MYOSTAT MOTION
CONTROL INC.

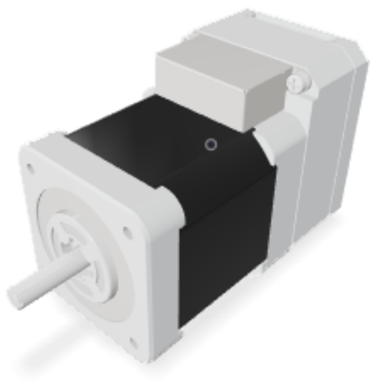
CM1-C User Guide

Version: 3.00
Date: 12/05/2025

1 Table of Contents

2 CM1-C DESIGN REFERENCES

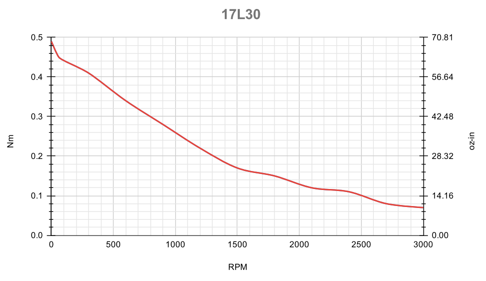
The CM1 Cool Muscle line of servo motors provide all the common components required for motion control embedded into the motor itself. The seamless integration of both software and hardware components creates highly efficient motion modules that can be easily integrated into existing designs, or used to shorten the development cycle of your new machine.



2.1 USER GUIDE

Use the search bar or links on the left to find the content you are looking for. Alternatively download a copy of the user guide to pdf.

2.3 TECHNICAL SPECIFICATIONS



2.4 DATASHEETS

- CM1-C-11 Datasheet
- CM1-C-17 Datasheet
- CM1-C-23 Datasheet

2.5 CAD FILES

3D Models

- All CAD files for the CM1 Series

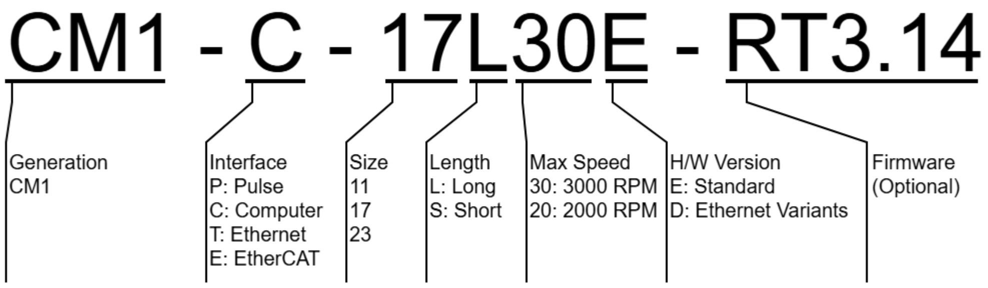
2.6 SOFTWARE

Programming IDE

- Control Room

2.2 PRODUCT BROCHURE

3 Motor Specifications



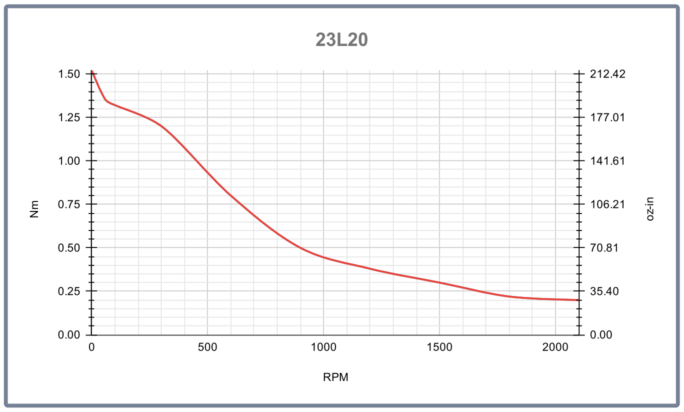
3.1 Specifications

Model	11L30	11S30	17L30	17S30	23L30	23S30	Unit
Max Speed	3000	3000	3000	3000	2000	3000	**RPM**
Continuous Torque	0.055 (7.78)	0.027 (3.8)	0.36 (50.98)	0.084 (11.89)	0.89 (126)	0.3 (42.48)	**NM (oz•in)**
Peak Torque	0.078 (11.1)	0.039 (5.5)	0.53 (75)	0.12 (16.56)	1.24 (175.6)	0.42 (65.14)	**NM (oz•in)**
Max Load Inertia	180 (2.5 x 10^{-3})	80 (1.1 x 10^{-3})	760 (1.07 x 10^{-2})	380 (5.38 x 10^{-3})	4.6 x 10^{3} (6.5 x 10^{-2})	1.4 x 10^{3} (1.9 x 10^{-2})	**g•cm^{2} (oz•in•s^{2})**
Motor Inertia	18 (2.5 x 10^{-4})	8 (1.1 x 10^{-3})	74 (1.04 x 10^{-3})	36 (5.09 x 10^{-4})	3.6 x 10^{2} (5.09 x 10^{-3})	180 (2.5 x 10^{-3})	**g•cm^{2} (oz•in•s^{2})**
Input Supply Voltage	24 ± 10%						**VDC**
Continuous Current	1.2	0.8	1.5	0.8	2.6	3.9	**A**
Peak Current	1.5	1.0	1.8	1.0	3.4	5.1	**A**
Operating Temp.	0 - 40						**°C**

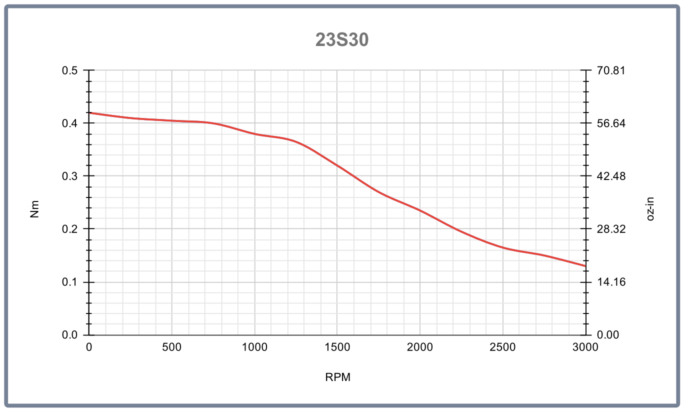
Storage Temperature	-20 - 60						**°C**
Operational Humidity	<90						**% Relative Humidity**
Shock	<10						**G**
Vibration	<1						**G**
Mass	295	234	470	323	1110	552	**g**

