Instruction set of the AttoBASIC interpreter
Version: 2.34

Instruction set of the AttoBASIC interpreter
Version: 2.34

How to read this reference

Values and Variables

AttoBASIC is a "byte-wide BASIC", meaning that it processes data in 8-bit "chunks". Therefore, the data has a value in the range of "0" to "255".

Like all other programming languages, AttoBASIC processes data from an input to an output. Input may be from the console, a port as digital or analog input (or pin), an SPI, TWI or other peripheral device. Output may be to the console, an output port (or pin) as digital or analog, an SPI, TWI or other peripheral device. In any case, the data must be passed to and from each keyword to perform some function on it.

AttoBASIC supports the variables "A" through "D", "A" through "M" through "A" to "Z", depending on the chosen MCU, as temporary (volatile) storage locations.

Most AttoBASIC keywords can accept either a direct value, the contents of a variable or the result of another keyword. Other keywords default to a value if one is not provided on the command or program line.

In all cases where AttoBASIC expects to see a value with a keyword, the value is represented by a letter enclosed in square brackets, as in "[x]", "[y]", etc. In other words, if the syntax for a keyword shows "**XOR [x] [y]**" then one would substitute the 1st value for "[x]" the 2nd value for "[y]" for. Example: **PRINT XOR $55 $AA**, where $55 and $AA are direct values, or **PRINT XOR A B**, where A and B are both variables.

Keywords

AttoBASIC uses "keywords" as commands to tell it to do something. Each keyword executes a series of low-level processor instructions, a "macro instruction".

In flavors of BASIC running on high-end computers, the entire keyword is needed. In other words, the **PRINT** command must be spelled as "**PRINT**" or an error will be generated.

Being that AttoBASIC runs on a microcontroller with a limited amount of RAM, program storage is also limited, so as a general rule, only the 1st three (3) characters of a keyword are needed. In four (4) letter keywords, it is acceptable to use all four but not necessary. Thus, the keyword **PRINT** may be spelled as "**PRI**" or "**PRIN**". To make it clear, the following AttoBASIC command instruction reference will show the 1st three (3) characters of the keyword in upper case and for clarity, the rest of the keyword in lower case.

Delimiters

AttoBASIC uses "delimiters" to separate certain commands and values from each other. The delimiters are the space (" ") and comma (",") characters. Example: **POKE $02,$FF,$55** is the same as **POKE $02 $FF $55**.

Instruction set of the AttoBASIC interpreter
Version: 2.34

Keyword Reference

Control

CONTROL-C ..... The control-c keyboard combination halts program operation.

CONTROL-S ..... The control-s keyboard combination stops the program until another key is pressed (not another ^S).

CONTROL-U ..... In editing or interactive mode, the control-u keyboard combination deletes the line.

NEW ..... New Program EX: **NEW**

LISt ..... List program Ex: **LIST**

PRInt [x] ..... Print value to screen Ex: **PRINT A**

PRX[1] [x] ..... Print hex Ex: **PRX 100** results in the output: **$64**

PRB[1] [x] ..... Print binary Ex: **PRB INB** prints PINB in binary

Note that **PRI**nt, **PRX** *and* **PRB** *will take more than one command line parameter. However, the values printed will be separated by a space and printed in reverse order. Ex:* **PRX 1 2 3 4** *will print* **$04 $03 $02 $01**.

KEY ..... Get key from terminal Ex: **A := KEY** ; or KEY (return) to pause.

EMIt ..... Emit value as ASCII character to terminal Ex: **EMIT $20** sends a space character.

RND ..... Creates an 8-bit random number. Ex: PRI RND [or] A:= RND.

RUN [x] ..... Run program. If the EEPROM File System is enabled, then [x] is a file number, which is first loaded into program memory (the current program is overwritten) then run. Ex: **RUN** will run the program currently in memory while **RUN 2** will first load the program stored in file number 2 into memory then run it.

IF-THEn ..... Control structure. Ex1: **IF A=31 THEN GOTO 100** Ex2: **IF A=31 THEN PRINT "Match at " ; PRINT A**

FOR-TO-NEXt .... Looping structure. FOR-NEXT loops can be nested four (4) levels deep (or two (2) levels deep on MCU's with less that 1024 bytes of RAM). The **TO** keyword is optional. Ex: **FOR I=1 TO 3 ; PRINT I ; NEXT**

GOSub-RETurn ... Program flow control. GOSUB's can be nested four (4) levels deep (or two (2) levels deep on MCU's with less that 1024 bytes of RAM).

GOTo [x] ..... Flow Ex: **GOTO 100**

DELay [x] ..... Delay "X" * 10mS. Ex: **DELAY 20** will cause a delay of 200mS

FREe[1] ..... Print the number of bytes used by a program and the remaining bytes of program space available.

END ..... Stop execution of program. This command is not required at end of program. EX: **END**

WTF ..... Calculates with exact precision, the answer to life, the universe and everything. Ex1: **PRB WTF** prints the binary representation of the answer to life, the universe and everything. Ex2: **A:= WTF** assigns the answer to life, the universe and everything to the variable A.

<backspace> .... Destructive backspace during line editing

BLDr[1] ..... Invoke the boot-loader [*Note: this command uses the AVR's BOOTSZ1:0 fuse bits to determine the location and existence*

[1] This keyword (or keyword set) is a specific language extension to AttoBASIC and is not a standard keyword of BASIC.

of a boot-loader before jumping to it. An error message is displayed if no boot-loader exists.]

