Pololu TReX Jr Firmware Version 1.0: Configuration Parameter Documentation

Quick Parameter List:

- 0x00: Device Number
- 0x01: Required Channels
- 0x02: Ignored Channels
- 0x03: Reversed Channels
- 0x04: Parabolic Channels
- 0x05: Motor 1 Deadband Brake PWM
- 0x06: Motor 2 Deadband Brake PWM
- 0x07: Serial Timeout
- 0x08: UART-Error Shutdown
- 0x09: Motor 1 PWM Prescaler
- 0x0A: Motor 2 PWM Prescaler
- 0x0B: Motor 1 PWM Maximum
- 0x0C: Motor 2 PWM Maximum
- 0x0D: Auxiliary Motor PWM Maximum
- 0x0E: Motor 1 Acceleration
- 0x0F: Motor 2 Acceleration
- 0x10: Auxiliary Motor Acceleration
- 0x11: Motor 1 Brake Duration
- 0x12: Motor 2 Brake Duration
- 0x13: Motor 1 Current Limit
- 0x14: Motor 2 Current Limit
- 0x15: Motor 1 Current Limit Proportionality Constant P
- 0x16: Motor 2 Current Limit Proportionality Constant P

- 0x7B: Motor Mode
- 0x7C: Channel Input Source
- 0x7D: CRC-7 Polynomial
- 0x7E: UART Settings
- 0x7F: Reset All Parameters to Factory Defaults

General Overview:

- Once a parameter is set, its new value is saved until it is set again. Saving a parameter takes approximately 4 ms. There is no delay involved in reading a parameter.
- Setting parameters 0 – 0x16 will affect the TReX Jr on the fly; setting parameters 0x7B – 0x7F requires a device reset to make the changes active.
- Parameters 0x01 – 0x06 only affect how the TReX Jr behaves when it is RC or Analog mode with serial override off. They influence how the TReX Jr converts channel inputs into motor control. Serial mode/serial override bypasses them by directly controlling the motors.
- The Set Configuration Parameter command (0xAF) can only accept 7-bit parameter value data bytes, which means that all of the TReX Jr parameters are restricted to 7-bit values. Because of this, there are a few parameters that are 7-bit representations of 8-bit values. This applies to parameters 0x05, 0x06, 0x0B, 0x0C, 0x0D, 0x13, 0x14, and 0x7D.

Parameters in Detail:

0x00: device number (default value: 0x07)

The device number can be used in conjunction with the extended serial command protocol to communicate with one of many devices connected to the same serial line. The command protocol would be:

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

When the TReX Jr receives the command byte 0x80, it then compares the subsequent device # data byte to its own device number. If the two match, it processes the command, otherwise it ignores it.

0x01: required channels (default value: 0x01 – only channel 1 is required for operation)

This is a safety parameter that only has meaning when the TReX Jr is in RC mode. If bit n of the required channels byte is set, then channel n+1 is considered "required for operation." If the TReX Jr is operating in RC mode, it will only execute when it detects a valid pulse train on all required channels. If the pulse train on a required channel stops, the TReX Jr will go in to a "wait for safe start" mode, shutting down all the motors until the pulse train is established again.

When setting this parameter, bits 5 and 6 must be zero (i.e. required channels $\leq$ 0x1F).

0x02: ignored channels (default value: 0x00 – no channels are ignored)

This is a safety parameter that directs the TReX Jr to ignore certain channels. If bit n of ignored channels is set, then channel n+1 will be ignored. If you know you're not going to use a

channel, ignoring it can help protect you from any unwanted signals/noise that it might detect. This can be especially useful if you're running in analog mode, since the TReX Jr cannot tell that the ADC result from a disconnected analog channel is bogus. A disconnected analog channel, if not set as ignored, could also keep the TReX Jr from being able to get past the initial "safe-start" check.

When setting this parameter, bits 5 and 6 must be zero (i.e. ignored channels ≤ 0x1F).