3.2 Torque-Speed Curves

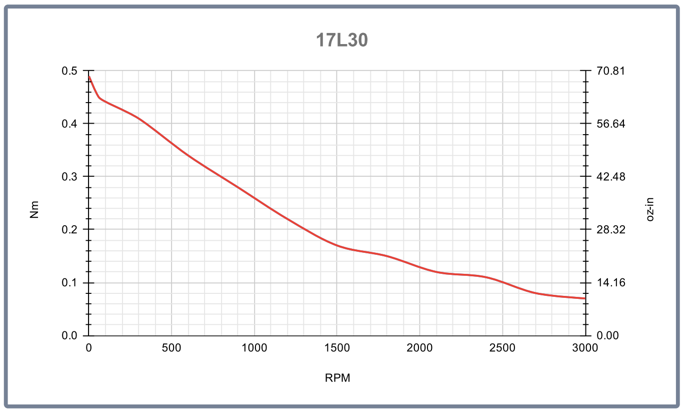
3.2.1 23L20



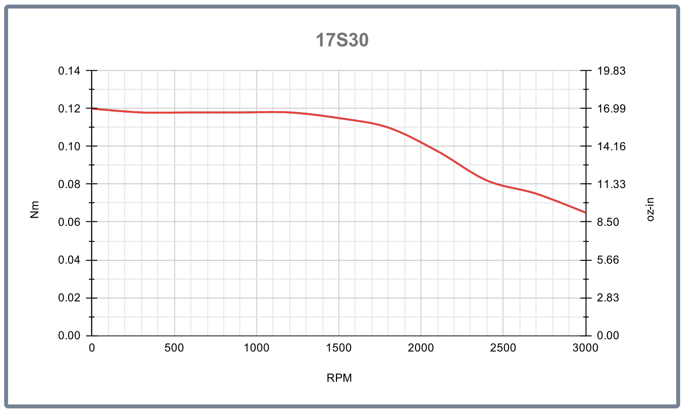
3.2.4 23S30



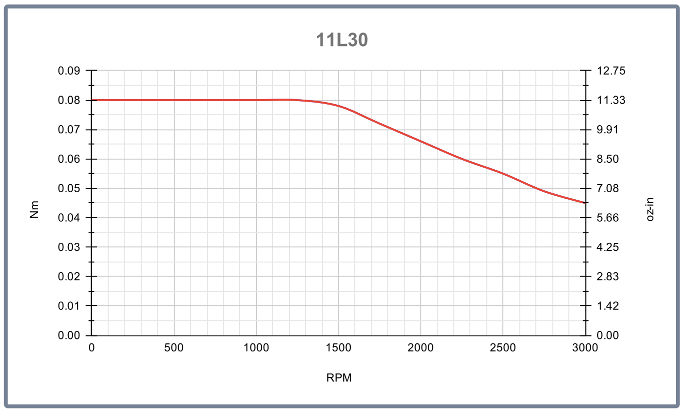
3.2.2 17L30



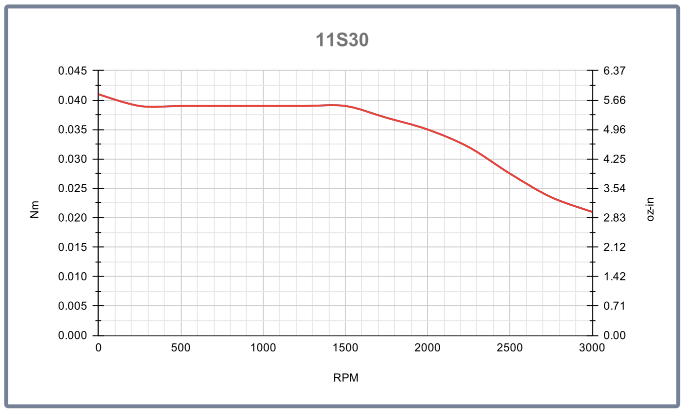
3.2.5 17L30



3.2.3 11L30



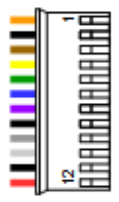
3.2.6 11S30



4 Getting Started

4.1 Wiring the CM1

The CM1 motor connects to everything via the 12-pin molex connector on the top of the motor. This connection includes power, communications, and all of the inputs and outputs to the motor.



1	+24VDC
2	GND
3	IN2-
4	OUT2
5	OUT1
6	IN4
7	IN3
8	IN1-
9	IN2+
10	IN1+
11	GND
12	+5VDC Out

1 Fig 1 - CM1 Molex Connector (51065-1200) pinout diagram

4.2 Power

The CM1 motor is powered by the orange and black, pins 1 and 2 on the 12-pin connector. Pin 1, orange, is the positive connector for +24VDC and pin 2, black, is for the ground connection. A green varistor is supplied with the motor. This should be placed across the power terminals to help reduce any spikes or voltage feedback from sudden changes in the speed of the motor. The varistor should be placed as close to the motor as possible.

A five volt output and ground are also provided from the motor on pins 12 and 11 respectively. This can be used to power inputs and outputs, or other devices as required. The five volt output has a maximum current of 50mA.

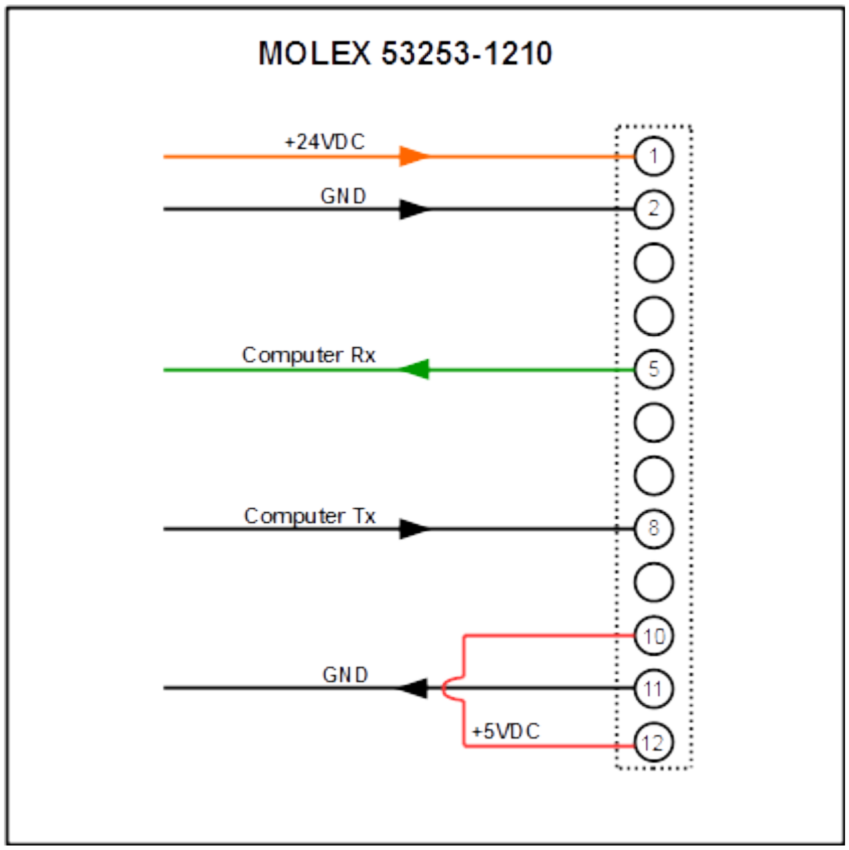
⚠ The power GND connections on the motor are common and are NOT chassis grounds.

4.3 Communications

The CM1 motor is capable of natively communicating TTL level serial communications. Other interfaces are available, for further information see *CM1 Interface Modules.*

Communications are achieved by using input one and output one as serial ports. This means that if you have your CM1 motor connected to a PC or other serial controller you will not be able to use input 1 or output 1 as digital I/O.

See the diagram below for how to connect the CM1 motor for standard TTL serial communications. The Rx and Tx in the diagram refer to the receive and transmit on the user device.



2 Fig. 2 - Wiring for Power and TTL Communications

The diagram in Fig.2 shows the connection for an inverted Tx from the computer, as used in the USB Y-Cable. For a non-inverted UART, the Tx should connect to pin 10, and pin 8 should be connected to ground.

4.3.1 Using a Y-Cable

The USB Y-Cable (CM1US1-1800) is the basic power and communications cable for use with the CM1 motor and your PC. This cable comes pre-wired with a 12-pin connector on one end, and the other end has two connections: a USB plug for connecting to your PC, and a pair of wires for connecting your 24VDC power supply. Upon plugging the USB plug in to your PC, windows should automatically install drivers for this cable. If this does not happen, or there is a problem with the driver installation you may also find these drivers at: http://myostat.ca/CoolMuscleSoftware under "USB Communications Cable Driver". This cable will emulate a serial COM port on your computer. Once connected, you can communicate to the motor using any standard terminal software, or the Control Room software.

An RS232 option for the Y-cable is also available with a DE-9 connector (CM1C3-1800). This is particularly useful for connecting the motor directly to PLCs or HMIs requiring serial communication.

4.3.2 Serial Communication Settings

The default baud rate for the motor is 38400.

If you are using a terminal software you will need to set it to:

8 data bits

1 stop bit

no parity

no flow control

Each command to the motor must be terminated with a carriage return.

4.4 Networking Multiple CM1 Motors

Up to 15 CM1 motors can be connected in a daisy chain network. While daisy chained, the motors are in communication with each other and you may communicate to all motors through the serial connection.

The CM1 network functions in a master/slave configuration. The first motor on the daisy chain will be designated motor one and this motor will be the master. All communications must pass through this motor.

