PicoCount Serial Communications Protocol

Developers Edition

Last Updated 06/07/2017

Introduction

The PicoCount counters use a packet form of serial communications. This was chosen to maximize error detection and data integrity, and to minimize power consumption in the PicoCount counter. There are two types of packets, command packets which are generated by the host computer system and sent to the PicoCount, and response packets which are sent by the PicoCount in response to commands. All commands have a response. In this “Developers Edition” there are some “protected” commands left out that are used in manufacturing, engineering and test.

Communications is at 115200 Baud, 8 data bits, no parity, 1 stop bit.
When downloading memory, and if the device will support it, download occurs at up to 921600 Baud (this is done under software control).

Each packet is composed of a string of characters, to help make the syntax clear, there are certain general syntax symbols used in this document:

<n> n is any single byte number between 0 and 255.
<nn> nn is a two byte number between 0 and 65535.
<ee> ee is a two byte EEPROM address between 0 and 4095.
(data) data is a number of data bytes specified by the byte count.
<cs> cs is a two byte simple checksum of the command byte, the byte count, and the data.

All single byte data is considered unsigned 8 bit.
All two byte data is considered unsigned 16 bit with the low byte first.

The PicoCount counters are optimized for extremely low power consumption and will run for 10 years or more on the built-in battery. It is very important that when in communication with the PicoCount counters, to minimize how often you communicate with it, especially in idle conditions. The PicoCount is always in a sleeping mode and is woken up with every command (or character) sent to it, so it is very important to not send extraneous characters to the counter and to not interrogate it a high data rates when the unit is in idle condition.

Most of the commands execute very quickly, however, you must keep timing issues in mind when executing the commands. If you always wait for the response to be completely received, then you can keep from getting into trouble. Care must be taken when changing Baud rates and when executing the Bootloader, extra timing and/or tests must be run to insure proper operation.

When trying to stay in “constant” communication with software, one should pace commands that check for the presence of the counter to ¼ or ½ second. Also there should be a “user not present” timeout that occurs after a few minutes of communication with no user interaction (to reduce unnecessary battery drain). Avoid repeated downloads of the same data to help minimize battery drain. We have put pacing tests into the firmware that will cause the Picocount to cease communicating if too many commands are being sent to it in a period of time. This test is renewed each hour. If during development the unit ceases communication -- that is the most likely cause -- you will have to wait until the top of the next hour at which time the unit will resume communications.

Software for General Use

If you are developing software for use with our products outside of just your company, for general public use, please notify us and send us a copy of your software that we can install to review and test. We will promote your software on our website at no charge.

The General Command Syntax

<0><0>]C<0><cs>
or
<0><0>]C<n>(data)<cs>
or
<0><0>@C<0><cs>
or
<0><0>@C<n>(data)<cs>

All commands begin with two bytes of all zeros <0>, or NULL. These bytes wake the PicoCount from sleep and gives it enough time to prepare the serial data input device (UART) for receiving the rest of the data packet. Use of any character other than NULL for the wake-up will give unpredictable results, since the UART may not get properly “synced” to the data packet.

Byte 1&2 are zero (null character | binary 0).
These are the command wake-up bytes.

Byte 3 is an ASCII “]” (close brace character), or an ASCII “@” (at symbol character).
This is the command start byte.

Byte 4 is an ASCII printable character.
This is the command byte.

Byte 5 is the packet data count. If data count is zero there will be no data bytes.
This byte determines how many data bytes are to follow.

Byte 6 is first byte of data, if byte 5 is non-zero.

Last (byte - 1) is the low byte of the 16 bit checksum.

Last byte is the high byte of the 16 bit checksum.

Please note: When calculating the 16 bit checksum math should be done with a 16 bit unsigned integer, that way if the 16 bit value overflows it will just wrap around. If you do the checksum calculations with a larger integer you just need to truncate the number to the lowest 16 bits, which will give the same result.

