Pololu TReX Jr Firmware Version 1.0: Command Documentation

Quick Command List:

Data-Query Commands:

- 0x80: Expanded Protocol Packet Start Byte
- 0x81: Get Signature
- 0x82: Get Mode
- 0x83: Does Serial Control Motors?
- 0x84: Get Status Byte
- 0x85: Get UART Error Byte
- 0x86: Get Raw Channel Values
- 0x87: Get Remapped Channel Values
- 0x8D: Get Motor 1 Current
- 0x8E: Get Motor 2 Current
- 0x8F: Get Motor Currents
- 0x90 – 0x94: Get Channel Calibration Values
- 0x95: Get Calibrated Channels
- 0x9F: Get Configuration Parameter

Set-Parameter Commands:

- 0xA0 – 0xA4: Set Channel Calibration Values
- 0xAF: Set Configuration Parameter

Motor Commands:

- 0xC0 – 0xC3: Set Motor 1
- 0xC4 – 0xC7: Accelerate Motor 1
- 0xC8 – 0xCB: Set Motor 2
- 0xCC – 0xCF: Accelerate Motor 2
- 0xD0 – 0xDF: Set Motors 1 & 2
- 0xE0 – 0xEF: Accelerate Motors 1 & 2
- 0xF0: Set Auxiliary Motor
- 0xF1: Accelerate Auxiliary Motor

General Overview:

- Command bytes must have their most significant bits set (i.e. 128 – 255) while data bytes must have their most significant bits cleared 0 (i.e. 0 - 127).
- Data-Query commands (0x81 – 0x9F) are used to obtain data and can be issued in any TReX Jr mode. These commands all return data.
- Set-Parameter commands (0xA0 – 0xAF) are used set parameters and can only be issued in serial mode. These commands all return data.
- Motor commands (0xC0 – 0xF1) are used to control the motors; they can be issued in any TReX Jr mode, but the motors will only respond in serial mode or when serial override is active. If serial override is enabled, the motors will be set according to the last motor settings received via serial commands (even if those commands came in while serial override was disabled). These commands do not return data.
- If cyclic redundancy check is enabled, an additional CRC-7 byte must be tacked onto the end of every command packet and an additional CRC-7 byte will be tacked onto the end of all data packets transmitted by the TReX Jr in response to a command. The commands detailed in this document assume that cyclic redundancy check is disabled and make no mention of these additional CRC-7 bytes. Please see the Cyclic Redundancy Check section for more details.

Commands in Detail:

0x80: expanded protocol packet start byte

This command exists to make the TReX Jr compatible with the serial protocols used by other Pololu products, which in turn means it can be chained along with such products to the same serial line. The expected command packet is:

0x80, device #, command byte with MSB cleared, any necessary data bytes

The default device # for the TReX Jr is 0x07; this is a configuration parameter that can be changed. As an example, here are the two command packets you can send to read the raw values of all five channels:

1. Expanded protocol: 0x80, 0x07, 0x06, 0x1F
2. Compact protocol: 0x86, 0x1F

0x81: get signature

This command takes no data bytes and returns nine bytes. The first six of these bytes will always be 'T', 'R', 'e', 'X', 'J', 'r'. The last three are the firmware version (major byte, '.', minor byte).

Example signature: “TReXJr1.0”

0x82: get mode

This command takes no data bytes and returns one byte that indicates the mode as set by the mode selection shorting block:

'R' (0x52) = RC mode
'A' (0x41) = Analog mode
'r' (0x72) = Serial mode with channels configured for RC input signals
'a' (0x61) = Serial mode with channels configured for analog input signals

When operating in serial mode, serial is always in control, however it can choose to read and use the channel inputs as it sees fit. The channel input source parameter (0x7C) determines whether the channel inputs are configured for RC or analog signals when the TReX Jr is in serial mode. This parameter has no effect when the mode shorting block selects for RC or analog mode.