When multiple motors are connected they are designated with a decimal number, called the motor ID, after all commands. For example, if I want to program a value in P1 for motor one (the master) and motor two, I need to send:

P1.1=100

P1.2=500

P1.1 refers to the P1 register in motor one, and P1.2 refers to the P1 register in motor 2. This is true for any command that you send to the motor.

If a motor ID is not specified, the motors will assume you are communicating to the last motor that was specified. In this way it is possible to send:

P1.1=100

P2=50

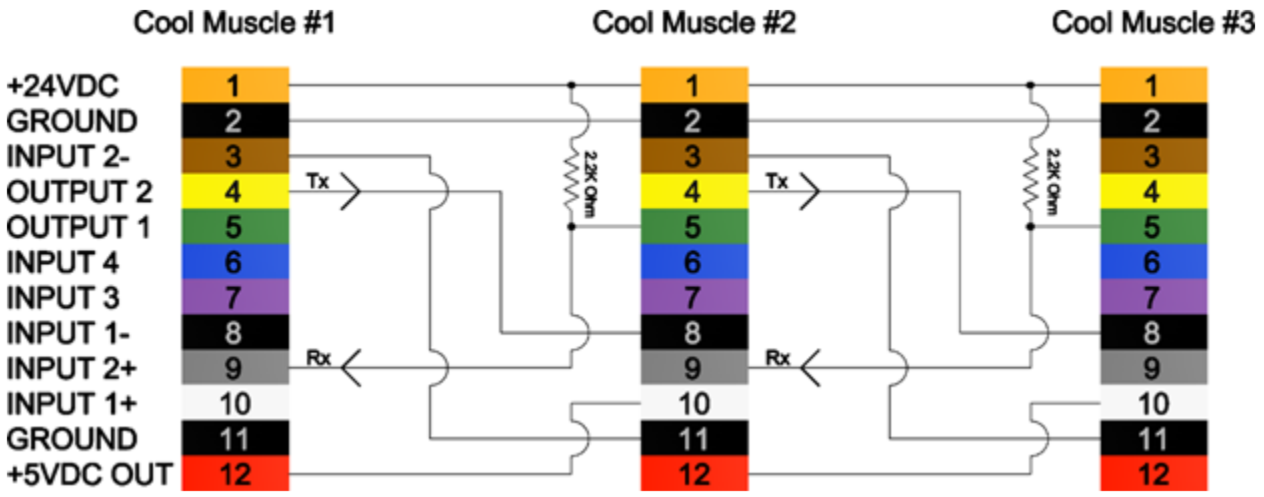
P3=500

P1.2=500

P2=1000

The network will understand this as me sending the first three values to motor one, and the last two values to motor two.

4.4.1 Daisy Chain Wiring



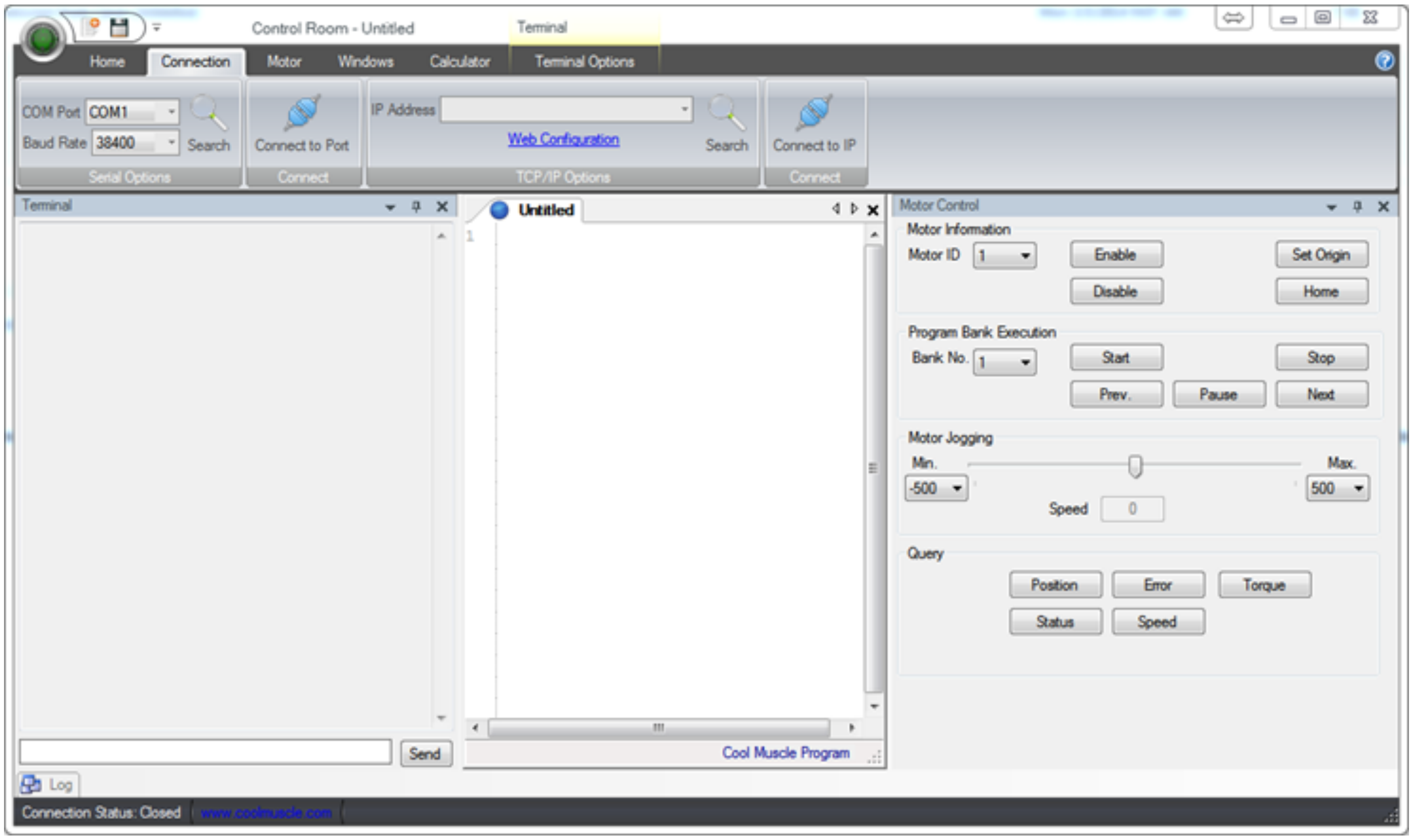
3 CM1 Daisy Chain Wiring

4.5 Communicating with your CM1 using Control Room

Control Room is the recommended control software for communicating and programming the CoolMuscle motors. This software can be downloaded from: http://myostat.ca/CoolMuscleSoftware under “CM1 and CM2 Interface Software”.

After Running Control Room you will be shown the main window. By default this consists of the Terminal window, the Program window, and the Control window.()

Along the top of the screen is the ribbon bar with tabs for various functions. When you first open the program you will be on the “connection” tab. Select the COM port you wish to connect to, as well as the baud rate. By default, the baud rate in the motor is 38400. Once you have selected the correct COM port and baud rate, press the “Connect to Port” button, and you will be connected and ready to communicate to the motor.



4 Control Room

For more information on Control Room, see Control Room.

5 Motor Parameters

Basic control of the cool muscle motor is performed by adjusting a set of basic parameters in the motor. These are: Speed, Acceleration, Position, and Torque. All of these parameters can be edited manually on the fly in Direct Mode, or automatically adjusted inside of a *Program or Logic bank*.

5.1 Pages

- Acceleration
- Position
- Speed
- Direction
- Status and Error States
- Torque
- Motor Type

5.2 Acceleration

The motors acceleration is measured in thousands of pulses per second squared ($Kpps^2$). This means that if you set the motors acceleration value to 10, for example, the motor will accelerate to the desired speed at 10,000 $pulse/sec^2$. For example, if the speed was set to 5000 pulses per second, we would reach this speed in half a second with an acceleration value of 10.