The General Response Syntax

<ACK><0><cs>
or
<ACK><n>(data)<cs>
or
<ACK><255><nn>(data)<cs>
or
<NAK>

Byte 1:
If command was successful:
Byte 1 is always an ASCII ACK character (0x06).
This byte is the start of the response.
If the command was not successful:
Byte 1 is always an ASCII NAK character (0x15).
This is only byte sent, the response is finished.

Byte 2:
If byte 2 is 0 to 254:
Byte 2 is the data count.
If byte 2 is 255:
The next two bytes are a 16 bit data count.

Byte 3:
If byte 2 is non-zero and not 255:
Byte 3 is the first byte of data.
If byte 2 is 255:
Byte 3 is the low byte of the data count.

Byte 4:
If byte 2 is a one:
Byte 4 is the low byte of the checksum
If byte 2 is greater than 1 and not 255:
Byte 4 is the second byte of data.
If byte 2 is 255:
Byte 4 is the high byte of the data count.

Last (byte - 1) is the low byte of the 16 bit checksum.

Last byte is the high byte of the 16 bit checksum.

Please note: When calculating the 16 bit checksum math should be done with a 16 bit unsigned integer, that way if the 16 bit value overflows it will just wrap around. If you do the checksum calculations with a larger integer you just need to truncate the number to the lowest 16 bits, which will give the same result.

PicoCount Commands and Responses

General notes on protocol and commands

Our commands have standard convention that we will try to stick with.

Commands that begin with a **]** (closing bracket) are **generic** commands that we intend to use in all our products from this point out. So by using these commands you should even be able to get information on new products at least at some level.

Commands that begin with an **@** (at symbol) are **product-specific** commands that will only work with that model. So after querying the model of a device, if it is an unknown model, @ commands should not be used.

Lowercase commands are commands that write data or change settings, and uppercase commands are all others. Most commands are uppercase.

Generic Commands

]A - Read the Timeout Days Remaining

The PicoCount counters may be put into a **timeout** mode. In this mode, the counter will behave in all ways like a normal counter until the number of days specified in the **timeout** days counter reaches zero, at which time the counter will not allow downloads. The **timeout** days can be extended or the **timeout** feature can be disabled in the field by calling the factory and getting a new authorization code. If the returning data byte is 255, then the **timeout** feature is not active. If the returning data byte is 0, then the unit is in the timed-out state.

Command: **<0><0>]A<0>A<0>**
0x00 0x00 0x5d 0x41 0x00 0x41 0x00

Response: **<ACK><1><byte><cs>**
0x06 0x01 0x2d 0x2e 0x00
This unit has 45 days remaining in this example.

]a - Write an Activation Code to the PicoCount

The activation code is a 12 character encrypted string that is generated by the Activation Code Generator software. When an activation code is written to the PicoCount, it will first de-crypt the code, and if it matches expected parameters, it will then activate the desired feature. This feature could be the unit **timeout**, in which case the days of **timeout** encrypted in the activation code will be set into the unit **timeout**. Or this feature may be to zero the passwords (both **admin** and **working**). This command, if successful, will deactivate the current key.

Command: **<0><0>]a<12>[String12]<cs>**
0x00 0x00 0x5d 0x61 0x0c 0x53 ….. 0x43 0x66 0x05

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

]B - Go to Bootloader

The PicoCount counters can be firmware upgraded in the field. The bootloader is a special protected portion of the firmware that facilitates this. During upgrade, the bootloader is called to perform the upgrade. The bootloader has its own communications protocol which is not covered in this document.

Command: **<0><0>]B<0>B<0>**
0x00 0x00 0x5d 0x42 0x00 0x42 0x00

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

]b - Change Baud Rate

The PicoCount counter normally runs at 115200 Baud. However, it is desirable to speed up the Baudrate during memory downloads to shorten the download time. This command is used to speed up and slow down the communications rate. If a Baudrate higher than 115200 Baud is specified, the unit will revert back to 115200 Baud after 2 seconds, unless the **Read NAND Block @R** command is issued which extends the higher Baudrate for 2 seconds each time it is issued..