0x83: serial controls motors?

This command takes no data bytes and returns one byte that indicates whether serial is in control of the motors. Serial controls the motors when the TReX Jr is in serial mode or when serial override (as controlled by channel 5) is active.

Return value: 0 = serial is not in control, 1 = serial is in control

0x84: get status byte

This command takes no data bytes and returns one byte that contains status information.

Status byte:
bit 7: Motor 2 current over current limit
bit 6: Motor 2 fault
bit 5: Motor 1 current over current limit
bit 4: Motor1 fault
bits 3-1 not used
bit 0: UART error (read UART error byte for more information)

The bits of the status byte are latched. This means that once set, they will stay set until the status byte is read. Reading the status byte clears all the status bits.

0x85: get UART error byte

This command takes no data bytes and returns one byte that contains UART error information.

UART error byte:
bit 7: timeout
bit 6: command packet format error

bit 5: CRC error
bit 4: frame error (generated by UART hardware)
bit 3: data overrun (generated by UART hardware)
bit 2: parity error (generated by UART hardware)
bit 1: read buffer overrun
bit 0: send buffer overrun

- Read and send buffer overruns should never happen if you follow the proper protocol when communicating with the TReX Jr. Specifically, so long as you don't transmit to the TReX Jr until you have finished receiving all the data you have requested from the TReX Jr with the previous command, you should be fine.

- Frame, data, and parity errors are generated by the UART hardware itself. If any occur, they are passed on to the user via the UART error byte. A frame error occurs if the UART fails to see a stop bit when it's expecting one. A parity error occurs only when you have configured the UART to use either even or odd parity and the received character doesn't match that parity. A data overrun error occurs when a byte is received while the UART's internal receive buffer is full. A data overrun error should never happen.

- A CRC error occurs when you have enabled cyclic-redundancy checking and the CRC byte at the end of the command packed does not match the CRC the TReX Jr expects for that packet. In such a case, the entire command packet is discarded as untrustworthy.

- A command packet format error occurs when the received command packet does not conform to certain expectations the TReX Jr has about the contents of the command packet. If a format error occurs, the command packet will be discarded as untrustworthy. Such an error will be generated by any of the following:

    1. a non-existent command is issued
    2. a new command byte is received in the middle of a command packet
    3. a data byte with an illogical/bad value is sent

- The timeout bit is set if too much time goes by between TReX Jr serial receptions and the motors are shut down as a result. This exists as a safety feature that will disable all the motors should the serial controller go dead. The timeout feature is only active when the serial timeout parameter is non-zero.

The bits of the UART error byte are latched. This means that once set, they will stay set until the UART error byte is read. Reading the UART error byte clears all the UART error bits.

0x86: get raw channel input values

This command takes one data byte and will return 0 – 10 bytes, depending on the data byte. If bit n of the data byte is set, the two-byte input value for channel n+1 will be returned. A data byte of 0x1F will request values for all five channels. For each two-byte channel value, the low byte is transmitted first. The values of lower channels are transmitted before those of higher

channels.

When the channel inputs are being treated as RC pulses, a channel value is the width of the last RC pulse received in units of .4us. For example, if the last RC pulse on channel 1 was 1.5ms wide, sending a data byte of 0x01 will return a value of 1500us/.4us = 3750. If the last pulse received was an error (i.e. it was too long or too short), or if no pulse has been received on the channel in the past 100ms, the value returned for that channel will be 0xFFFF. Before using an RC channel value, you must check to see that it's not 0xFFFF. The update rate of the RC channel values depends on the pulse-train frequency of your receiver; the standard rate for receivers is 50 Hz.

When the channel inputs are being treated as analog voltages, the channel value is the result of the last 10-bit analog-to-digital conversion performed on that channel. The value will hence range from 0 to 1023. Each analog channel value is the average of 16 consecutive conversion samples. The update rate for the analog values is approximately 50 Hz.