5.3 Position

The motors target position is measured in pulses. By default the motors resolution is 1000 pulses per revolution. This can be adjusted in discrete steps from 200 pulses per revolution all the way up to 50,000 pulses per revolution by changing the value of *K37*. Pulses are distributed evenly around the circumference of the motor; in the case of 1000 pulses per revolution, a position of 250 would be 90 degrees, and 500 would be 180 degrees.

If the position is set to 1,000,000,000 the motor will run indefinitely with no target stop point.

Positions can be called in a program in either an absolute or incremental fashion. For example, you can tell the motor to go to position 5000, and no matter where the motor currently is, it will travel to 5000 pulses from the origin. You can also increment the current position by 5000, meaning that the motor will advance from whatever its current position is by 5000 pulses.

For information on the position commands, see Program Bank Commands

5.4 Speed

The speed parameters in the motor are measured in pulses per second. The values you program in the motor are a multiple of the *speed unit* that is set in parameter *K37*. By Default, this speed unit is set to 100 pulses per second. This means that if you set the motors speed value to 50, the rotational speed of the motor will be 100pps x 50 = 5000 pulses per second.

In order to calculate the speed of the motor in revolutions per minute, you would use the following formula:

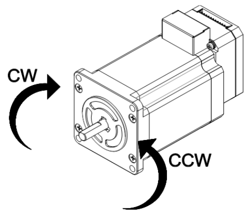
$$\frac{Speed\ Value\ x\ Speed\ Unit\ x\ 60}{Motor\ Resolution} = Motor\ Speed\ (RPM)$$

For example, using the above speed value of 50, the default speed unit of 100, and 1000 pulse per revolution resolution:

$$\frac{50\ x\ 100\ x\ 60}{1000} = 300\text{RPM}$$

When moving the motor, the sign of the speed is ignored, unless the position is set to 1,000,000,000. If this is the case, the motor will move indefinitely with no target position. The direction is then determined by whether the speed is a negative or positive value. If the speed is 0, the motor will stop.

5.5 Direction



The direction of the motor is always referred to as either clockwise or counterclockwise. The CW/CCW direction is determined when facing towards the output shaft of the motor. By default, the positive direction on the motor is clockwise, but this can be reversed in parameter K45.

5.6 Status and Error States

The motor status can be queried by sending the motor a *?99* command. The motor will then reply with a status number that responds to a specific status or error. An error may be any of the following discreet numbers, or a combination of them. For instance, a status of 8 would mean that the motor is ready and in position, and a status of 32 would mean that the push mode torque limit has been reached. It is also possible to have a status of 40 which would mean both of these statuses are true. Due to the numbering of the statuses, there is only one possible way for these sums to be made.

5.6.1 0: Motor Running

Any time the motor is executing a move, the status will be 0.

5.6.2 1: Position Error Overflow

A position error overflow means that the position error, the distance between the actual motor position and the instantaneous target position, is too great. This will cause the motor to disable. This can occur when the load is too great and the motor cannot accelerate to the required speed in the time it needs to, or if something is impeding the movement of the motor. The amount of position error that is required to cause this error can be changed in *K56*.

5.6.3 2: Over Speed/Voltage

This error can occur under two conditions: the speed of the motor is too great, or the input voltage is too high. This will result in the motor being disabled. The most likely reason for this error is that the speed of the motor is too high. See Motor Specifications for the maximum speed of your motor.

The other possible reason for this error is that the voltage across the 24V input and GND is too high. This can occur via voltage spikes from the power source, or conversely from regenerated voltage from the motor. If the motor is accelerating a very high load, it is possible for it to produce large regenerated voltage spikes. This can be mitigated by utilizing the varistor which comes with the CM1 motor, or by using a SRL module which contains an addition voltage regulation circuit. This error will occur instantaneously when the voltage rises above the limit programmed in parameter *K72*.

5.6.4 4: Over Load/Current

The over load error will occur whenever the motor current, and by extension the torque on the motor, exceeds the maximum rated torque of the motor for a certain amount of time. This will result in the motor being disabled. The amount of time that is required to cause this error is able to be changed in parameter *K57*. See Motor Specifications for the rated torque of your motor.

5.6.5 8: In Position / Ready

This is the normal resting status of the motor. This status means that the motor has reached the target position and is now ready and waiting for another command.

5.6.6 16: Disabled

The disabled status will only occur if the motor is manually disabled through the) command, or through an input programmed as “Motor Free”. This status will not occur if the motor is disabled via another method or as the result of another error disabling the motor.

5.6.7 32: Push Mode Torque Limit Reached

This status will appear when using push mode. When the motor reaches the torque limit programmed in parameter *K60* the status will change to this until the push move finishes or another command is executed.

5.6.8 128: Over Temperature

The over temperature error will occur if the motors control board located at the back of the motor reaches a temperature threshold, as programmed in *K71*. This can occur with large loads, large duty cycles, and high speeds.

5.6.9 256: Push Mode Timeout not Reached

The push mode timeout error will occur when the motor is executing a push move, and the target position is reached. This means that there is no push being executed, as the motor has reached the position without resistance. When this error occurs, the current program bank will be paused, but the motor will remain enabled. Sending a run command to the motor will resume the bank at the next step.

5.6.10 512: Emergency Stop

The emergency stop status will occur whenever the emergency stop command, * is sent to the motor, or when an input programmed as an emergency stop is triggered. This error will be present if it is triggered on any motor on the daisy chain network. Under this condition the motor will not disable, but it will halt movement at its current position, as well as ceasing any running programs or actions and will not respond to further commands until the error is cleared. This error can be cleared by sending the **1* command or by deactivating the emergency stop input.

5.7 Torque

The amount of torque for a specific move can be controlled by using the M register. You can set register values, such as M1=80, M2=50, M3=100, and when you call these in conjunction with a move command the motor will utilize 80%, 50%, or 100% of full torque respectively. The torque value that you program is a percentage. Valid entries are from 0 to 100. This percentage specifies the percentage of the motors peak torque to be used for the move. See Motor Specifications for peak torque.

For example:

```
M1=75
A1=100
S1=100
P1=1000


B1
A1,S1,M1,P1
End
```

The above program, when run using default settings, will move the motor to position 1000, one full rotation, at a speed of 10Kpps and acceleration of 100Kpps^2 utilizing 75% of the motors full torque.

5.8 Motor Type

There are two variants of the CM1 that are available. The C-type and P-type.

5.8.1 C-Type

C-type stands for "computer type" and is the standard version of the motor used most often. A C-type motor has the most functionality and can be run from serial commands sent from an external source, by a program loaded in to the motor, or by programmable inputs.

5.8.2 P-Type

P-Type stand for "Pulse Type". This motor cannot be programmed and can only by run from an external controller sending pulses to the motor. A p-type motor can be run with pulses dictating step/direction, or clockwise/counter clockwise operation as dictated by parameter K36.

For information on wiring IN1 and IN2, see Input Wiring.

Step and Direction

In step and direction mode, IN1 is used for your Step signal and IN2 is used for your direction signal.

When your direction signal on IN2 is off, if the step signal on IN1 is pulsed, the motor will step in the clockwise direction. If the direction signal is on and the step signal is pulsed, the motor will step in the counter-clockwise direction. For each pulse on the step signal, the motor will move one pulse in the desired direction. By Default, the resolution of the motor is 1000 pulses per revolution, and this is true for the P-type as well. This means that with the default settings, for each revolution of the motor, you will need to pulse the step signal 1000 times.

CW/CCW

In CW/CCW mode, IN1 is used for the CW signal and IN2 is used for the CCW signal. In this mode, for every pulse on IN1, the CW signal, the motor will step once in the clockwise direction. Likewise, for each pulse on IN2, the CCW signal, the motor will step once in the counter-clockwise direction.

6 Inputs and Outputs

The cool muscle motor is equipped with a number of inputs and outputs which can be configured to do various operations.

IN1 and OUT1 are used as serial communication ports when communicating to the motor from a computer or controller. If you wish to use IN1 and OUT1 as digital I/O ports, you will not be able to communicate via the serial port at the same time.