The allowed values for **b** in the command (below):

0 - 115200 Baud.
1 - 230400 Baud.
2 - 460800 Baud.
3 - 921600 Baud.

Command: **<0><0>]b<1><b><cs>**
0x00 0x00 0x5d 0x62 0x01 0x03 0x66 0x00
In this example Baudrate is being set to 921600.

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
The acknowledge is always sent at the command Baudrate, **before** switching Baudrates.

]C - Communications Check

This command is used to verify that the PicoCount is communicating with the software. Whenever the PicoCount sees this command, it just issues an ACK response. No response means the PicoCount is not connected for some reason. You should never see a NAK response in normal operations, if one is received, there could be a hardware issue, like a faulty download cable.

Command: **<0><0>]C<0>C<0>**
0x00 0x00 0x5d 0x43 0x00 0x43 0x00

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

]E - Read User EEPROM Byte

The PicoCount uses its internal EEPROM to store certain parameters and counters, this command is used to read a byte from the EEPROM. The PicoCount has 2048 bytes of EEPROM of which all locations below 1024 are reserved for the PicoCount. This command will only work for locations 0x0400 to 0x07ff.

Command: **<0><0>]E<2><ee><cs>**
0x00 0x00 0x5d 0x45 0x02 0x0a 0x04 0x55 0x00
In this example we are reading the byte at EEPROM location 0x040a (1034).

Response: **<ACK><1><byte><cs>**
0x06 0x01 0x75 0x76 0x00
In this example the byte at EEPROM location 0x04a (1034) has a 0x75 (117) in it.

]e - Write User EEPROM Byte

The PicoCount uses its internal EEPROM to store certain parameters and counters, this command is used to write a byte to the EEPROM. The PicoCount has 2048 bytes of EEPROM of which all locations below 1024 are reserved for the PicoCount. This command will only work for locations 0x0400 to 0x07ff.

Command: **<0><0>]e<3><ee><byte><cs>**
0x00 0x00 0x5d 0x65 0x03 0x0a 0x04 0x75 0xe4 0x00
In this example we are writing 0x75 (117) to the EEPROM location 0x040a (1034).

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

]G - Read Battery Voltage

The PicoCounts internal battery voltage is read with this command. The voltage is nominally 3.00 Volts dropping to 2.70 Volts near end of life. This reading is used to give an early warning of battery failure. The voltage reading is calibrated in production. The reading is a two byte value of between 0 and 325 which represents 0.00 to 3.25 Volts.

Command: **<0><0>]G<0>G<0>**
0x00 0x00 0x5d 0x47 0x00 0x47 0x00

Response: **<ACK><2><nn><cs>**
0x06 0x02 0x31 0x01 0x34 0x00
In this example the voltage reads 0x0131 (3.05) Volts.

]H - Calculate and Read Firmware Checksum

This command is used to verify that the internal firmware of the PicoCount has not been corrupted. It is one of many health checks run each time the PicoCount is connected to the software. If a bad checksum is detected, it can be assumed that the internal firmware is corrupt and the unit should be immediately updated with the latest firmware by connecting the unit to our TrafficViewer Pro software. Following the command is a two byte firmware size. The checksum is calculated from location 0 of the CPU flash up to the two byte size specified, then sent back in response. Each firmware version will have it's own unique size.

Following are the sizes and valid checksums for various firmware versions:

Unit Type	Firmware Version	Firmware Size	Checksum
PicoCount 2500	2.30	15122	31272
PicoCount 2500	2.35	15272	25645
PicoCount 4500	2.30	15220	50563
PicoCount 4500	2.35	15208	7920

Additional size and checksum information can be provided for other versions upon request.

Command: **<0><0>]H<2><nn><cs>**
0x00 0x00 0x5d 0x48 0x02 0x12 0x3b 0x97 0x00
In this example we are wanting the checksum for 0x0000 to 0x3b12 of CPU flash.

Response: **<ACK><2><nn><cs>**
0x06 0x02 0x28 0x7a 0xa4 0x00
In this example the computed checksum was 0x7a28.