RST[1] ..... Causes a system reset using the watchdog "System Reset" mode. *Note that devices using USB Serial I/O will disconnect from the host.*

CFG[1] [x] [y] .... Reads or writes AttoBASIC's configuration to non-volatile memory (EEPROM), where x = bit position (0 to 7 or 8) and y = value (0 or 1). Without x and y returns the byte value of the configuration register. Ex1: **PRX CFG 0** prints the value of the configuration register in hexadecimal. With x only, places the value of bit position "x" on the data stack. Ex2: **PRINT CFG 1** prints the value of bit position "1" in the configuration register. With x and y, writes the value of "y" into bit position "x" in the configuration byte. Ex3: **CFG 1 1** stores the value of "1" in bit position "1" of the configuration register. However, if x = 8 then y is assumed to be a byte-wide value to be written to the configuration register. Ex4: **CFG 8 $AA** directly stores the value of "$AA" to the configuration register. As currently defined;

POS	Description
0	1 = enable self-start, 0 = disable self-start

NOTE: On the ATtiny85 and ATmega88[PA], only byte-wide reading and writing to the configuration memory is supported. Ex1: ***PRX CFG 0*** *prints the value of the configuration register in hexadecimal. Ex2:* ***CFG $AA*** *directly stores the value of "$AA" to the configuration register.*

$[1] (Dollar Sign) Convert the one (1) or two (2) following characters from ASCII HEX. Ex: **A:= $31** will convert the value to 49.

'[1] (Apostrophe) . Convert the following one (1) to eight (8) characters from ASCII binary. Ex1: **A:= '10101010** will convert the binary value to 170 and assign the value to the variable A. Ex2: **PRX '101** will convert the binary value to hexadecimal $05 and print it.

;[1] (Semicolon) . Used to separate commands on a program line. One can use this keyword to add multiple commands on a program line. This helps to overcome the 255 program line limitation. Ex: **10 TWI ; TWS ; TWW $5C $55 ; TWP** initializes the TWI interface, asserts a START condition, addresses the slave at address $5C, writes "$55" to it and asserts a STOP condition. [*Note: the semicolon is only valid when embedded in a program line and not in immediate mode*]

#[1] (pound sign) . The line interpreter ignores (does not store) all characters between the "#" and a CR or LF. This allows one to add comments to programs created with an external text editor and uploaded via the terminal interface. *[Note: this command not the same as the* **REM** *command.]*

REM ..... Leave a remark. EX: **10 REM This is Program 1** allows comments to be stored in the program. *[Notes: 1) this command is only available when the EFS support is enabled and is intended to be used to identify a program when the* ***CAT*** *command is issued. 2) Use this command with caution as it eats program memory. 3) this command is not the same as the "#" above.]*

Instruction set of the AttoBASIC interpreter
Version: 2.34

Relational

:= [1] ..... Set equal to, which is used to assign a value to a variable. Ex: **A:=128**

[x] = [y] ..... Used for evaluation; usually in a conditional test (as in **IF A = B THEN ...**) but can be used to assign the result to a variable for later use (as in **D:= A AND $F0**)

[x] <> [y] ..... Not equal to.

[x] != [1] [y] **....** Not equal to. This is an alternative to "<>".

[x] > [y] ..... Is greater than

[x] < [y] ..... Is less than

[x] - [y] ..... Subtraction, 8-bit unsigned

[x] + [y] ..... Addition, 8-bit unsigned

[x] * [y] ..... Multiplication, 8-bit unsigned

[x] / [y] ..... Division, 8-bit unsigned

[x] % [y] ..... Modulus, 8-bit unsigned

AOV[1] [x] ..... Enable arithmetic overflow and underflow detection where x = 1 enables error detection and x = 0 disables error detection. Without [x] is same as x = 0. Defaults to x = 1. Note that when detection is disabled, the result from an arithmetic operation will return the 8-bit result. Expect errors if not careful!

Note: The relational operators can be used with the **IF** keyword or used to assign their result to a variable for later use. See the *AttoBASIC Application Note* for more examples.

Operator

[x] AND [y] .... Logical AND between two 8-bit values

[x] &[1] [y] ..... Same as the AND operator

[x] OR [y] ..... Logical OR between two 8-bit values

[x] |[1] [y] ..... Same as the OR operator

[x] XOR [y] .... Logical Exclusive OR between two 8-bit values

[x] ^[1] [y] ..... Same as the XOR operator

[x] MOD[1] [y] .... Modulus of X / Y

[x] %[1] [y] ..... Same as the Modulus operator

LSL[1] [x] [y] .... Logical shift [x] left by [y] bits. If "y" is not specified then y = 1.

LSR[1] [x] [y] .... Logical shift [x] right by [y] bits. If "y" is not specified then y = 1.

COM[1] [x] ..... Compliment (1's compliment or bitwise inversion)

![1] [x] ..... Same as the COM operator

NEG[1] [x] ..... Negate (2's compliment)

SWAp[1] [x] ..... Swap low nibble with high nibble.

NBH[1] [x] ..... Return the high nibble of [x].

NBL[1] [x] ..... Return the low nibble of [x].

REV[1] [x] ..... Reverse the bit order of [x].

CRC [x y z] .... Return the CRC-8 of "x", "y", "z". Up to sixteen (16) values can be supplied with the **CRC** command. Without command line parameter calculates the CRC-8 of the data held in the DATA statement buffer. Ex1: **B:= CRC** assigns the CRC value of the data held in the DATA statement buffer to the variable B. Ex2: **PRX CRC $11 $22 $33 $44** prints the hexadecimal CRC value of the data **$11 $22 $33 $44**.

Note: Most operators can be directly used with the **IF** keyword but they must be presented in "reverse condition". They can also be used to assign their result to a variable, which can also be used with the **IF** keyword. See the *AttoBASIC Application Note* for more examples.