If you are daisy chaining multiple motors, IN1 and OUT1 will be used to communicate to any motors upstream of the current motor, and IN2 and OUT2 will be used to communicate to any motors downstream of the current motor. See Daisy Chain Wiring for detailed information.

6.1 I/O Auto Detection

Upon powering up the motor, it will observe the state of IN1 and IN2. If one of these is active when powering up, the motor assumes that it is connected to a serial communications device. If the input is not activated upon powering up, the motor assumes that there is no communications connected and will set the inputs and outputs to be used as digital I/O.

This can cause problems in some cases. By changing the setting of parameter K52, you can disable the auto detect feature and set I/O 1 and 2 to a specific function.

⚠ IN1 and OUT1 cannot be forced to be digital I/O, but can be forced to be serial communications.

- Outputs
  - Output Functions
- Inputs
  - Input Functions

6.2 Outputs

The cool muscle motor has two available outputs. These outputs are both common emitter sinking outputs. This means that they will provide a path to the ground connection for your load when active, and will be floating when inactive.

6.2.1 Output Specification

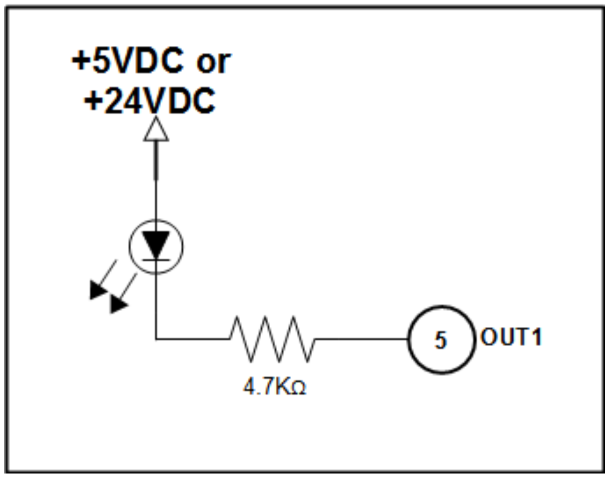
OUT1 and OUT2	Min	Max	Unit
Voltage Range	0	24	VDC
Operating Current	0	50	mA
Switching Frequency	-	1000*	Hz
Pulse Width	1*	-	ms

⚠ *When using Quadrature Encoder Output, this will increase to 5000Hz, or 200µs

The output current is shared between both OUT1 and OUT2, IN3 and IN4, and the main 5VDC output on the motor. The current draw for all of these sources combined should not exceed 200mA.

Output Wiring

The example in Fig. 8 illustrates connecting an LED to OUT1. We are using a 4.7K? resistor for current limiting when connected to a 24VDC supply. When OUT1 is active, there will be a path to ground through OUT1 and will illuminate the LED.

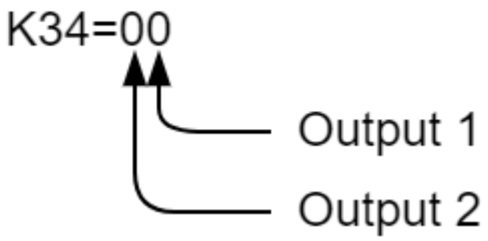


5 Fig.8 - Connecting an LED to OUT1

6.2.2 Output Functions

Output Functions

Output 1 and output 2 are assigned values as follows on K34:



Value	Function	Description
0	AO2 - Analog Output	Write to the analog output using a variable assigned to "AO2"
1	In Position	Output indicate the motors In Position status

2	Alarm	Output is asserted when the motor is in an alarm state
3	O1/F1	Only applicable to OUT1. Use O1 and F1 to manually switch OUT1 on/off
4	O2/F2	Only applicable to OUT2. Use O2 and F2 to manually switch OUT2 on/off
5	Analog Output	Analog output value is set by K35
6	Merge Motion	Digital output is asserted when a merge motion is executed
7	Quadrature/Index Output	OUT1 and OUT2 are combined to output a quadrature signal tracking the motors position
8	Motor Free	Digital output indicates whether is in a motor free state or not
9	Push Mode Torque Limit Reached	Digital output is asserted when the push mode torque limit is reached

Default: K34=21

AO2

AO2 will only work on output 2. Programming output 2 as AO2 will set output 2 to analog and allow you to control the analog value of the output via a variable programmed as “AO2”. The valid range for this variable is 0 to 255, corresponding to 0VDC to 5VDC. To use the Analog output variable, K34 must be set to 0.

In Position

The In Position Output type will activate whenever the motor status is “Ux=8”, which means that the motor has reached its target position and is in the ready state. Any errors or change in status will deactivate this output.

Alarm

The Alarm output will activate whenever the motor is in an alarmed state. The alarmed state refers to any error status on the motor which will cause the motor to disable. This includes over load, over speed, and position error overflow errors. This output will not activate if the motor is manually disabled nor if an emergency stop is triggered.

CML O1/F1

This function will only operate on output 1. By programming output 1 as CML O1/F1, the output can be manually controlled by sending the O1 command to turn on output 1 or the F1 command to turn off output 1. These commands can be sent either manually via the serial communications, or they can also be added to a program or logic bank to execute at a specified time.

CML O2/F2

This function is the same as the O1/F1 command, but will only operate on output 2.

Analog Output

By programming the output as analog output, it will be an analog voltage output which will behave as programmed in parameter *K35*. The range of output voltages on the analog output is 0VDC to 5VDC as measured between the OUT and +5V terminals.

Merge Motion

When using a merge motion command, the merge motion output will activate when the motor passes by a target position and is on the way to the next passing point. The length of the output pulse is determined by parameter *K73*.

Quadrature/Index Output

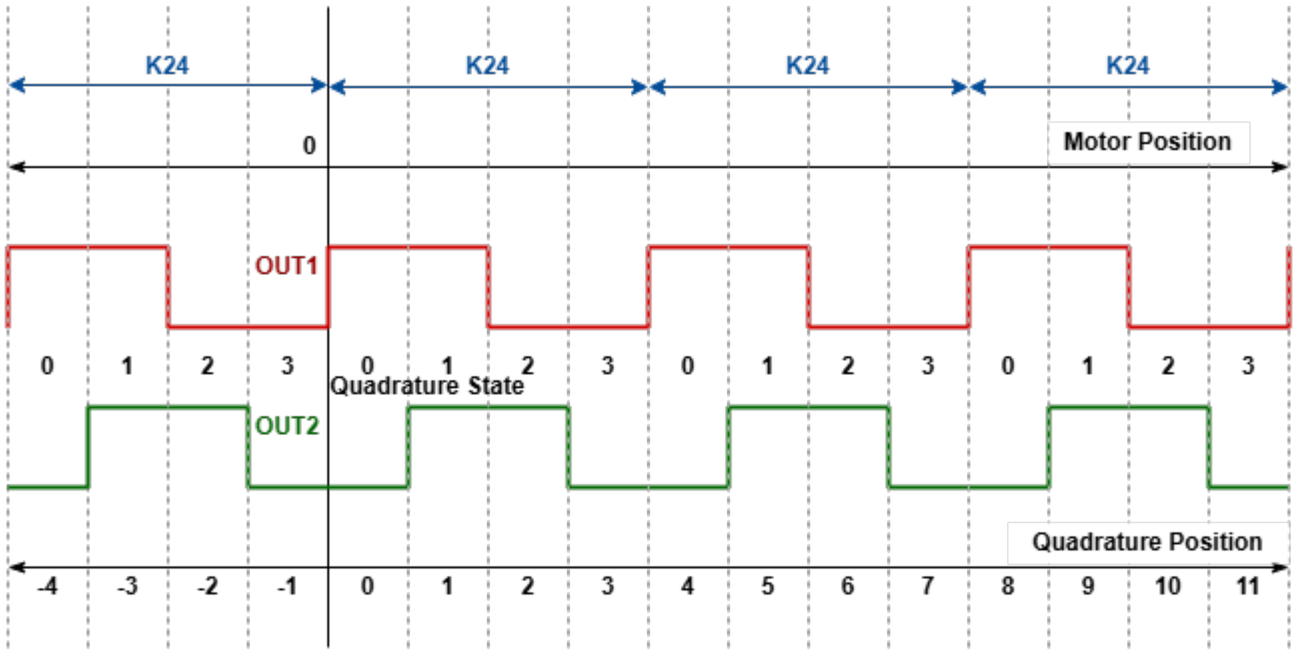
The quadrature output can be used with dual outputs as a quadrature signal or as single outputs giving an index output.