]I - Read Unit ID

The PicoCount has a 32 byte unit ID field which the user can program.

Command: **<0><0>]I<0>I<0>**
0x00 0x00 0x5d 0x49 0x00 0x49 0x00

Response: **<ACK><32><string32><cs>**
0x06 0x20 0x48 0x65 0x6c 0x6c 0x6f 0x00 0x00 …. 0x00 0x14 0x02
In this example the ID reads “Hello”.

]i - Write Unit ID

The PicoCount has a 32 byte unit ID field which the user can program.

Command: **<0><0>]i<32><string32><cs>**
0x00 0x00 0x5d 0x69 0x20 0x48 0x65 0x6c 0x6c 0x6f 0x00 0x00 …. 0x00 0x7d 0x02
In this example we are writing a unit ID of “Hello” to the unit.

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

]K - Get a Key from the PicoCount

This command is used to generate an 8 character unique key, or indicate that a key has already been supplied, but not used or timed out.. The key is used by the Activation Code Generator program to construct an activation code for the PicoCount. The user has up to 48 hours to use this key. If a new key is requested before 48 hours, the old key will expire. This command takes either a 0 or a 1 for an argument. If the argument is a 0, then this command will return an acknowledge if there is already an active (unused) key, else, it will return a **NAK**. If the argument is a 1, then this command will return an 8 digit unique key, at the same time resetting the key timeout to 48 hours.

Command: **<0><0>]K<1><1><cs>**
0x00 0x00 0x5d 0x4b 0x01 0x01 0x4d 0x00

Response: **<ACK><8><string8><cs>**
0x06 0x08 0x41 0x4a 0x42 0x43 0x51 0x49 0x4c 0x46 0x52 0x07 0x02
In this example the key returned is “AJBQILFR”..

Command: **<0><0>]K<1><0><cs>**
0x00 0x00 0x5d 0x4b 0x01 0x00 0x4c 0x00

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
In this example the last key requested is still active (unused).

or

Response: **<NAK>**
0x15
In this example there is no active (unused) key available.

]M - Read Unit Memory Information

The PicoCount uses Gigabit NAND Flash memories to store data. This command returns information regarding the memory chip. In the future, we may have to utilize different memory devices and this information will allow the software to properly handle the memory differences. Other products utilize different types of memories so their responses would be different. Use this command to determine the memory configuration and how much data is currently stored in the unit for downloading the data.

For the NAND flash 13 bytes of data are returned:
Memory type: 1 byte
NAND page size: 2 bytes
NAND block size: 2 bytes
NAND max blocks: 2 bytes
Current data page pointer: 2 bytes
Current data block pointer: 2 bytes
Current buffer pointer: 2 bytes

Command: **<0><0>]M<0>M<0>**
0x00 0x00 0x5d 0x4d 0x00 0x4d 0x00

Response: **<ACK><13><info><cs>**
0x06 0x0d 0x02 0x00 0x08 0x40 0x00 0x00 0x08 0x12 0x00 0x3c 0x00 0x5f 0x02 0x0e 0x01
A typical response to this command.

Brief description of the values for NAND flash storage:
Memory type: The memory type will return a value of 2.
NAND page size: This is the number of bytes per page for the current device.
NAND block size: This is the number of pages per block for the current device.
NAND max blocks: This is the total number of blocks the device's memory has.
Current data page pointer: This is a pointer to the page where the next page of data will be written. (The page pointer starts at 0, so this is the same as the number of pages that have already been written to the current block.) The pointer will be incremented immediately when the buffer fills causing a new page to be written and it is set to 0 when a new block starts.
Current data block pointer: The is a pointer to the block where the next page of data will be written. (The block pointer starts at 0, so this is the same as the number of full blocks that have been written.) The pointer will be incremented immediately when the last page of a block is written.
Current buffer pointer: This is a pointer to the byte location where the next byte of data will be written. (The buffer pointer starts at 0, so this is the same as the number of bytes that have been written to the buffer.) The pointer is incremented each time a byte is written to the buffer. This gets set to 0 when the buffer fills and the buffer is written to the location of the current page pointer.

