# Linear Actuator Serial Control

### Move Relative to current Position

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|---|---|---|---|---|---|
| 0x50 | Data Long | ... | ... | ... | Checksum |

Bytes [2..5] are a 32 bit signed long indicating number of steps to move, with the sign indicating direction.
Byte 2 is the MSB

The Checksum is bytes [1..5] XORed together

### Move to Absolute Position

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|---|---|---|---|---|---|
| 0xB0 | Data Long | ... | ... | ... | Checksum |

The actuator internally tracks it's absolute position, and it can be set remotely (see next command).

Bytes [2..5] are a 32 bit signed long indicating desired position. The absolute position CAN be negative.
Byte 2 is the MSB

The Checksum is bytes [1..5] XORed together

### Set Absolute Position NOT WORKING

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|---|---|---|---|---|---|
| 0x3A | Data Long | ... | ... | ... | Checksum |

Bytes [2..5] are a 32 bit signed long which the current absolute position will be set to. Note: this Command will NOT cause the actuator to move. Setting absolute position to 0 is effectively “Home-ing” the device.
Byte 2 is the MSB

The Checksum is bytes [1..5] XORed together

### Get Status

| Byte 1 | Byte 2 |
|---|---|
| 0x3C | Checksum |

Actuator responds with ASCII data describing the actuators current position, Potentiometer value, and Encoder value, and whether the motor is currently in it's home position.

The Checksum is effectively 3C as well, since there is only one byte to XOR.

Example Strings:
AckB GSt Pos 32 Pot 9098 Enc 0 MtrHome  eol
AckB GSt Pos 63 Pot 9099 Enc 0 MtrNotHome  eol

**Get Internal Temperature**

| Byte 1 | Byte 2 |
|---|---|
| 0x3F | Checksum |

Actuator responds with ASCII data describing the six temperature sensors mounted internally in the actuator, as unconverted (Raw ADC Value) decimal Integers, space separated, terminated by the string “eol”.

The Checksum is effectively 3F as well, since there is only one byte to XOR.

**Get External Temperature**

| Byte 1 | Byte 2 |
|---|---|
| 0x30 | Checksum |

Actuator responds with ASCII data describing the six external temperature sensors (Mounting positions not determined at this time), as unconverted decimal numbers, space separated, terminated by the string “eol”.

The Checksum is effectively 30 as well, since there is only one byte to XOR.

**Temp Sensors (Both)**
In both cases, if the temperature sensor is not present, the actuator will return the value represented by 0xFF_FF_FF_FF when converted to decimal as a signed integer. This is -2^(32-1), or -2,147,483,648

**Control LEDs**

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|---|---|---|---|---|---|
| 0x75 | Data Long | ... | ... | ... | Checksum |

Bytes [2..5] are a 32 bit signed long which Dictate the LEDs behavior.

{{

IICCOnfig has a bundle of functions
Since we only have 4 outputs, IICConfig is broken into 8 4 bit sections.
Bits : 32 <--- 32 bit long ---> 1
Sections : 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0
Upper # sections Usurp lower sections.

Section 0: If all other sections are 0, this sets the output state. Each bit maps to an Output Pin
Section 1: A 1 here causes the corresponding output pin to flash at a fixed rate
Section 2: A 1 here causes the corresponding pin to flash once. Any value here is zeroed when the pin flashes
Section 3: A 1 here causes the LED to mimic the stepper motors Step LED
Section 4: A 1 here causes the LED to mimic the stepper motors Direction LED

}}

Byte 2 is the MSB

The Checksum is bytes [1..5] XORed together"

LED Behavior may change

# Control motor power state

**Motor Off**

| Byte 1 | Byte 2 | Byte 3 |
|---|---|---|
| 0x11 | Data Byte | Checksum |

Byte 2 controls the motors power state. A value of 0xFF turns the motor on, a value of 0x00 turns the motor off. The behavior when sent any other value should be regarded as indeterminate. However at this point, any value != 0xFF will turn the motor off .

