single operation.

GPS

This starts up a background routine (in a cog) that discards any buffered GPS phrase, then begins discarding any buffered serial input up to the first "\$" character which begins the next GPS phrase. Characters are then stored in the internal GPS phrase buffer (used by GPS [ <expr> ]) until a "" is seen. The next two characters must be hexadecimal values which are used as a checksum for the phrase. If the checksum is invalid, the phrase is discarded and the routine begins searching for the next phrase. If this background routine is already active, it is stopped and any saved information is discarded before starting it again. Once this background routine successfully finds a complete GPS phrase, it stops itself.

### **Extensions for TriBlade Prop Blade #1 use**

The Application key is used as a "break" key rather than ESC.

The SD card support is removed.

### **Expressions**

FLASH [ <flash address> , <byte count> , <buffer address> ]

This transmits 1 to 4 bytes to a Winbond Flash Memory and optionally receives or transmits a block of data. <flash address> is 1 to 4 bytes in size with the most significant byte containing an instruction code for the Flash Memory (see the datasheet for details) and usually the 3 least significant bytes containing a 24 bit address in the Flash Memory. If <byte count> is negative, its absolute value is a count of bytes to be sent to the Flash Memory from <buffer address>. If <byte count> is positive, it's a count of bytes to be received from the Flash Memory to <buffer address>. Normally <buffer address> is a hub memory address. If it is zero, then the address of the (long) return value for FLASH[ ] is used.

PAGE

This returns the address of a 256 byte buffer within FemtoBasic. This is intended for use in BYTE[ ], WORD[ ], and LONG[ ] expressions and in FLASH[ ] expressions.

### **Statements**

TERM <recv pin> , <xmit pin> , <mode> , <Baud>

This starts a simple terminal emulator using FullDuplexSerial with a 256 byte buffer. The parameters are the corresponding values used by the FullDuplexSerial start call. All keyboard characters are sent to the serial pins specified except as noted below based on Turbo Pascal's default definitions. Received characters are displayed on the VGA display. It's intended that the VT100 interpreter from propComm will be used, but currently only the control codes used by Chip's hi-res text VGA driver are interpreted. Carriage return and line feed have their usual meaning.

\$C0: Left Arrow	- ^S
\$C1: Right Arrow	- ^D
\$C2: Up Arrow	- ^E
\$C3: Down Arrow	- ^X
\$C4: Home	- ^Q ^E
\$C5: End	- ^Q ^X
\$C6: PgUp	- ^R
\$C7: PgDn	- ^C
\$C8: Backspace	- ^H
\$C9: Del	- ^G
\$CA: Ins	- ^V
\$CB: Escape	- ^[
\$CC: Application Key	- Exit to FemtoBasic
\$D0..\$DB:	
Function keys become ESC A to ESC L	