0x03: reversed channels (default value: 0x00 – no channels are reversed)

By setting bit n of this parameter, you can reverse channel n+1. If by default your control stick moves motor 1 forward when you push it down and this is not the behavior you want, you can reverse channel 1 by setting reversed channels = 0x01, and now motor 1 will move forward when you push the control stick up.

When setting this parameter, bits 5 and 6 must be zero (i.e. reversed channels ≤ 0x1F).

0x04: parabolic channels (default value: 0x00 – no parabolic channels)

By setting bit n of this parameter, you can scale channel n+1 parabolically rather than linearly. Normally, when you move the control stick through an angle *a*, motor speed will increase by an amount proportional to *a*. With parabolic scaling, speed will increase by an an amount proportional to a^2 (still scaled so that when *a* is at its maximum value, the resulting motor speed is 127. The result is that you get better (less sensitive) slow-speed control in the region near neutral and worse (more sensitive) high-speed control at the extremes.

When setting this parameter, bits 5 and 6 must be zero (i.e. parabolic channels ≤ 0x1F).

0x05: motor 1 deadband brake PWM (default: 0 – coast while in the deadband)
0x06: motor 2 deadband brake PWM (default: 0 – coast while in the deadband)

When a channel is within its deadband (i.e. within a certain distance of neutral), the motor controlled by that channel will either coast or variable brake, depending on the setting of this parameter.

Note: these are seven-bit parameters that are used to represent eight-bit values as follows:

- if parameter value = 0,
  desired brake PMW = 0 (coast in deadband)
- else
  parameter value = (desired brake PWM – 1) / 2
  so, *desired brake PWM = (parameter value * 2) + 1*

For example, if you want a motor to brake as hard as possible in the deadband, which would mean a desired brake PWM of 0xFF, you would give this parameter a value of 0x7F. If you

want a small amount of braking, such as 0x21, you would give this parameter a value of 0x10. If you just want to coast while in the deadband, you would give this parameter a value of zero.

0x07: serial timeout in .1 s (default: 0 – no timeout)

This safety parameter causes the TReX Jr to shut down its motors if serial is in control (i.e. it's in serial mode or serial override is active) and it goes for too long without receiving any serial commands. This parameter allows the TReX Jr to safely handle situations in which the serial controller gets disconnected or fails in some way. The serial timeout is specified in tenths of a second, allowing for a maximum timeout of ~13 s. A timeout of zero disables the timeout feature.

0x08: UART-Error Shutdown (default: 1 – shutdown motors on UART error)

This safety parameter causes the TReX Jr to shut down its motors if serial is in control (i.e. it's in serial mode or serial override is active) and a serial error occurs. A serial error counts as anything that would result in the setting of a UART Error Byte bit. The rationale behind this parameter is that the serial error might have occurred during the transmission of a command intended to stop the motors. Corruption of such a command could be dangerous, so the TReX Jr takes the default position of assuming the worst. This safety feature can be disabled by setting the UART-Error shutdown parameter to zero.

0x09: motor 1 PWM prescaler (default: 1 – PWM frequency = 312.5 kHz / (PWM max + 1))
0x0A: motor 2 PWM prescaler (default: 1 – PWM frequency = 312.5 kHz / (PWM max + 1))

These parameters affect the frequencies of the motor 1 and motor 2 PMWs. PWM frequency is given by:

frequency = 20 MHz / prescaler / (PWM maximum + 1)

The PWM maximum for each motor is a parameter (0x0B and 0x0C) that has a default value of 0x7F. Below is a list of the prescalers available and the frequencies they would produce if the PWM maximum is at its default value:

PWM prescaler parameter
0 = prescaler of **8** (which gives a frequency of **19.5 kHz** for PWM maximum = 0x7F)
1 = prescaler of **64** (which gives a frequency of **2.44 kHz** for PWM maximum = 0x7F)
2 = prescaler of **256** (which gives a frequency of **610 Hz** for PWM maximum = 0x7F)
3 = prescaler of **1024** (which gives a frequency of **153 Hz** for PWM maximum = 0x7F)

Of the four frequencies listed above, only 19.5 kHz is outside the typical range of human hearing, which means that motors running at that frequency won't produce an irritating whine. Unfortunately, the MC33887 motor drivers used by the TreX Jr only support operation up to 10 kHz, so you should avoid using a prescaler and PWM maximum combination that results in a frequency above 10 kHz. The drawback to higher frequencies is greater power loss due to switching. On the low end of the frequency spectrum, you won't have much power loss due to

switching, but your motor motion might be choppier (as an extreme example, imagine trying to drive your motor at 50% speed by alternating between turning it on full for 10 seconds and off for 10 seconds).