]S - Read Serial Number and Manufactured Date

This command returns a 10 byte unit serial number (we use 8 most of the time) and the date of manufacture (1 byte for day of month, 1 byte for month, and 2 bytes for year). NOTE: The date of manufacture may be unused in many cases (but it can be approximated based on the serial number).

Command: **<0><0>]S<0>S<0>**
0x00 0x00 0x5d 0x53 0x00 0x53 0x00

Response: **<ACK><14><info><cs>**
0x06 0x0e 0x31 0x31 0x31 0x30 0x30 0x33 0x30 0x31 0x00 0x00 0x03 0x0a 0xdb 0x07 0x13 0x01
In this case the serial number is “11100301” and the manufactured date is 3rd day of October, 2011.

]V - Read Model and Firmware Version Number

This command returns the model of counter and the firmware revision level currently in the unit. It is a 23 character string with the first 16 characters the model of the counter (padded with spaces), and the last 7 characters are the firmware revision level (i.e. V1.07A). NOTE: This command is very important for future firmware changes. The version and model will determine what capabilities (new features or changes) a unit's firmware has -- as products develop it is sometimes common to test against the version before issuing certain (new) commands.

Command: **<0><0>]V<0>V<0>**
0x00 0x00 0x5d 0x56 0x00 0x56 0x00

Response: **<ACK><23><info><cs>**
0x06 0x0e 0x50 0x43 0x2d 0x32 0x35 0x30 0x30 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x56 0x31 0x2e 0x30 0x37 0x41 0x20 0x32 0x04
In this case the model is "PC2500 " and the version is "V1.07A "

]z - Zero Data

This command zero's all of the data pointers and updates the PicoCount's internal clock with the current PC date/time. The date/time is loaded as 8 bytes, hundredths of seconds, seconds, minutes, hours, day of month, month, low byte of year, high byte of year.

Command: **<0><0>]z<8><string8><cs>**
0x00 0x00 0x5d 0x7a 0x08 0x00 0x0d 0x2c 0x07 0x03 0x0a 0xdb 0x07 0x1b 0x01
In this example we are writing the date/time 07:42:13.00 on 3rd of October, 2011

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

Product-Specific Commands

@D - Read unit hardware dwell

This command reads the hardware dwell time of the PicoCount units. This command returns a zero if the hardware is not supporting the programmable dwells. Otherwise if it is a PC2500, it returns a byte of 32 to 254, the delay is calculated from dwell=(256-byte)/2 -- (1 to 112ms). If it is a PC4500 it returns a byte of 1 to 255 which corresponds to 1 to 255ms.

Command: **<0><0>@D<0>D<0>**
0x00 0x00 0x40 0x44 0x00 0x44 0x00

Response: **<ACK><1><d><cs>**
0x06 0x01 0xc4 0xc5 0x00
In this example we read a dwell time of 30ms from a PicoCount 2500. 0xc4 is 196, so dwell = (256-196)/2 = 30.
In this example we read a dwell time of 196ms from a PicoCount 4500.

@d - Set unit hardware dwell

This command sets the hardware dwell time of the PicoCount units. For the PC2500 the byte to send is computed from byte=256-(dwell*2), and must be from 64 to 236 (5ms to 127ms). For the PC4500 the byte to send is the dwell time in milliseconds. ***Note, this command should be used very carefully.*** *We recommend leaving it a 30ms which experience has shown to be the best overall. We then implement longer dwells in software when needed for filtering. There may be special permanent setups where a longer hardware dwell setting might be justified to reduce the data file sizes. Be aware though, if you collect data with the hardware set high - say 100ms, then later decide you really needed a 50ms dwell, you are out of luck. Our TrafficViewer Pro software cannot set the minimum dwell below the unit's hardware dwell.*

Command: **<0><0>@d<1><d><cs>**
0x00 0x00 0x40 0x64 0x01 0xc4 0x69 0x01
This example shows a dwell time of 30ms for the PC2500. (byte=256-30*2=196 which is 0xc4).
This example shows a dwell time of 196ms for the PC4500.