Instruction set of the AttoBASIC interpreter
Version: 2.34

Debug[1]

DEBug [x] ..... Enable/disable debug display, where x = 0 to disable debug display and x = 1 to enable debug display. Debug displays each line number BEFORE it is executed.
DUMP ..... Dump RAM memory in hex format EX: DUMP
VDUmp ..... Dump the contents of the variables [A..Z] Ex: VDUMP
EDUmp ..... Dump EEPROM memory in hex format EX: EDUMP
IDUmp ..... Dump I/O space in hex format EX: IDUMP
FILL [x][p][o][y][z] ..... Fill RAM memory, where x = fill value, p = starting page, o = starting offset, y/z = # of pages and bytes to fill to. Ex: **FILL $55 $01 $00 $00 $80** fills RAM with $55 from $0100 to $0180 ($0100 + $0008 = $0180).

System Clock

OSC [x] ..... Allows reading from or writing to the **OSCCAL** register, thereby allowing one to adjust the frequency of the internal RC oscillator. With [x] sets the value of the **OSCCAL** register. Without [x] reads the value from the **OSCCAL** register. Ex1: **OSC $81** stores the value of $81 to the **OSCCAL** register. Ex2: **PRX OSC** prints the value of the **OSCCAL** register.
CLK [x] ..... Allows reading from or writing to the **CLKPR** register, thereby allowing one to set the system clock divider of the main system clock. With [x] sets the value of the **CLKPR** register. Without [x] reads the value from the **CLKPR** register. Ex1: **CLK 2** stores the value of 2 (divide by 2) to the **CLKPR** register. Ex2: **PRX CLK** prints the value of the **CLKPR** register.

Notes: **1)** Refer to the appropriate MCU's data sheet for specifics on use of these registers. In particular, setting the system clock divider with the **CLK** command WILL affect all internal timing. **2)** This command is not available on all MCU's.

Volatile (local) Data Storage

DATa ..... Allows the use of up to sixteen (16) values to be stored in volatile memory by a program for later retrieval with the **READ** command. Ex: **DATA $AA $55 $00 $FF** stores the four (4) values.
REAd [x] ..... Allows access to the values stored in a previously executed DATA statement. Where 0 <= X <= 7 (zero inclusive) is optional and is a method to specify an index into the stored data. Each execution of the **READ** command will increment the pointer and return the next value in the list until all data has been read, at which time another **READ** will produce an error. Ex: **PRINT READ** will print the next available data value. Ex: **PRINT READ 2** will print the third value in the list. NOTE: Using "X" with **READ** does not increment the data pointer.
REStore ..... Reset the data pointer to the 1st value (position "0") and returns the number of bytes held in the data statement's buffer.

Notes: **1)** There is no specific requirement that the DATA, READ and RESTore commands be used in a program. They may be used immediate mode as temporary (volatile) storage. **2)** if the nRF24L01, SPI, 1-Wire® or TWI support is enabled, the DATA statement may contain up to 32 values. **3)** Data storage is limited to eight (8) values for 512 byte RAM parts.

Instruction set of the AttoBASIC interpreter
Version: 2.34

Byte I/O (Data and EEPROM space)

PEEk [P] [O]..... Read value from data space. Where [P] is page number and [O] is offset into the page. Ex: **PRX PEEK $04,$FF** reads the byte at $04FF.

POKe [X] [P] [O] . Write value [x] to data space. Where [P] is page number and [O] is offset into the page. Ex: **POKE A,$01,$00** (POKE VALUE, destination).

EER[1] [P] [O] ..... Read value from EEPROM. Where [P] is page number and [O] is offset into the page. Ex: **PRX EER $01,$FF** reads the byte at $01FF. *Note if only [p] is specified that the page is assumed to be "0" and [p] denotes the address within page 0.*

EEW[1] [X] [P] [O] .. Write value [x] to the EEPROM. Where [P] is page number and [O] is offset into the page. Ex: **EEW A,$01,$00** writes the value held in variable A to at $0100. Note that there is a special use for the EEW command; when the address specified is E2END+1, the entire EEPROM contents are erased. Ex: **EEW $00,$04,$00** specifies that location $0400 is to be erased on a Atmega328, which has and EEPROM with and ending address of $03FF. *Note if only [x] and [p] are specified that the page is assumed to be "0" and [p] denotes the address within page 0.*

VPG[1]..... Returns the page number in RAM (DESG) that AttoBASIC's internal variable are stored at. This aids if one wishes to access them using **PEEK** and/or **POKE** without having to consult the target build's MAP file.

@[str] [1] ..... Fetch symbol value. Returns the value of the named symbol's offset into the system variables page. Intended to be used in conjunction with the "VPG" command.

Notes: 1) Variables A-Z may be used, **2)** if only [P] is specified, it is used as [O], the offset into page zero], **3)** use caution when the EEP File System is enabled, **4)** When the EFS is used, bytes $10 through $1F are available for use. Improper use of the EEW command can corrupt the file system. **5)** When using the EEW command to bulk erase the EEPROM, the value for [x] is irrelevant as bulk erasing will return all bytes to 0xFF.

Bit I/O[1]

Note: *For the following Port I/O commands, substitute [p] for the port value (A..D, etc.) and [b] for the port bit number (0..7) if relevant for the MCU AttoBASIC has been compiled for. Examples are show for each command.*

Port Direction

OD[p] [b] ..... Output data direction register DDR[p] EX: OD[B] $FF
ID[p] [b] ..... Input from data direction register DDR[p] EX: J:= ID[C]
SD[p] [b] ..... Set bit in data direction register DDR[p] EX: SD[D] 3
CD[p] [b] ..... Clear bit in data direction register DDR[p] EX: CD[A] 3

Port Output

OP[p] [b] ..... Output PORT[p] EX: OP[A] $1A
SB[p] [b] ..... Set bit on PORT[p] EX: SB[B] 3
CB[p] [b] ..... Clear bit on PORT[p] EX: CB[B] 3
XB[p] [b] ..... XOR (toggle) bit on PORT[p] EX: XB[C] 3
PB[p] [b] ..... Pulse (momentary toggle) a bit's state on PORT[p] for 10mS. This command may also be used to send a stream of 10 mS pulses by embedding in a FOR-NEXT loop. EX1: **PBC 3** will toggle the state for PORTC3 for 10 mS. EX2: **PBC 3** is the equivalent to "**XBC 3 : DEL 1 : XBC 3**".