Note: the Set Configuration Parameter command (0xAF) will reject a PWM prescaler parameter that is greater than 3.

0x0B: motor 1 PWM maximum (default: 0x7F – max motor speed setting = max motor speed)
0x0C: motor 2 PWM maximum (default: 0x7F – max motor speed setting = max motor speed)

Motor speed setting magnitude will always range from 0 – 0x7F. The duty cycle of the motor PWM is calculated as:

duty cycle = motor speed setting / PWM maximum

The duty cycle, coupled with motor voltage, ultimately determines the speed of the motor. If you find that your motor is spinning too fast when set to a speed of 0x7F, you can increase PWM maximum beyond 0x7F to slow the motor down. Setting the PWM maximum to 0xFF will effectively halve the motor's maximum speed. Conversely, decreasing the PWM maximum below 0x7F will cause the motor to reach its top speed faster (when the motor speed setting = PWM maximum). These parameters can be a good way to compensate for motor voltages that would cause your motors to spin faster than is desirable for your particular application.

These parameters also provide a great way to compensate for asymmetries between motors 1 and 2. For example, perhaps you have a radio-controlled differential-drive robot that curves left when you try to make it go straight because the left motor turns more slowly than the right when given the same speed setting. You can correct this by decreasing the PWM maximum for your left motor or by increasing the PWM maximum for your right motor (or by employing a combination of the two).

Changing the PWM maximum will change the frequency of your motor PWM as follows:

frequency = 20MHz / prescaler / (PWM maximum + 1)

See the prescaler parameter section (0x09 and 0x0A) above for more information on this. Be aware that decreasing the PWM maximum much below 0x7F while using a prescaler of 0 might cause problems for the TReX Jr's motor drivers.

Note: these are seven-bit parameters that are used to represent eight-bit values as follows:

parameter value = (desired PWM maximum – 1) / 2
so, *desired PWM maximum = (parameter value * 2) + 1*

For example, if you want to set your PWM maximum to 0xFF, give this parameter a value of 0x7F.

0x0D: motor 3 PWM maximum (default: 0x7F – max motor speed setting = max motor speed)

Motor speed setting magnitude will always range from 0 – 0x7F. The duty cycle of the motor PWM is calculated as:

duty cycle = motor speed setting / PWM maximum

The duty cycle, coupled with motor voltage, ultimately determines the speed of the motor. If you find that your motor is spinning too fast when set to a speed of 0x7F, you can increase PWM maximum beyond 0x7F to slow the motor down. Setting the PWM maximum to 0xFF will effectively halve the motor's maximum speed. Conversely, decreasing the PWM maximum below 0x7F will cause the motor to reach its top speed faster (when the motor speed setting = PWM maximum). This parameter can be a good way to compensate for motor voltages that would cause your motors to spin faster than is desirable for your particular application.

This parameter does not affect the frequency of the auxiliary motor's PWM, which is always fixed at 38 Hz.

Note: this is a seven-bit parameter that is used to represent an eight-bit value as follows:

parameter value = (desired PWM maximum – 1) / 2
so, *desired PWM maximum = (parameter value * 2) + 1*

For example, if you want to set your PWM maximum to 0xFF, give this parameter a value of 0x7F.