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

@F - Show Live Data

This command activates the live data operation of the PicoCount for 300 seconds, after which the PicoCount will terminate the live data operation. If you need to last longer than 300 seconds, then you can just reissue the command before the 300 seconds has elapsed to keep your live data totals from being reset to zero. While live data is active you can issue the **@L** command to read the totalizing counters.

Also during live data the counter outputs a single letter ('a', 'b', 'c', or 'd') each time a hit is recorded. The letter indicates which channel was hit and is in real-time. You could use this information to generate your own totalizers. Note, the letters are accurately issued by the PicoCount unit in real-time, however, most PC serial ports (especially USB ones) are notoriously non-real-time.

This command also records a special (14) timestamp into memory. It is issued by the CountBuddy devices when they first connect to the PicoCounts. So can be used to indicate start of a new study in multi-study uses.

Command: **<0><0>@F<0>F<0>**
0x00 0x00 0x40 0x46 0x00 0x46 0x00

.Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00

@I - (uppercase i) - Show Unit Status Information

This command returns the current date/time in the unit, and the start date/time (which is the last time the unit data was zeroed/cleared) from the PicoCount.

Current date/time is in 8 bytes:
milliseconds: 0-127 (128 counts per second).
seconds: 0-59
minutes: 0-59
hour: 0-23
day: 0-31
month: 0-11
year: Two byte unsigned integer (Normal range: 2000-20xx)

Start date/time is in 7 bytes:
seconds: 0-59
minutes: 0-59
hour: 0-23
day: 0-31
month: 0-11
year: Two byte unsigned integer (Normal range: 2000-20xx)

Two more bytes follow, which are currently reserved for future use. For now they are unused.

Command: **<0><0>@I<0>I<0>**
0x00 0x00 0x40 0x49 0x00 0x49 0x00

.Response: **<ACK><17>(data)<cs>**
0x06 0x11 0x55 0x1e 0x03 0x0d 0x03 0x0a 0xbd 0x07 0x37 0x21 0x07 0x03 0x0a 0xbd 0x07 0x05 0x05 0x9f 0x02
In this example current date/time was 13:3:30.66 on 3rd day of October 2011, start date/time was 7:33:55 on 3rd day of October 2011.

@L - Read Live Data

This command reads the live data counters which are 16 bit unsigned for each channel. These counters are enabled by the **@F** command.

Command: **<0><0>@L<0>L<0>**
0x00 0x00 0x40 0x4c 0x00 0x4c 0x00

Response: **<ACK><4><nn><nn><cs>**
0x06 0x04 0x24 0x00 0x3a 0x00 0x62 0x00
In this example the A channel counter is reading 36 and the B channel counter is reading 58.

@P - Compare PicoCount Passwords

This command compares the sent string against the **working** or **admin** passwords stored in the PicoCount 2500. The first data byte of the command indicates whether the next 12 bytes are the **working** password (value=0) or the **admin** password (value=1). If the **working** password is sent, but it is not enabled, this function will return a **NAK** character. If the **working** password is enabled and matches, then the PicoCount will be unlocked. The PicoCount will remain unlocked until there has been no comm activity for 30 seconds, at which time it will lock. If the **admin** password is sent and does not match, the PicoCount will return a **NAK** character. If the **admin** password is sent and it matches, then PicoCount will set an **update** flag which will allow one locked command to be executed (this would normally be the **@p** command, used to change passwords).

Command: **<0><0>@P<13><0>HELLO WORLD1<0><170><3>**
0x00 0x00 0x40 0x50 0x0d 0x00 0x48 0x45 0x4c 0x4c 0x4f 0x20 0x57 0x4f 0x52 0x4c 0x44 0x31 0x00 0xaa 0x03

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
In this example the PicoCount was locked and we sent the **working** password, which matched, and unlocked the PicoCount.