Instruction set of the AttoBASIC interpreter
Version: 2.34

Port Input

IN[p] ..... Input from PIN[p]. Ex: **J:= INA** will assign the 8-bit value from the PINA register to the variable "J".

IB[p] [b] ..... Input bit value from PIN[p]. [b] Ex: **IF IBD 2 THEN GOTO 100** will execute the command GOTO 100 if the value from PIND2 = 1.

Pulse Width Modulator[1]

PWM [x] [p] .... Pulse width modulation (8-bit) on OCxA/B pin. EX: **PWM 17,1** outputs duty cycle corresponding to "17" on channel 0 (OCxA).

PWE [x] [y] [p] Pulse width modulation (10 bit) on OCxA/B pin. EX: **PWE 2,0,1** outputs duty cycle corresponding to $200 on PWM channel 1 (OCxB).

PWO ..... PWM off for both OCxA and OCxB pins (does not affect the port's data direction register). Use PWM 0,[p] or PWE 0,0,[p] to turn of only one channel.

Notes: 1) On Attiny85, only 1 channel is supported on OC1B. **2)** On Attiny84/85, the PWE command is not available. **3)** Unless the SPI and Data File support is disabled, only 1 Channel is supported on Atmega88/168/328 devices, Otherwise, 2 Channels are supported. **4)** Without [p] is the same as p = 0, **5)** If after enabling both channels and one channel needs to be turned off, use PWM 0,p to do so, **6)** Variables A-Z may be used for x, y and/or p.

Analog Comparator[1]

ACO ..... Analog comparator output Ex: **IF ACO THEN PRINT A**. Prints the value of "A" if analog comparator output is high

ACS [x] ..... Analog comparator "AIN1" select. Selects the input pin to use for the analog comparator's internal "-" input. Where x = 0 to 4. Selects ADC0 to ADC3 (ADCx) when x = 0 to 3 and x=4 (to 7) for AIN1. Ex1: **ACS 2** Selects the ADC2 pin as the AIN1(-) input to the comparator. Ex2: **ACS 4** Selects the AIN1 pin for the AIN1(-) input to the comparator.

Notes: 1) The AIN0(+) input of the comparator is connected to the internal bandgap reference.

Analog to Digital Converter[1]

ADR [x] ..... ADR [x] initializes the ADC and sets the ADC reference to Internal or External. x = 0 for INT and x = 1 for EXT. Without [x] is same as x = 0, int. ref. Ex: ADR 1 selects external Vref for ADC. *[Note: this command must be executed before obtaining readings from the ADC]*

ADC [x] ..... 8-bit ADC conversion Ex1: **PRX ADC 0** Ex2: **A:= ADC 9** Ex3: **PRX ADC 15**. *[Refer to the appropriate AVR data sheet for valid ADC channel numbers as some AVR's support reading the on-chip temperature, 1.1 volt Vref and GND. This command does not error check 'x' for selected channel validity.]*

DS Interface[1]

DSD ..... Send a byte over the DS Interface as data EX: "**DSD $AA**" sends the value of $AA.

DSC ..... Send a byte over the DS Interface as a command EX: "**DSC C**" sends the value contained in the variable "C".

DSR ..... Read a byte from the DS Interface EX: "**PRX DSR**"

Notes: 1) The DS interface is enabled in the HEX file builds by default. 2) The ATTN and DATA pins must have external 10KΩ pull-ups. 3) The DSR command will return 255 if a timeout occurs. 4) transmission and reception errors will emit an error message but not abort a running program.

DALLAS 1-Wire Interface[1]

OWI ..... Initialize the 1-Wire® bus and return the status of the bus, where the status = "0" if no devices were present and "1" if at least one (1) device is present. Ex: **"S:= OWI"** assigns the status of the bus to the variable S.

OWW [x y z] .... Write data to the 1-Wire® bus, where the allowable number of data bytes is within the range of 1 to 32. EX1: "**OWW $AA**" sends the value of $AA to a device on the bus. EX2: "**OWW C**" sends the value stored in the variable C to a device on the bus.

OWR [n] ..... Read data from the 1-Wire® bus. Without "n" reads a single data byte from the 1-Wire® bus. Where "n" is specified, read "n" bytes from the 1-Wire® bus and save it in the DATA statement buffer. EX1: "**PRINT OWR**" prints the data read from a device on the bus. EX2: "**OWR 8**" reads 8 bytes of data read from a device on the 1-Wire® bus and places it into the DATA statement buffer for subsequent reading by the READ command.

Notes: 1) The 1-Wire interface is enabled in the HEX file builds by default. 2) The DQ pin must have an external 4.7KΩ to 10KΩ pull-up. 3) The OWR command will return 255 if a timeout occurs. 4) transmission and reception errors will emit an error message but not abort a running program.

DDS (Direct Digital Synthesis) Interface[1]

DDS [x] ..... Outputs a frequency on the defined port pin at the 6-BCD-digit frequency held in the X/Y/Z (or K/L/M) variables. x = 0 to disable DDS and x = 1 to enable DDS [X/Y/Z set first]. Without "x" is the same as x = 0 [disable]. The DDS output frequency range will be 0 to 25KHz in 1Hz steps. Ex (as separate commands): **X:= $01, Y:= $23, Z:= $45, DDS 1** will emit a 12,345 Hz square wave on the *DDSOut* pin.