RC input Example
transmission: 0x86, 0x13
reception: 0x35, 0x0E, 0xFF, 0xFF, 0x9B, 0x0A
interpretation:

- channel 1 input value = 0x0E35 = 3637 (.4us) = 1.455ms
- channel 2 input value = 0xFFFF = last pulse was invalid
- channel 5 input value = 0x0A9B = 2716 (.4us) = 1.086ms

0x87: get remapped channel input values

This command takes one data byte and will return 0 – 5 bytes depending on the data byte. If bit n of the data byte is set, the single-byte remapped input value for channel n+1 will be returned. A data byte of 0x1F will request values for all five channels. Values of lower channels are transmitted before those of higher channels.

Internally the raw channel values are remapped into unitless, single-byte values. When using remapped values, one doesn't have to care whether the input source was analog or RC. The remapping is performed by scaling the raw channel based on channel calibration settings (max value, neutral value, min value, and deadband range) that can be user-specified or automatically learned. If the raw value is greater than or equal to the channel maximum, the channel's scaled value is 127. If the raw value is within deadband around neutral, it's scaled value is 0.

Bits 6:0 of the remapped value indicate its magnitude (0 – 127) and bit 7 indicates its direction (you can think of bit 7 = 0 as + and bit 7 = 1 as -). The computation of the remapped values is affected by the following factors:

- channel calibration (max, neutral, min, deadband)
- flipped channels parameter
- parabolic channels parameter
- channels 1 & 2 are also affected by the mix shorting block and by the flip channel (4)

In short, **the remapped value for channels 1 is the speed/direction the TReX Jr would be trying to set motor 1 to were it in control, and the remapped value for channel 2 is the speed/direction the TReX Jr would be trying to set motor 2 to were it in control**. The lower six bits of the remapped value for channel 3 are the speed to which the TReX Jr would be trying to set motor 3. The remapped channel 1 value will always correspond directly to motor 1, regardless of the state of the mix shorting block or the flip channel. This is in contrast to the raw value for channel 1, which would apply to motor 2 if the flip channel indicates the robot is inverted.

This means you could easily write a serial program that mimics RC/analog mode by requesting the remapped channel values and setting motor 1 based on the channel 1 value, motor 2 based on the channel 2 value, and the auxiliary motor (motor 3) based on the channel 3 value.

0x8D: get motor 1 current
0x8E: get motor 2 current

This command takes no data bytes and returns one byte indicating the motor current. Motor current is only available when running in independent motor mode (as opposed to joint motor mode, where you are using both H-bridges in unison to drive a single motor). The motor current is an 8-bit 64-sample ADC average that updates 50 times per second if the TreX Jr is treating the channel inputs as analog. The update rate is approximately 80 Hz if the TReX Jr is treating the channel inputs as RC servo pulses. The returned value will range from 0 to 255, and you can multiply this value by 40 mA to get the approximate current drawn by the motor. For example, if this command returns a value of 55, the associated motor is drawing roughly 2.2 A.

0x8F: get motor currents

This command takes no data bytes and returns two bytes. The first byte is motor 1 current and the second byte is motor 2 current.

0x90 – 0x94: get channel calibration values

These commands take no data bytes and return 8 bytes that provide the four two-byte calibration values for the channel specified by the command. Command 0x90 refers to channel 1, 0x91 to channel 2, ..., and 0x94 to channel 5. The calibration values returned are, in order, minimum, neutral, maximum, and deadband. The low byte of each value is transmitted first. RC calibration values are transmitted if the TReX Jr is in RC mode, or if it's in serial mode and the channel input source parameter is set for RC mode. Analog calibration values are transmitted if the TReX Jr is in analog mode, or if it's in serial mode and the channel input source parameter is set for analog mode.

0x95: get calibrated channels

This command takes no data bytes and returns one byte whose bits indicate which channels

have calibration values that differ from the defaults. Such a channel is considered to have been "learned". If bit n is set, channel n+1 has been learned, either through the automatic learning process or via commands 0xA0 – 0xA4 (set channel calibration).