The following additional K parameters are used when setting up the quadrature/index output

Parameter	**Name**	**Description**
K24	Output Interval (pulses)	K24 sets the value of the quadrature interval • Value is for a full cycle of a single output • Actual quadrature count will be ¼ of K24 Example: K24=100 will output a full wave form on OUT1 and OUT2 for each 100 pulses the motor moves. If the signals are combined and read as a quadrature encoder there will be an incremental encoder count each 25 pulses.
K33	Invert Output	K33 inverts the output. • Invert a single output to output the quadrature in reverse
K54	Quadrature Interval Offset (pulses)	K54 sets an optional offset which shifts that phase of each output by the offset amount.

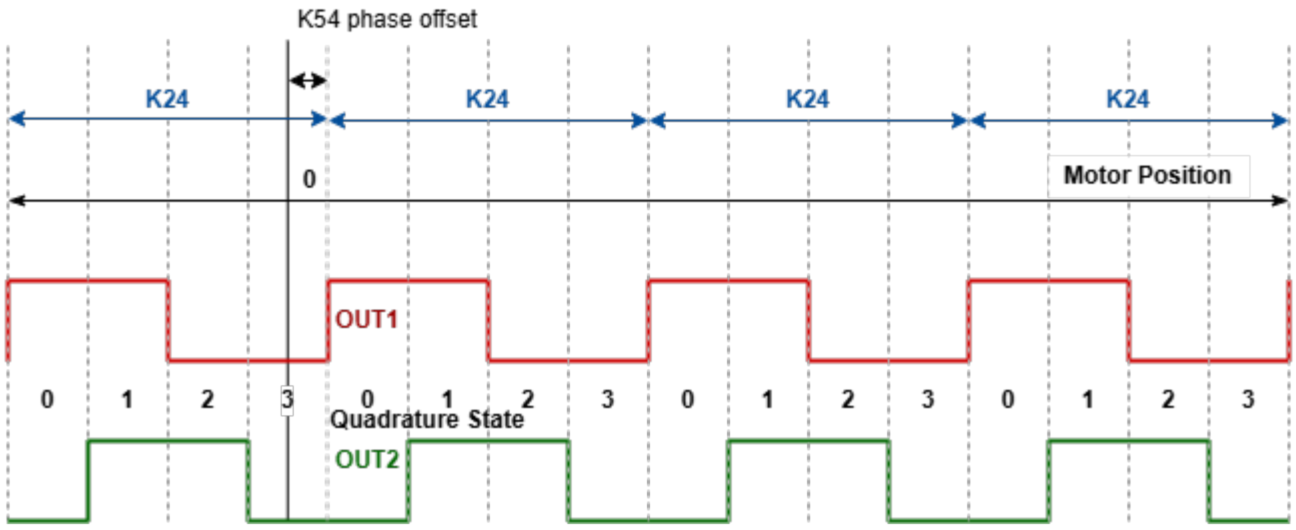
Quadrature Output

K34=77 must be set to have both outputs generate a quadrature signal. The diagram below shows the individual output signals and their timing



K54 - Phase Shift

K54 can be used to shift the signal by the number of pulses in K54,



Index Output

If either OUT1 or OUT2 are set to 7 in K34 they will output a single phase based on the motor's position. As seen in the above diagram:

- OUT1 signal is inline with 0
- OUT2 signal is phase shifted by 90°

Motor Free

The motor free output activates whenever the motor is disabled. This can be either from being manually disabled, or from an error or alarm state which caused the motor to disable itself. This output will deactivate when the motor is re-enabled.

Push Mode Torque Limit Reached

This output will activate whenever you are using a push move, and the torque limit programmed in *K80* is reached. The output will deactivate when the push time programmed in *K61* expires and the push move ends.

6.3 Inputs

IN1 and IN2 on the motor are photo-coupled and can be connected in either a sinking or sourcing configuration. IN3 and IN4 are sourcing inputs, and IN4 can also be configured to act as an analog voltage input.

6.3.1 Input Specifications

Digital Inputs 1 & 2	**Minimum**	**Maximum**	**Unit**
Voltage Range	0	24	VDC
Low Level	0	0.8	VDC
High Level	2.4	24	VDC
Operating Current	7	15	mA
Switching Frequency	-	500	KHz
Pulse Width	0.8	-	µs
Serial UART	**Minimum**	**Maximum**	**Unit**
Voltage Range	0	5	VDC
Communication Speed	9600	512000	bps

Digital Inputs 3 & 4	**Minimum**	**Maximum**	**Unit**
Voltage Range	0	5	VDC
Low Level	0	0.8	VDC
High Level	2.4	5	VDC
Operating Current	7	15	mA
Switching Frequency	-	833	Hz
Pulse Width	10	-	ms

Analog IN4	**Minimum**	**Maximum**	**Unit**
Voltage Range	0	5	VDC
A/D Resolution	-	10	bits
Numerical Resolution	0	1023	-

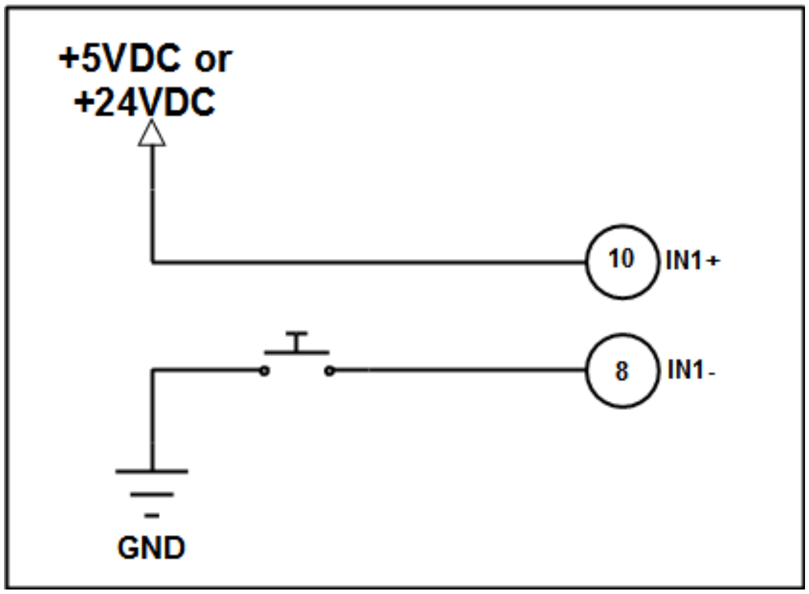
6.3.2 Input Wiring

IN1 and IN2

IN1 and IN2 can be configured as either sinking or sourcing inputs. They will function as long as the appropriate voltage is seen between the IN+ and IN- Terminals. The following examples utilize a push button to act as the input trigger. In reality this push button would often be replaced by an output from another device, such as a PLC, or other controller. IN1 and IN2 can operate at voltages up to 24VDC. Any of the below examples showing IN1 are also true for IN2.