Notes: 1) In BCD, each character position may only contain a number from 0 to 9 and must be specified in hexadecimal using the "**$**" keyword prefix. 2) The accuracy of the output frequency is directly related to the accuracy of the CPU's oscillator. 3) For MCU's with 512 bytes of RAM, use the variables K/L/M. 4) Due to interrupt servicing latency, some jitter will be seen on the output signal.

Input Capture[1]

ICG [x] ..... Initializes ICP mode on the ICP pin and sets Input Capture gate time to x[0..7] where x is optional (default 0). Ex: ICG 7 enables ICP registers and sets gate capture time to 1 second.

0 = disables the ICP function.
1 = 10mS gate time
2 = 20mS gate time
3 = 50mS gate time
4 = 100mS gate time
5 = 200mS gate time
6 = 500mS gate time
7 = 1000mS gate time

ICE [x] ..... Optionally sets the capture edge. Where x = 0 for falling

and 1 for rising (default is 1). Ex: ICE 1 set capture on rising edge.

ICP ..... Returns the low byte value and stores the high byte in variable 'Z'. The value held in 'Z', if any, is overwritten when executing this command, so be aware. Returns an error if there is a 16-bit overflow (and clears 'Z'). Ex: PRX ICP then PRX Z [*Note: the maximum capture frequency depends on the AVR's system clock. Consult the datasheet for specifics*]

Note: The input capture function works by checking the timer's TCNT register after a preset period of time. Therefore, a properly conditioned signal, preferably a square wave, must be presented to the "Tn" pin. Refer to the AttoBASIC Application Note for MCU specific pin assignments.

SPI Interface[1]

SPM [x] ..... MUST be called first to initializes the SPI hardware to operate in Mode [0..3]. Without [x] is same as x = 2; Master, Mode 2, F_CLK/16, MSB first. (Refer to the AVR data sheet for explanation of mode #'s)

SPO [x] ..... Optionally set MSB/LSB data order where x = 0 for MSB and x = 1 for LSB.

SPC [x] ..... Optionally set SPI clock to [0..7]. (Refer to the AVR data sheet for explanation of SPI clock dividers)

SPW [x .. Z] ... SPW sends data to the SPI bus. Ex 1: **SPW A B C** sends the data held in variables A, B and C to a device on the SPI bus. *Note that the state of the SPI_SS pin is not affected, thus the user must set the pin's state with the SPS command.*

SPW [x] ..... **Atmega88 only**: Write a single byte to the SPI bus.

SPW "abcdef" .... SPW sends the string data "abcdef" to the SPI bus. Ex: **SPW "Hello"** sends the string "Hello" to the SPI bus. [Note: do not mix values and strings with this command.]

SPR [n] ..... Read data from the SPI bus and place it into the DATA statement buffer for subsequent reading by the READ command. Where "n >= 0" and is a value representing the number of bytes to read from the SPI device. This command returns no value unless n = 0, then it returns one (1) byte from the device and does not use the DATA buffer. EX1: "**S:= SPR**" reads one (1) byte of data and assigns the value to the variable S. EX2: "**SPR 8 ; PRB READ**" reads eight (8) bytes of data into the DATA statement buffer and prints the first byte in binary. *Note that the state of the SPI_SS pin is not affected, thus the user must set the pin's state with the SPS command.*

SPR ..... **Atmega88 only**: Read a single byte from the SPI bus.

SPS [x] ..... Set the SPI_SS pin to logic level of [x]. Without "x" is the same as x = 1.

TWI (I2C) Interface[1]

TWI [x] ..... TWI must be called first to initialize the TWI interface. X = 0 for 400Kbps and x = 1 for 100Kbps clock. Without [x] is same as x = 0. Defaults to Master @ 400Kbps with PORT pull-ups enabled.

TWS ..... Assert a START condition on the bus. When the TWI interface is initialized, a START condition is asserted and the bus status is returned. However, the user must re-assert a START condition after a STOP condition to ready the bus for

the next message sequence.

TWP ..... Assert a STOP condition on the bus. The user must assert a STOP condition after the last message byte has been sent to or received from the slave or to abort a transfer in progress.

TWA [x] ..... **This command is obsolete and has been combined with the TWW command.**

TWW [x .. z] ... TWW sends data to the bus and returns the bus status. This command is also used to address a slave on the bus and will continue transferring data until a NACK is received, at which time it will abort and return the bus status. An ACK is always placed on the bus after a data byte is transmitted, even if the last data byte of a **TWW** command has been sent. This allows for multiple **TWW** commands to be used. Ex 1: **TWW $D0 B C** sends the data held in variables B and C to a slave on the bus responding at address $D0. Ex 2: **A:= TWW $A0 $55 $10** sends $55 and $10 to the slave on the bus responding at address $A0 and returns the bus status in variable A. Ex 3: **TWW $D1 ; A:= TWR** addresses the slave on the bus responding at $D1 then a separate **TWR** command retrieves the data and places it into variable A. [*Note: This command should be used after issuing a START condition to send the desired slave address as the 1st data byte. The user must insure bit 0 of the slave address contains the R(ead) or W(rite) indicator bit AND'ed or OR'ed with the 7-bit slave address before sending. The address may need to be left-shifted one bit position from the data sheet specification.*]

TWW "abcdef" .... TWW sends the string data "abcdef" to the bus and returns the bus status. Ex: **TWW "Hello"** sends the string "Hello" to the previously addressed slave on the bus and returns the bus status into variable A. [Note: do not mix values and strings with this command.]