0x9F: get configuration parameter

This command takes one data byte representing the desired parameter and returns one byte representing the value of that parameter. **Please see the Configuration Parameter documentation for detailed information on the TReX Jr's configuration parameters.**

Note: The TReX Jr must be in serial mode for commands 0xA0 – 0xAF to work.

0xA0 – 0xA4: set channel calibration (requires serial mode)

These commands take 10 data bytes and return one byte. They set the calibration values for the channel specified by the command (command 0xA0 refers to channel 1, 0xA1 refers to channel 2, ..., and 0xA4 refers to channel 5). The 10 data byte are, in order:

1. minimum low byte
2. minimum high byte
3. neutral low byte
4. neutral high byte
5. maximum low byte
6. maximum high byte
7. deadband low byte
8. deadband high byte
9. 0x55 (format byte 1)
10. 0x2A (format byte 2)

The following restrictions apply to the values:

minimum ≤ neutral AND neutral ≤ maximum AND minimum < maximum

These data are treated as RC calibration values if the TReX Jr is in RC mode, or if it's in serial mode and the channel input source parameter is set for RC mode. They are treated as analog calibration values if the TReX Jr is in analog mode, or if it's in serial mode and the channel input source parameter is set for analog mode.

The return byte is not sent until the calibration parameters have been set, which will take approximately 32ms if the command packet format is correct. Receiving the return byte is an indication that the TReX Jr is now ready for the next command. The value of the returned byte lets you know if the calibration parameters were set or if there was a problem with the format. The possible return byte values are:

0 = command OK

2 = bad value (e.g. maximum = minimum)
3 = TReX Jr isn't in serial mode

If the return value is not zero, the TReX Jr has rejected the command packet and the calibration values were not set. If you do not receive a return value within 100ms, there was a problem with the command packet that made it untrustworthy (e.g. the format bytes were incorrect). The UART error byte might contain useful information if this happens.

0xAF: set configuration parameter (requires serial mode)

This command takes four data bytes and returns one byte. The data bytes are, in order

1. parameter #
2. parameter value (7-bit representation)
3. 0x55 (format byte 1)
4. 0x2A (format byte 2)

The return byte is not sent until the parameter has been set, which will take approximately 4ms if the command packet format is correct. Receiving the return byte is an indication that the TReX Jr is now ready for the next command. The value of the returned byte lets you know if the parameter was set or if there was a problem with the format. The possible return byte values are:

0 = command OK
1 = bad parameter #
2 = bad value for the specified parameter
3 = TReX Jr isn't in serial mode

If the return value is not zero, the TReX Jr has rejected the command packet and the calibration values were not set. If you do not receive a return value within 50ms, there was a problem with the command packet that made it untrustworthy (e.g. the format bytes were incorrect). The UART error byte might contain useful information if this happens.

Once a parameters is set, it is saved, so it will never need to be set again (unless you wish to change it later). Setting parameters 0 – 0x16 will have an immediate effect on the TReX Jr; setting parameters 0x7B – 0x7F requires a device reset (i.e. cycle the power) to make the changes active.

Please see the Configuration Parameter documentation for detailed information on the TReX Jr's configuration parameters.

Note: The TReX Jr must be in serial mode or serial override for these commands to affect the motors. If serial override is not active, the TReX Jr will make note of the commands and store the directives for potential later use. When serial override is enabled, the motors will be set according to the most recently received serial motor commands.