0x0E: motor 1 acceleration (default: 0x50 – motor 1 speed increases by 0x50 every 100ms)
0x0F: motor 2 acceleration (default: 0x50 – motor 2 speed increases by 0x50 every 100ms)
0x01: motor 3 acceleration (default: 0x50 – motor 3 speed increases by 0x50 every 100ms)
0x11: motor 1 brake duration in .01 s (default: 0 – no braking on acceleration direction change)
0x12: motor 2 brake duration in .01 s (default: 0 – no braking on acceleration direction change)

Acceleration provides a great way to smooth out your motor control and reduce current spikes caused by sharp increases in motor speed or changes in motor direction. When you issue an acceleration motor command, you tell the TReX Jr your target speed and direction. The TReX Jr then ramps up the motor speed towards your target at a rate specified by that motor's **acceleration** parameter.

The TReX Jr updates the motor speed 100 times per second. If a motor's speed 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 10 ms, the following acceleration logic is performed:

- if motor speed < target speed,
  new motor speed = *min*(motor speed + **acceleration**/10, target motor speed)
- if motor speed > target speed,
  new 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, which is in increments of 10 ms. For example, if the brake duration parameter equals 0x7F (127), the motor will brake for 1.27 s when an acceleration command requests a direction change. The TReX Jr will then set the motor speed to zero and accelerate from there to the target speed in the target direction. If the brake duration is set to zero, there will be no braking on an acceleration direction change.

Special case: setting motor acceleration equal to zero essentially requests infinite acceleration. The next update will simply set the motor speed equal to the target speed if there is no direction change. If there is a direction change, the motor will brake for the amount of time specified by its brake duration parameter, and then its speed will be set directly to the target speed.

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 below for more details. Also note that when the TReX Jr controls the motors in response to RC or Analog input, it issues acceleration commands, so these parameters will influence how the TReX Jr behaves even when the motors are not being controlled by the serial interface.

0x13: motor 1 current limit (default: 0 – no motor 1 current limit)
0x14: motor 2 current limit (default: 0 – no motor 2 current limit)
0x15: motor 1 current limit proportionality constant P (default: 0x0A)
0x16: motor 2 current limit proportionality constant P (default: 0x0A)

The TReX Jr lets you to limit the amount of current motors 1 and 2 are allowed to draw. Every time the motor speed is updated, which happens 100 times per second, the current being drawn by the motors is compared to the **current limit** parameters. The following current-limiting logic is then performed every 10 ms:

- if **current limit** = 0, there is no current limit;
  take no actions based on motor current; use acceleration logic only
- if **proportionality constant P** = 0,
  turn off the motor if its current is over the limit or if a motor fault occurs
- else
  motor speed += *min*(acceleration, **P** * (**current limit** – current)) / 10
  if motor speed > target speed, motor speed = target speed

If the current limiting feature is enabled (i.e. the current limit parameter is not zero), current limiting can affect acceleration. The proportionality constant P determines how the motors will react when the current is in the vicinity of the limit. If the current is just under the limit and P is small, the motor speed may not be allowed to increase as much as would be dictated by the acceleration parameter alone. If the current is over the current limit, the quantity:

*P * (current limit – current)*

becomes negative and the effect will be a reduction in motor speed. The motor speed will continue to drop at a rate proportional to the difference between the current and the limit until

the current equals the limit. You will most likely need to empirically determine the parameter value for your desired current limit and the best constant of proportionality P for your particular application.

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 the 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: the current limits are seven-bit parameter that are used to represent eight-bit values as follows:

current limit parameter value = desired current limit / 2
*desired current limit = current limit parameter value * 2*

For example, if you want to set your current limit to 0xFE, give the current limit parameter a value of 0x7F. This does not apply to the proportionality constant P.

Note: setting the following parameters requires a device reset to make them active.

0x7B: motor mode (default: 0 – motors 1 and 2 are two independent motors)

The motor 1 and motor 2 outputs of the TReX Jr can either work independently to control two motors or work in unison to control a single, more powerful motor. To control a single motor using both motor outputs, this parameter must be set for joint-motor mode.

motor mode
0 = independent-motor mode (control two motors independently)
1 = joint-motor mode (both outputs are synchronized to control a single motor)

When operating in joint-motor mode, the single motor is considered to be "motor 1". This means that it is affected by all of the motor 1 parameters and motor 1 commands. "Motor 2" has no meaning when running in joint-motor mode. Similarly, the concept of "channel mixing" has no meaning in joint-motor mode and channel 2 will have no effect on the motor.