TWR [n] ..... Read data from a slave on the TWI bus. If "n" is not supplied then a single byte is returned. If "n" is supplied then data is transferred and placed it into the DATA statement buffer for subsequent reading by the READ command. When "n" is supplied and "n > 0" then "n" is a value representing the number of bytes to read from the slave. EX1: "**PRINT TWR**" reads one (1) byte of data and prints it on the console. EX2: "**S:= TWR**" reads one (1) byte of data and assigns the value to the variable S. EX3: "**PRINT TWR 1**" reads one (1) byte of data into the DATA statement buffer and prints the status of the bus. EX4: "**K:= TWR 8**" reads eight (8) bytes of data into the DATA statement buffer and assigns the status of the bus to the variable K.

TWB ..... Queries the TWI status register for the last detected condition of the bus. The value returned is the exact value contained within the TWSR register. Ex: **IF $40 = TWB THEN PRINT "ACK returned~"** will print the message "ACK returned" if the status of the bus returns $40.

NOTES: 1) Unless otherwise specified, all commands return the status of the bus after the command has executed and finished. **2)** A 6.4MHz clock is required to operate the TWI interface at a 400K bus speed. Therefore using a clock below 6.4MHz will always initialize the TWI interface at 100K regardless of the value given for "x". **3)** If it is desired to use alternate pull-ups, they can be paralleled with the AVR's internal pull-ups without any issues. **4)** Failure of an

unresponsive device will cause a program to abort.

Real-time Counter[1]

The Real-time Counter commands are used to access an implementation of a 32-bit internal counter that increments sequentially after a preset period of time has occurred.

RTI ..... Sets the Real-time counter increment interval, the resolution, to "N", where: N = 0 for 1mS (default), N = 1 for 10mS, N = 2 for 100mS. Ex: "RTI A" will set the interval to the value stored in variable "A".

RTR ..... Resets the Real-time counter to zero.

RTP ..... Assign or print the 32-bit value of the Real-time timer. When assigned to a variable, nothing is printed to the console (see notes). When printed to the console, the full 32-bit value (10 digits) is printed (this option is intended to be used when capturing time-referenced data via the console and importing into a spreadsheet or the like). Ex: A: = RTP (assign low 8-bits to variable "A"). Ex: RTP (prints the 10-digit value or the RTC to the console).

RTC[N] ..... Returns the page offset of the system's 32-bit Real-time counter registers. Where 0 <= N <= 3. Use with the "@" (fetch) operator. Ex: "PRINT PEEK VPG@RTC3" returns the value held in the most significant digit of the RTC's 32-bit internal register.

NOTES: **1)** The RTC has an internal resolution of 1mS. **2)** There may be an intrinsic error in the Real-time counter, which is dependent on the master CPU clock frequency used. At CPU clock frequencies of 4, 8, 16 and 20MHz, a typical error of +0.2% will be seen. Other frequencies may yield different errors. **3)** At 1mS ("RTI 0"), the maximum count will yield ~49.71 days before "rolling over" to "0". At 100mS ("RTI 2"), the maximum count will yield ~4971 days before "rolling over" to "0". **4)** The "**RTP**" command can be used to assign the value held in the lower 8-bits of the Real-time Counter to a variable, thus time differences of up to 25.50 seconds can be directly measured. **5)** One must be aware that when assigning and comparing two values held in variables sampled from the RTC at different times, the second reading may have rolled over from 255 to 0 and be numerically less than the 1st sample. The "AOV" setting will not flag this.

Data File[1]

The Data File routines use a serial EEPROM attached to the SPI interface. The SS pin is used as the chip-select. Regardless of the device's internal page size, AttoBASIC pages are 256 bytes long with a maximum of 256 pages (16-bit addresses = 65,536 bytes). One use for this command set is as a chart recorder.

DFR [a] [p] .... Read data from Data File at address [a] of page [p]. EX: **DFR $80 4** reads the value from address $80 of page 4 (or 0x0480).

DFW [d] [a] [p] Write data [d] to Data File at address [a] of page [p]. EX: **DFW A 0 8** writes the value contained in variable "A" to address 0 of page 8 (or 0x0800).

DFL [d] ..... Log (write) data [d] to Data File and increment the internal address counter. EX: **DFL A** writes the value contained in variable "A" to the data file. Without [d] is the same as d = 0.

DFX ..... Reset the internal data file address counter to "0". The

address counter is also reset to "0" upon a hardware RESET or invoking the "RST" command.

DFF [n] ..... Set Data file dump stream format to [n] values per record. EX: **DFF 4** sets the dump format to four (4) values per record thus allowing data recording for multiple values per record. Without [n] is the same as n = 1. Note: there is no restriction on [n] except N != 0.

DFD [p] [n] .... Dump sequential data stream from Data File starting at address 0 of page [p] for [n] pages. EX: **DFS $10 1** prints 1 page of sequential data starting at address 0 of page $10 (or 0x1000). If the DFF command has specified N > 1 then each data value is separated by comma and each record by a CR/LF combination (ex: "0,1,2,3[cr/lf]"). This makes the data output CVS compliant for easy capture.

DFI [p] [n] .... Data File Initialize [n] pages starting from page [p]. EX: **DFI 0 1** initializes 1 page of the Data File staring at page 0. Without [n] erases one page. Without [p] and [n] starts erasure at page zero for one page. *Note: unless set with the DFB command, "0" is the fill value used.*

DFV [d] ..... Set the value to be used when initializing the Data File with the DFI command. EX: **DFV $55** sets the initialization value to "$55". Without [x] is the same as "0".

DFA[N] ..... Returns the page offset of the system's internal 16-bit data file address pointer register used with the DFL command. Where 0 <= N <= 1. Use with the "@" (fetch) operator. Ex: **PRINT PEEK VPG@DFA1** returns the value held in the most significant digit of the 16-bit internal address pointer register.

NOTES: 1) There is no test for the existence of an attached device. **2)** This version of software does not support additional SPI peripherals when the Serial EEPROM routines are enabled 3**)** It is the user's responsibility to insure the page number specified is within the attached device's range.