0xC0 – 0xC3: set motor 1
0xC8 – 0xCB: set motor 2

This command takes one data byte and returns nothing. It immediately sets the speed of the specified motor equal to the data byte and the motor direction based on the two least significant bits of the command byte. The direction bits work as follows:

00 = brake low (command 0xC0/0xC8)
01 = reverse (command 0xC1/0xC9)
10 = forward (command 0xC2/0xCA)
11 = brake low (command 0xC3/0xCB)

0xC4 – 0xC7: accelerate motor 1
0xCC – 0xCF: accelerate motor 2

This command takes one data byte and returns nothing. It sets the target speed of of the specified motor equal to the data byte and the target motor direction based on the two least significant bits of the command byte. The direction bits work as follows:

00 = brake low (command 0xC4/0xCC)
01 = reverse (command 0xC5/0xCD)
10 = forward (command 0xC6/0xCE)
11 = brake low (command 0xC7/0xCF)

Motor speed is updated 100 times per second. If it is below its target speed and if its current direction is the same as its target direction, each update will adjust the speed by increasing it by a tenth of its acceleration parameter. If it is above its target speed and its direction is in the target direction, the update will merely set its speed equal to the target value (i.e. there is no deceleration). Every 10ms, the following acceleration logic is performed:

if motor speed < target speed, motor speed = motor speed + acceleration/10
if motor speed > target speed, motor speed = target speed

If the target direction differs from the current direction, the first update will brake the motor at 100% duty cycle for the amount of time specified by the its brake duration parameter. It will then set the motor speed to zero and accelerate from there to the target speed in the target direction.

If you intend to use the current-limiting feature of the TReX Jr, you should use acceleration commands to control your motor speed. This is because acceleration commands schedule motor updates to happen as part of a fixed update cycle that also takes care of the current-limiting logic. “Set motor” commands set the motor speed the instant they're received and will, at least temporarily, override any current-limiting actions the TReX Jr is taking. If you want to limit your current but you don't want acceleration, set the acceleration parameter to zero; this essentially requests infinite acceleration.

Note: acceleration does not apply to braking or to a speed decrease that does not also result in a change of direction. Motor speed can also be influenced by current-limit settings, which add

additional considerations to the logic detailed in this section. Please see the current limit section of the parameter documentation for more details.

0xD0 – 0xDF: set motors 1 and 2

This command takes two data bytes and returns nothing. It immediately sets the speed of motor 1 equal to the first data byte and the speed of motor 2 equal to the second data byte. Motor 1 direction is specified by bits 1:0 of the command byte and motor 2 direction is specified by bits 3:2 of the command byte. If you want to set both motors at once, using this command lets you do it with just three bytes. If you have a two-bit m1 direction value (see the direction values listed in the "set motor 1/set motor 2" commands above) and a two-bit m2 direction value, your command packet would look like:

command byte = 0xD0 | (m2 direction * 4) | (m1 direction)
data byte 1 = m1 speed
data byte 2 = m2 speed

0xE0 – 0xEF: accelerate motors 1 and 2

This command takes two data bytes and returns nothing. It sets the target speed of motor 1 equal to the first data byte and the target speed of motor 2 equal to the second data byte. Motor 1 target direction is specified by bits 1:0 of the command byte and motor 2 target direction is specified by bits 3:2 of the command byte. If you want to accelerate both motors at once, using this command lets you do it with just three bytes. If you have a two-bit m1 direction value (see the direction values listed in the "set motor 1/set motor 2" commands above) and a two-bit m2 direction value, your command packet would look like:

command byte = 0xD0 | (m2 target direction * 4) | (m1 target direction)
data byte 1 = m1 target speed
data byte 2 = m2 target speed

See the "accelerate motor 1/accelerate motor 2" commands (0xC4-0xC7/0xCC-0xCF) above for more information about how motor acceleration works.

0xF0: set auxiliary motor (M3)

This command takes one data byte and returns nothing. It immediately sets the speed of of the auxiliary motor equal to the data byte.

0xF1: accelerate auxiliary motor (M3)

This command takes one data byte and returns nothing. It sets the target speed of the auxiliary motor equal to the data byte. See the "accelerate motor 1/accelerate motor 2" commands (0xC4-0xC7/0xCC-0xCF) above for more information about how motor acceleration works.