IN1 - Sourcing Input

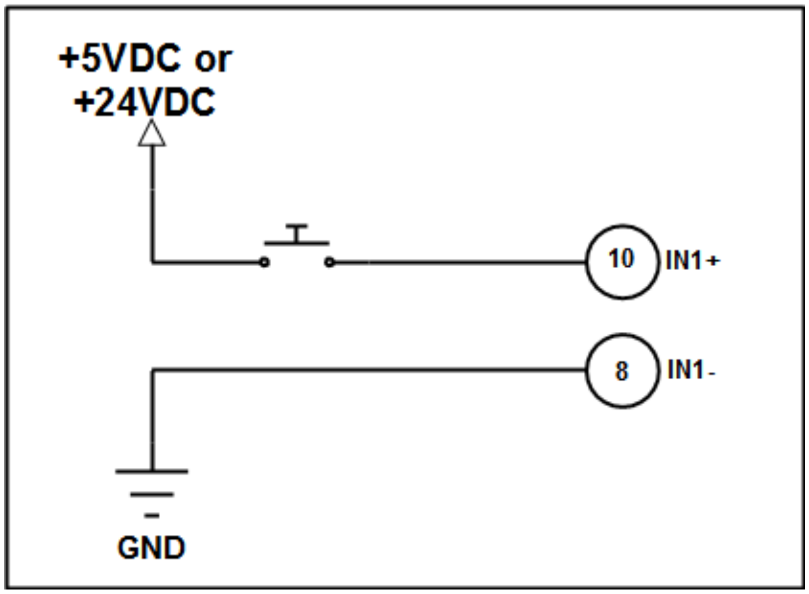
IN1 and IN2 can be configured as either sinking or sourcing inputs. They will function as long as the appropriate voltage is seen between the IN+ and IN- Terminals. The following examples utilize a push button to act as the input trigger. In reality this push button would often be replaced by an output from another device, such as a PLC, or other controller. IN1 and IN2 can operate at voltages up to 24VDC. In Fig.1, a push button is used to connect the negative side of IN1 to ground. Since the positive side of IN1 is connected to the positive voltage source, this results in the activation of IN1.



6 Fig. 1 - IN1 as a sourcing input

IN1 - Sinking Input

In Fig.2, a push button is being used to connect the positive side of IN1 to the +5VDC supply from the motor. Since the negative side of the input is connected to the ground, this results in the activation of IN1.



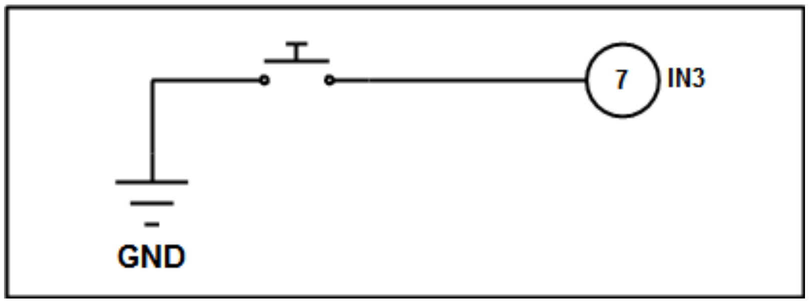
7 Fig. 2 - IN1 as a sinking input

IN3 and IN4

IN3 and IN4 are both 0V inputs. They have an internal 4.7K pullup resistor to 5VDC. They can tolerate a maximum of 5V.

IN3 connected to a Push Button

The example in Fig.3 illustrates using a push button to connect the input to ground to activate IN3.



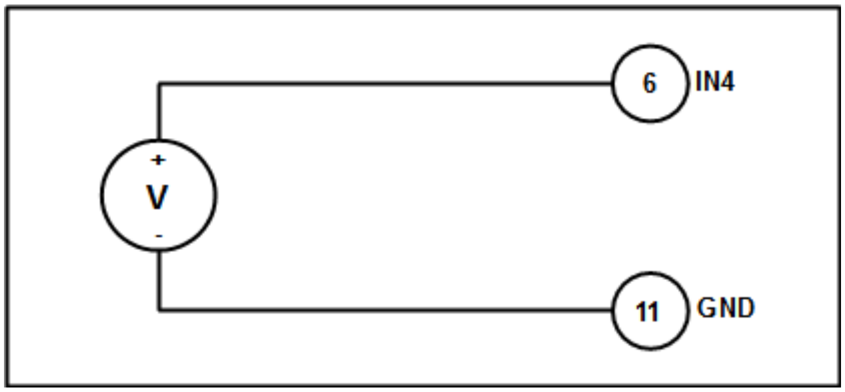
8 Fig. 3 - IN3 using a push button

6.3.3 Analog Input

IN4 on the cool muscle can be configured to act as an analog input. This analog input has a working voltage of 0-5VDC and a ten bit resolution. This means that the digital conversion reading you get from the motor will be a value from 0 to 1023. This gives an effective resolution of about 204 counts per volt. For instance, if you have an analog voltage of 2VDC applied to the input, you should see corresponding value in the motor of about 408.

Analog Input Wiring

The analog voltage source should be wired between the IN4 terminal, and the Ground terminal, as showing in Fig. 4.



9 Fig. 4 - Analog input connection

Setting the Analog Input to a Variable

We can obtain the digital conversion of the analog value directly by setting a variable in the V register as “AIN4”, for example V2.1=”AIN4” will set variable 2 in motor 1 to be equal to the current analog input value. This will allow us to grab that value and we can then perform mathematical operations or compare it in order to achieve more complex results.

Analog Speed and Position Control

It is possible to configure the analog input on the Cool Muscle to directly control speed or position without having to write a program. There are five parameters which will affect this behavior:

Parameter	Description
K38	Analog Interface (Set speed or position control)
K39	Voltage Filter Gain
K40	Max Speed (only used for speed control)
K41	Travel Range (only used for position control)
K64	Analog Input Function

The parameter K64 allows us to set a number of different functions for the analog input. For instance, we can have register P25 always equal to the analog input value, or use the analog input as a multiplier for position values. In order to use the speed and position control, we set K64=9. This sets the analog input to analog control only mode. Once we have done this, we set K38. If K38=0 we are using speed control, if K38=1 we are using position control.

The next step is to set our limits. If we are using speed control, we will set our max speed value in K40 in RPM. For example, if we set K40=300 the motors speed would vary from 0RPM to 300RPM when the analog input voltage varies from 0VDC to 4.8VDC respectively.

If we are using position control, we set the max travel range in K41 in pulses. This means that if we set K41=1000, the motors position would vary from 0pulses to 1000pulses as the analog input voltage varies from 0VDC to 4.8VDC respectively.

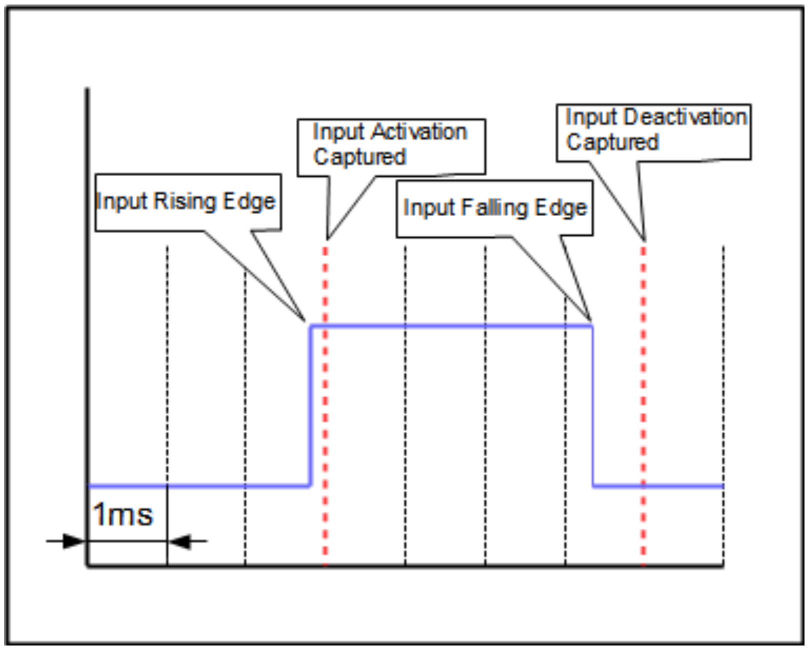
The final parameter to set would be K39. This is a voltage filter. You will use this if there is lots of ripple or bounce in your analog signal that you don't want the motor to respond to. The higher the value of K39, the less sensitive the analog input will be to small disturbances.

6.3.4 Input Activation

There are six methods for activating an input: Logical high quick response, logical high slow response, quick response rising edge, quick response falling edge, slow response rising edge, and slow response falling edge. These can all be used in conjunction with one another simultaneously. This means that you could have one input doing six different things depending on these conditions.