@p - Enable/Disable and set PicoCount Passwords

This command enables/disables password feature in the PicoCount 2500, and allows you to set passwords. The first data byte of the command indicates whether the next 12 bytes are the **working** password (value=0) or the **admin** password (value=1). Only the administrator may enable/disable the **working** password. If the **working** password is sent, the PicoCount will replace the current **working** password with the new one if the **update** flag is set (see the compare passwords command above). If the first character of the new password is non-**NUL**, then the **working** password will be enabled. If the first character of the new password is a **NUL** character, then the **working** password will be disabled. If a **working** password is sent, and the PicoCount is unlocked, then the password will be changed to the new password, provided the first character is a non-**NUL** character, otherwise a **NAK** will be returned. If the **working** password is sent but neither condition from above applies, then the PicoCount will return a **NAK** character. If the **admin** password is sent and the **update** flag in the PicoCount is set, then the **admin** password will be changed to the new password. If the **admin** password is sent and the **update** flag is not set, then the PicoCount will return a **NAK** character. Note that the password will terminate with the first **NUL** character.

Command: **<0><0>@p<13><0>HELLO WORLD1<0><202><3>**
0x00 0x00 0x40 0x70 0x0d 0x00 0x48 0x45 0x4c 0x4c 0x4f 0x20 0x57 0x4f 0x52 0x4c 0x44 0x31 0x00 0xca 0x03

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
In this example the PicoCount was unlocked and we changed the **working** password to “HELLO WORLD1”.

Command: **<0> <0>@p<13><1>HELLO<0><0><0><0><0><0><0><242><1>**
0x00 0x00 0x40 0x70 0x0d 0x01 0x48 0x45 0x4c 0x4c 0x4f 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf2 0x01

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
In this example the **update** flag was set in the PicoCount and we changed the **admin** password to “HELLO”. Remember the password terminates on the first NUL character..

Command: **<0><0>@p<13><0><0><0><0><0><0><0><0><0><0><0><0><125><0>**
0x00 0x00 0x40 0x70 0x0d 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x7d 0x00

Response: **<ACK><0><0><0>**
0x06 0x00 0x00 0x00
In this example the **update** flag was set in the PicoCount and we are disabling the **working** password feature.

@R - Read a Page of Data From NAND

This command retrieves a page of data from the PicoCounts NAND Flash, or the internal data buffer. The command sends the page to be retrieved, and the block number to retrieve the page from. The current valid page range is 0 .. 63, and the block range is 0 .. 4095.

If the page is specified as 0xFF (255), then the internal buffer will be output instead. In this case the block number is ignored and can be any value.

<byte> is the page number as a single byte.
<nn> is the block number as two bytes.

Command: **<0><0>@R<3><byte><nn><cs>**
0x00 0x00 0x40 0x52 0x03 0x0a 0x03 0x00 0x62 0x00
In this example, we want to read page 10 of block 3.

Response: **<ACK><255>(2-bytes-pagesize)<nn>(data)<cs>**
0x06 0xff 0x00 0x08 0x3a 0x00 … 0x03 0x62 0xf2
In this example page size is 2048 bytes.

A note on the internal buffer: The internal buffer is where the most recent data is kept. It is a temporary buffer and that contains all data that has not yet been written to the NAND flash. This buffer will be written to a flash page when it reaches a full page of data at that time the buffer will be cleared by setting all bytes of the buffer to 0xFF.

@s - Enable Real-Time Streaming (Beginning with V2.37)

This command starts real-time hit data streaming. Streaming is stopped when **any** other command is issued. During streaming, a data packet is sent each time there is a hose hit. Also, once streaming starts, there is a 'sync' packet sent each minute of no activity so the host knows the unit is still connected.

There are 3 streaming packet types:

1 - Full timestamps are sent each hit (12 bytes).
2. - A hit totalizer is sent each hit (9 bytes)
3. - A single character ('a','b','c',or 'd') is sent each hit.