To use the TReX Jr in joint-motor mode, one of your motor's leads should be connected to both of the motor 1 outputs (M1 A and M1 B). The other of your motor's leads should be connected to both of the motor 2 outputs (M2 A and M2 B). See the Motor & Power Connections section of the online User's Guide.

Note: the Set Configuration Parameter command (0xAF) will reject a motor mode parameter that is not equal to either 0 or 1. After setting this parameter, you must power-cycle the TReX Jr to activate the change.

0x7C: channel input source (default: 'R' (0x52) – expect RC inputs on the five channels)

This parameter tells the TReX Jr what type of input to expect on its five channels when it's running in serial mode. When the TReX Jr not running in serial mode, the channel input source is determined by the location of the blue mode shorting block and this parameter has no effect at all. The two possible values for this parameter are:

parameter value
'A' (0x41) = Analog mode
'R' (0x52) = RC mode

Note: the Set Configuration Parameter command (0xAF) will reject a channel input source parameter that is not 'A' or 'R'. After setting this parameter, you must power-cycle the TReX Jr to activate the change.

0x7D: CRC-7 polynomial (default: 0x09 – standard CRC-7 polynomial)

This parameter specifies the polynomial used to compute the 7-bit CRC when the TReX Jr has CRC error-checking enabled. The default value is the standard CRC-7 polynomial used by telecom systems and multimedia cards. Some CRC polynomials will prove stronger than others when it comes to maximizing error-detecting capability and minimizing collision probabilities. As such, we recommend you only change this value if you need to do so to be compatible with an existing external system (or if you have reason to believe that your new CRC-7 polynomial will work better than 0x89). Please see the CRC section of the TReX Jr documentation for more information on cyclic redundancy checking.

Note: a 7-bit CRC polynomial is an 8-bit value that always has an MSB of 1. As such, this parameter contains only the **lower 7 bits** of the CRC-7 polynomial. For example, to use a CRC-7 polynomial of 0x89, you would set this parameter to 0x09. After setting this parameter, you must power-cycle the TReX Jr to activate the change. When the TReX Jr restarts, you should notice a one-time delay of approximately 1 second as the CRC table is regenerated.

0x7E: UART settings (default: 0x05 – no parity or CRC, one stop bit, 19.2k baud)

This individual bits of this parameter determine the TReX Jr's serial configuration as follows:

bits 6:5 = error checking mode
0 0 = no error checking
0 1 = 7-bit cyclic redundancy check (CRC-7) enabled
1 0 = even parity (implemented by the UART hardware)
1 1 = odd parity (implemented by the UART hardware)

bit 4 = stop bit select
0 = 1 stop bit
1 = 2 stop bits

bits 3:0 = baud rate (bits per second) select
0x0 = 1200
0x1 = 2400
0x2 = 4800
0x3 = 9600
0x4 = 14.4k
0x5 = 19.2k
0x6 = 28.8k
0x7 = 38.4k
0x8 = 57.6k
0x9 = 76.8k (this baud will not work with the TReX Jr Configurator Application)
0xA = 115.2k
0xB – 0xF = not used (invalid selection)

Cyclic redundancy checking is explained in detail in the CRC section of the user's guide. Once you enable cyclic redundancy checking and reset your TReX Jr, it will not accept any command packets that don't have the expected CRC byte tacked onto the end.

Note: the Set Configuration Parameter command (0xAF) will reject a UART settings parameter that requests an invalid baud rate. After setting this parameter, you must power-cycle the TReX Jr to activate the change.

0x7F: reset all parameters to their factory default values

If you want to reset all of the parameters to their factory default values, you can set parameter 0x7F to a value of **0x7F** and cycle the TReX Jr's power. You may notice a longer-than-normal delay before the TReX Jr restarts as it resets the parameters (especially if one of the parameters being reset is the CRC-7 polynomial). If you attempt to set parameter 0x7F to any value other than 0x7F, the TReX Jr will return a “bad value” byte and reject the Set Parameter command.