Note – Since it is not possible to turn the motor off when it's position is in one of the microsteps, the actuator will refuse to turn the motor off unless the current position is a full step. This prevents the loss of position information due to the motor snapping to the nearest full-step upon power-off.

If the motor is on a full step, it will return a string “MtrOff eol” and power the motor off. otherwise, it will return “MtrHomeErr eol”.

The Checksum is bytes 1&2 XORed together

**Motor Really Off**

| Byte 1 | Byte 2 | Byte 3 |
|---|---|---|
| 0x15 | Data Byte | Checksum |

Byte 2 controls the motors power state. A value of 0xFF turns the motor on, a value of 0x00 turns the motor off. The behavior when sent any other value should be regarded as indeterminate. However at this point, any value != 0xFF will turn the motor off .

**Note -** Really turns the motor off. **Will** cause loss of position information.

The Checksum is bytes 1&2 XORed together

# Flash EEPROM

## Write

| Byte 1 | Byte 2 | Byte 3 | Byte 4 -Byte 32771 |
|---|---|---|---|
| 0xAA | 0x55 | 0xCC | EEPROM Data Blob |

Flashing the EEPROM is accomplished by sending a 3 byte Flash EEPROM command, followed by the 32 Kb (32768 byte) EEPROM data image.

Note that there is no checksumming or checking done on the eeprom data. The verification is accomplished by reading the EEPROM back and confirming that the contents of the EEPROM match the EEPROM data image. This must be done on the PC end of the control system.

The EEPROM data image is produced by the Propeller Tool.
Hit F8 → Click “Show Hex” → “Save EEPROM File”
The file produced has the suffix .EEPROM, and should be exactly 32768 bytes in size.

It is important to note that while in programming mode, the controller is **completely** unresponsive. **All** serial data received is written directly to the EEPROM, and the actuator sends no serial information. The controller will not resume normal operations until it has received all 32 kb. **It will block indefinitely if no data is sent!**
If the power is cycled or lost while programming, it will likely result in a unbootable state, which can only be rectified by reflashing through the dedicated programming interface.

When the controller has received all 32 kb, it sends the message “Done Programming eol”, and resumes normal functioning. However, the new firmware is not actually loaded until the controller is power-cycled or reset..

I'm going to implement a soft-reset function in the near future, to make it easy to reload the firmware into the controller

## Read

**Command**

| Byte 1 | Byte 2 | Byte 3 |
|---|---|---|
| 0x27 | 0x55 | 0xCC |

**Response**

| String (11 Bytes) | Data (32768 bytes) | String (13 Bytes) |
|---|---|---|
| “BeginEEPROM” | EEPROM Data | “EndEEPROM eol” |

Reading from the EEPROM is accomplished by sending a 3 byte command, to which the controller responds with the contends of the EEPROM, bookended by the strings “BeginEEPROM” and “EndEEPROM eol”.
The actuator dumps the whole EEPROM image as fast as it can (at 9600 baud, anyways).
Note: It may be possible to overrun the serial receive buffer on the computer (depending on driver settings, I believe), so you should be prepared to empty it continuously as the data is received.

At this time, it can be guaranteed that the begin and end strings do not appear anywhere in the actual EEPROM image.

Generally, one should write a .EEPROM file to the controller, and immediately read the image back to compare it against the original file. If the two match, the write was successful. Otherwise, the write cycle should be repeated.

## Reboot

Reboot the processor (used to load new firmware)

| Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
|---|---|---|---|---|---|
| 0x52 | 0x45 | 0x42 | 0x4F | 0x4F | 0x54 |

Immediately hard-resets the processor. **ALL** state information is lost. On boot, the processor loads the firmware from the EEPROM. Useful if you have uploaded new firmware.

Note, the command string in ASCII is “REBOOT” - in hex it is 52:45:42:4F:4F:54. Case sensitive.