Command: **<0><0>@s<1><1><117><0>**
0x00 0x00 0x40 0x73 0x01 0x01 0x75 0x00

Response: **<6><8><c><a><tttttt><cs> ……..**
0x06 0x08 0x00 0x00 0x85 0x4b 0x1f 0x00 0x00 0x00 0xf7 0x00 ---------->
In this example channel A was hit. The timestamp was 0x1f4b85 which is 2050949 ticks since data was cleared. Note, there are 32768 ticks per second which computes to a timestamp of 62.58999 seconds.
c - is the channel (A=0, B=1, C=2, D=3).
a - is the hit amplitude (when available).
tttttt - is the 6 byte timestamp
The responses to hits continue until **any** other command is issued to the PicoCount unit..

The 'sync' packet sent each minute looks like this:
Sync: **<'S'><0><0><0>.....**

Command: **<0><0>@s<1><2><118><0>**
0x00 0x00 0x40 0x73 0x01 0x02 0x76 0x00

Response: **<6><5><c><tttt><cs> ……..**
0x06 0x05 0x00 0x85 0x4b 0x00 0x00 0xd5 0x00 ---------->
In this example channel A was hit. The A channel total since the data was cleared is 0x00004b85 which is 19,333 hits. The responses to hits continue until **any** other command is issued to the PicoCount unit..

The 'sync' packet sent each minute looks like this:
Sync: **<'S'><0><0><0>.....**

Command: **<0><0>@s<1><3><119><0>**
0x00 0x00 0x40 0x73 0x01 0x03 0x77 0x00

Response: **a……..**
0x61 ----------
In this example channel A was hit. The responses to hits continue until **any** other command is issued to the PicoCount unit..

The 'sync' packet is the character 'S'
Sync: **S……..**

PicoCount Data Storage Protocol

The PicoCount generates a time-stamp of every hose hit. Data is only stored if there is a hose hit, therefore no memory is used if there is no activity. To conserve memory further and shorten download times, the time-stamp data is compressed. This section describes how the data is compressed.

The PicoCount runs a "tick" counter that is 6 bytes long with a tick resolution of 1/32768 seconds. This tick counter is zero'd when the data is zero'd (see **]z** command above).

To compress the data, only the tick bytes that have changed since the last stored tick count will be stored. Each stored tick count begins with an information byte which indicates how many tick bytes are stored and which channel the data is for.

Information byte structure:
Bits 0..3: (1=Channel A, 2=Channel B, 3=Channel C, 4=Channel D, 14=CountBuddy, 12=Start Study, 13=Stop Study, and other values are reserved for future use)
Bits 4..7: (9=1 byte, 10=2 bytes, 11=3 bytes, 12=4 bytes, 13=5 bytes, 14=6 bytes)

There will always be at least one byte of the tick counter stored. It is important to always begin reading the compressed data at the beginning of the data. Each time-stamp is built up on the last time-stamp. The first time-stamp is the tick-count since the unit was last zero'd, and will occur on the very first hose hit. All tick count bytes are sent with lowest byte first.

Examples:

`0xc2 0x34 0x6b 0xc3 0x04`	(Channel B with 4 lowest bytes of tick counter).
`0xa1 0x7f 0x73`	(Channel A with 2 lowest bytes of tick counter).
`0xb2 0x13 0xc4 0xc6`	(Channel B with 3 lowest bytes of tick counter).
`0xa1 0xa3 0xcc`	(Channel A with 2 lowest bytes of tick counter).

From the above sequence, let us assume that the first time-stamp is the very first time stamp of the data. The de-compressed data would be (lowest byte first)

Decompressed Tickcounts	Seconds since start
`0x34 0x6b 0xc3 0x04 0x00 0x00`	2438.837524
`0x7f 0x73 0xc3 0x04 0x00 0x00`	2438.902313
`0x13 0xc4 0xc6 0x04 0x00 0x00`	2445.531383
`0xa3 0xcc 0xc6 0x04 0x00 0x00`	2445.598724

This de-compressed data would then be used to compute the real time/stamps which are simply the start date/time plus the tickcount/32768.