DHTxx Temperature and Humidity Sensor Interface[1]

The DHTxx sensor interface commands are used to read the temperature and humidity from DHT11, DHT22 and compatible devices.

DHT ..... Acquire the temperature reading from the DHTxx sensor. The result is selectable between Fahrenheit and Celsius using the DHU command. For Fahrenheit measurements, the DHT command outputs 0 to 176 degrees (the sign is always ignored). For Celsius measurements, the DHT command outputs 0 to 80 degrees when the sign is ignored or –40 to 80 degrees when the sign is recognized (bit 7 is the sign bit).

DHH ..... Acquire the humidity reading from the DHTxx sensor, which is in % Relative Humidity between 0 and 100.

DHR ..... Return the availability status of the DHTxx sensor. "1" means "busy" (less than 2 seconds since last read), while "0" means **R**eady to read.

DHU [x] ..... Set temperature unit. X = 0 for Fahrenheit (default) and X = 1 for Celsius. Without [x] returns the state of the setting.

DHS [x] ..... Recognize or ignore signed temperature readings. X = 0 to ignore (default) and X = 1 to recognize. Without [x] returns the state of the setting.

DHI ..... Returns the page offset of the DHT sensor's last read hu-

midity integer. Use with the "@" (fetch) operator.

DHD ..... Returns the page offset of the DHT sensor's last read humidity decimal (tenths). Use with the "@" (fetch) operator.

DTI ..... Returns the page offset of the DHT sensor's last read temperature integer. Use with the "@" (fetch) operator.

DTD ..... Returns the page offset of the DHT sensor's last read temperature decimal (tenths). Use with the "@" (fetch) operator.

NOTES: 1) The DHT sensor routines use the pin-change interrupts, therefore, the DHT data pin must be attached to a pin supporting pin-change interrupt. Consult the AVR's data sheet for further information. For the ATMega32U4, this is restricted to PORTB. **2)** An internal timer is monitored by the DHT routines to insure that at least 2 seconds has elapsed between reads. If not then the previous reading will be returned. **3)** The integer and decimal values for temperature and humidity are stored in RAM, which can be accessed using the "VPG" and "@" (fetch operator) in conjunction with the symbols "DTI" (integer temperature), "DTD" (decimal temperature), "DHI" (integer humidity) and "DHDD" (decimal humidity). **4)** An error will issue for a non-responsive DHT sensor but a running program will not abort. **5)** An error will issue for a data and checksum mismatch but a running program will not abort.

EEPROM File System (EFS) [1]

SAVe [x] ..... Save the current program to EEPROM (at the file number specified) Ex: **SAVE** [*Note: the SAVE command will complain if the program is too big for storage in EEPROM*]. Ex: **SAVE 1** will save the program in file handle 1.

LOAd [x] ..... Load program from EEPROM (from the file number specified) Ex: **LOAD 2** loads the program stored in file handle 2.

CAT ..... Print a CATalog of the programs held in the EFS, displaying the file handle, number of block bytes used and the 1st line of the program. Ex: **CAT**

ERA [x] ..... Erase the program stored in the specified file handle. Ex: ERA 3 erases the program stored in file handle 3.

INIt ..... INITialize the EFS. This effectively erases all programs. It should be used to "format" the EFS volume before initial use. Ex: **INIT**

NOTES: 1) Without [x] is the same as x = 0], **2)** There is approximately 9% memory overhead when using the EFS. **3)** When EFS support is disabled, only the SAVE and LOAD command are available and no file handles are required. **4)** EFS is disabled on the Mega88 by default. **5)** Each EFS block is 32 bytes in length and can store 3o bytes of program data per block.

Low-power using Sleep[1]

SLP [x] ..... If x = 0 then the AVR executes the low-power sleep instruction and waits for a user-enabled event to occur. If "x" is greater than zero then the AVR enters sleep and uses the watchdog timer as a "one-shot" time delay event source. Without [x] is the same as x = 0. *Refer to the **Watchdog Timer** section of the specific AVR datasheet for valid time-outs for "x" values greater than zero.* ***CAVEAT: if an interrupt source is not set and enabled BEFORE executing the "SLP" command, then the AVR will enter a perpetual sleep mode.***

NOTES: 1) If the build uses USB for serial I/o and an interrupt source is not enabled before executing **SLP 0** then a hardware reset will be required to re-

start AttoBASIC, otherwise, pressing any printable key on the keyboard will create a USART interrupt and break the **SLP 0** free. **2)** This instruction can be used instead of DELay if low-power is needed but precision timing is not. **3)** The AVR's "idle mode" is always used. 4) Any supported interrupt source may be used as the event trigger **5)** Proper use of the sleep mode requires that the user enable the desired interrupt source(s) before executing this instruction. **6)** Using the **POKE** statement allows programming the desired Interrupt Control Register and Interrupt Mask Registers **7)** When invoked, the watchdog timer is implemented in **"Interrupt Mode"**, however, the prescaler value of "0" is not available. **8)** If the RTC and/or DDS routines are enabled then the OCRxA interrupt source for each TIMER used (see source code for specific timer/register) is unavailable. However, the other OCRxB/C/D/E/F interrupts sources are available.

NORDIC SEMI NRF24L01 2.4GHz Transceiver[1]

RFI [m] [c] [b]. Initialize the transceiver mode, channel and data rate, where:

m = 0 disable (standby)
m = 1 for transmitter
m = 2 for receiver
m = 3 to set nRF24L01 registers to factory defaults

c = channel 0 to 127 (2.400 GHz + [c]MHz)

b = 0 for 1mbps
b = 1 for 2mbps

Without m, c and b defaults to m = 1, c = 1, b = 1. Note that c and b are not needed when m = 0 or m = 3. Ex1: **RFI 1 45 1** will enable transmitter mode using channel 44 with a 2 mbps data rate. *Note: ShockBurst™ is enabled by default.*

RFP [x]..... Set the RF transmitter's output power level or return the current level, where x = 0 for -18 dBm, x = 1 for -12 dBm, x = 2 for -6 dBm, x = 0 for 0 dBm. Ex1: **RFP 2** sets the RF transmitter power level to -6 dBm. Ex2: **P:= RFP** assigns the current power level to the variable "P".

RFC [c]..... Set the RF transceiver's channel to the value of "c", where c = 0 to 127. Without [x] returns the currently selected channel. Ex1: **RFC 22** sets the RF transceiver to channel 22 (2.444 GHz). Ex1: **PRI RFC** returns the currently selected channel.

RFA [p] [mac]... Set the MAC address for pipe "p", MSB to LSB order, where p = 0 through 5 for the RX pipes and p = 7 for the TX pipe. Ex1: Assuming that the MAC address is set for 5 bytes, **RFA 7 0xA5 0x44 0xD5 0xC2 0x02** sets the address of the TX pipe to hex A5:44:D5:C2:02. Ex2: **RFA 3 0xAA** sets the LSB of the MAC address of RX pipe "3" to hex AA.

*Notes: **1)** The values(s) of "mac" can be 1, 3, 4 or 5 bytes in length, depending on the pipe and the setting of register 3, **2)** For RX pipes 2 through 5, only the LSB is used, **3)** For TX, RX0 and RX1 pipes, from 3 to 5 bytes can be assigned, **4)** Incorrect widths will produce an error.*

RFB [p] [x]..... Set the size of an RX pipe payload. Use x = 0 to disable a pipe. Without x returns the payload size of pipe "p". Ex1: **RFB 1 0x20** sets payload size of RX pipe "1" to 32 bytes. Ex2: **RFB 5 0** disables RX pipe "5". Ex3: PRI **PRI RFB 1** returns the payload size of RX pipe "1".

RFE [x] ..... Enable or disable the transceiver (CE pin), where x = 0 to disable (TX mode), x = 1 to enable (RX mode) and x = 2 to

pulse (initiate a TX). Without [x] is the same as x = 0. Ex: **RFE 1** enables the transceiver in RX mode.

RFD [x] ..... Set the data rate of the transceiver, where x = 0 for 1 mbps and x = 1 for 2 mbps. Without [x] returns the current data rate. Ex1: **RFD 1** sets the data rate to 2 mbps. Ex2: **PRI RFD** prints the current data rate to the console.

RFT [x y z] .... Transmit the data "x", "y", "z", up to thirty-two (32) bytes. This command always returns the status of the TX FIFO buffer, where 0 = data in FIFO buffer and room available, 1 = TX FIFO buffer empty, 2 = TX FIFO buffer full and 3 = FIFO buffer blocked (due to last transmission failure). Without "x" will attempt to retransmit the last failed payload packet. Ex1: **RFT 1 2 3 4 5 6 7 8** transmits the values 1, 2, 3, 4, 5, 6, 7 and 8 to a listening receiver. Ex2: **PRINT RFT 0 1 2 3 A B C D** transmits the values of "0", "1", "2", "3" and the values held in variables A, B, C, and D to a listening receiver then returns the status of the TX FIFO. Ex3: **A:= RFT** attempts to retransmit the last failed payload and assigns the value of the TX FIFO flags to the variable A.

RFX ..... Initiates a data <u>transfer</u> from the RX FIFO to the **DATA** statement's buffer. The **READ** command is then used to access the data. It is the equivalent of using a **DATA** statement. The number of the pipe whose data is in the FIFO is automatically detected and returned. If no data is available then value of "255" is returned. Ex1: **PRI RFX** detects which pipe has data, places the data from the pipe buffer into the **DATA** statement's buffer then prints the number of the pipe whose data was transferred. Ex2: **A:= RFX** detects which pipe has data, places the data from the pipe buffer into the **DATA** statement's buffer then assigns the number of the pipe whose data was transferred to the variable A.

RFF [x] Flush the TX/RX FIFO's and clear the interrupt flags in the status register (7), where x = 0 to clear both TX and RX FIFO's, x = 1 to clear the TX FIFO and x = 2 to clear the RX FIFO. Without "x" is the same as x = 0. Ex1: **RFF** clears both TX and RX FIFO's. Ex2: **RFF 2** clears the RX FIFO.

RFW [r] [v] .... Write a value to one of the nRF24L01's registers, where r = the register number and v = the value to write. Ex1: **RFW R V** writes the value held in variable V to the register value held in then variable R.

RFR [r] ..... Read one of the nRF24L01's registers and return its value, where r = the register number. Without [x] returns the FIFO status register (0x17). Ex1: **RFR** returns the status of register 0x17. Ex2: **PRI RFR 7** returns the value of the STATUS register. *<u>Note:</u> for multi-byte registers, this command returns only the 1st byte of the register's value.*

NOTE: 1) Unless a command specifically states a value will be returned, the value of status register 7 is returned. **2)** After the **RFI** command is executed, the transceiver is left in a powered-down state and the **RFE** command must be executed. **3)** Valid RX pipes are 0 to 5. **4)** Consult the nRF24L01(+) data sheet for the factory assigned MAC addresses and widths, **5)** the nRF24L01's IRQ pin is not (yet) supported, **6)** Caveat: only use channels 0 to 83 if operating under "U.S. jurisdiction". **7)** Consult the nRF24L01(+) datasheet for calculation of data rate bit values, channel frequencies, etc. **8)** The 250 kbps data rate is not directly supported, use the **RFR** command. **9)** If using external logic to expand the SPI port, the user must insure the nRF24L01(+) is enabled before using these commands.