Quick Response

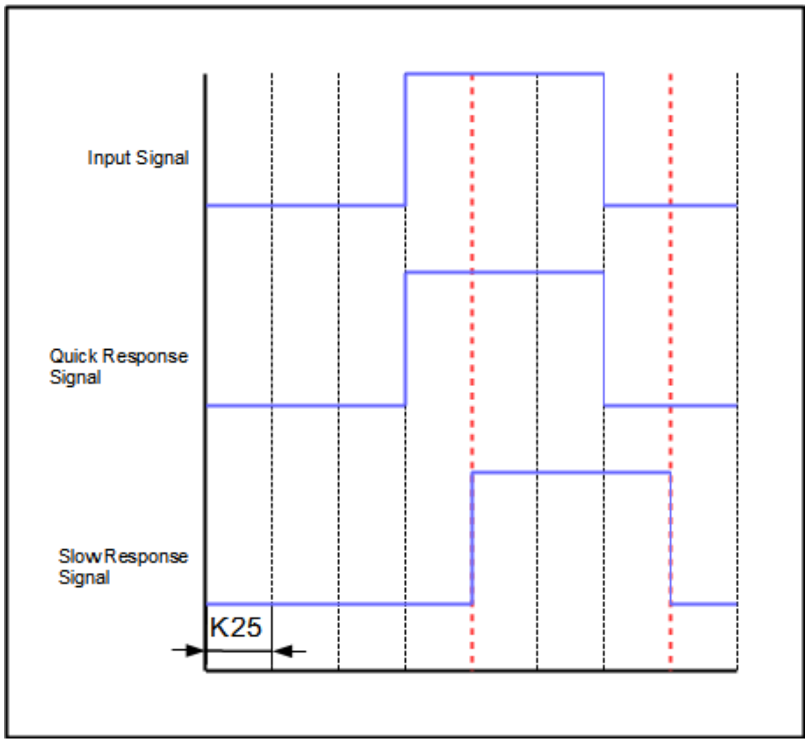
For the quick response conditions, the inputs are scanned every 1ms. This means that when the input is triggered, there is a maximum of 1ms before the input will register as activated within the motor, depending on where in the scanning cycle it is triggered. Likewise, when the input is shut off, there is a maximum of 1ms before the input will register as deactivated within the motor. The quick response logical high, rising edge, and falling edge functions are programmed in parameters K27, K28, and K29 respectively. Fig. 5 demonstrates the input activation cycle for the quick response input.



10 Fig. 5 - Quick response input activation

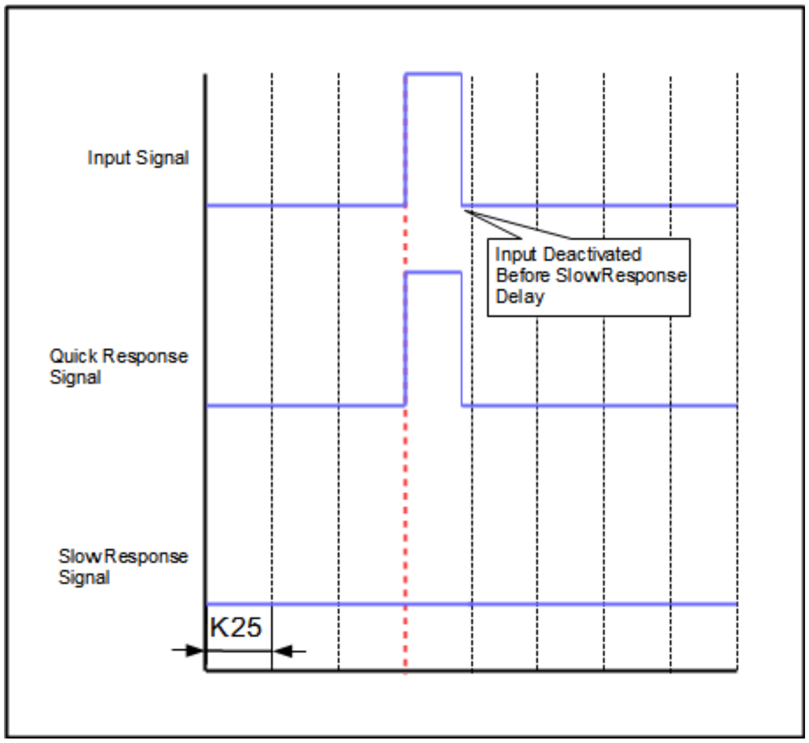
Slow Response

The slow response conditions are triggered at a programmable delay after the quick response is triggered. The delay can be found in parameter K25. For example, if the delay is programmed for 0.2sec in K25, when the input is activated, the quick response will trigger withing the next 1ms, and the slow response functions will trigger 0.2sec after that.



11 Fig. 6 - Slow response input activation

If the Input is deactivated before the slow signal response time is reached, only the quick response function is triggered and the slow response is not recognized.



12 Fig. 7 - Slow response input not recognized

6.3.5 Input Functions

Input Functions

The following is a list of the various functions which can be assigned to an input. Some of these are only available for specific activation conditions. Please see parameters *K27-K32* for more info.

No Action

If an input is programmed for No Action, this means that the input will not be used and have no response under the given activation condition.

General Use

The General Use setting is only available under the logical high settings. If an input is programmed as General Use, there will be no automatic response to being activated but the input status conditions may be used within logic or program banks.

Origin Sensor

Allows you to designate an input as the origin sensor. If the motor is programmed to home to a sensor, it will use this input to designate home.

Manual Feed CW/CCW

Upon activation of an input programmed as Manual Feed, the motor will rotate CW or CCW accordingly at the manual feed speed programmed in *K49* until the input is deactivated.

CW/CCW Limit Switch

An input programmed as a Limit Switch will act to cease rotation in that direction when triggered. For instance, if the motor is rotating clockwise and an input is triggered which is programmed as CW Limit Switch, the motor will come to a stop and output the "Ux=8" in position status as well as the "error: CW Limit!!" message. Further rotation in the clockwise direction will not be possible when the limit switch is activated. In addition, if the motor is set to home to a sensor, the activation of the respective limit switch will also designate the home position.

Emergency Stop

Activation of the Emergency Stop input will cause the motor to stop immediately, stopping any running program banks, hold position, and output the "Ux=512" emergency stop status. While the emergency stop status is active, any motion commands will be ignored, however, logic banks will continue to run. In order to return the motor to normal operation the input must be deactivated. Any operations which were in progress when the emergency stop was activated will not resume automatically upon removal of the emergency stop status.

Full Stop

The activation of the Full Stop input will cause any currently running program banks to come to a complete stop.

Alarm Reset/Pause

This input type has two types of operation: if the motor is currently in an error or alarm state, activation of this input will clear the status and return the motor to its normal ready state. If the error persists, the motor may return directly to the error state. If there is currently a program bank running, this input will pause the bank at the current step and bring the motor to a complete stop. When the bank is restarted, either by serial command or by input activation, the bank will continue from where it was paused as opposed to starting all over again, as if you had used the full stop command.

Motor Free

Upon activation of the Motor Free Input, the motor will become disabled and the status will change to "Ux=16". In this state, the motor will not run, and all control will be shut off; the motor can be turned freely. Any logic banks will continue to run.

Reset Position Counter

When this input is activated, the motors current position will be set to 0.

Execute Next Step

If there is a program bank running which has been paused, either through a serial command or through activation of the Pause input, activating this input will cause the motor to execute the next step in that program bank and then pause again. In this way it is possible to step through individual lines of a program. This input will do nothing if a program bank is not already started, and has no effect on logic banks.

Execute Previous Step

If there is a program bank running which has been paused, either through a serial command or through activation of the Pause input, activating this input will cause the motor to execute the previous step in that program bank and then pause again. You may only step back one line previous, any attempt to go back more than one step will result in the message “Can’t back!” being sent and no action from the motor. When the program bank has been stepped through completely, the “End!” command is sent from the motor and you will not be able to step back again, as the bank is no longer running.

Execute Bank 1

Activating this input will cause the motor to begin running any program stored in program bank one. This will only execute program bank one if the motor is in the ready state, if there is already a bank running or if the motor is in an error state, nothing will happen.

Go Origin

Upon activation of the Go Origin input, the motor will perform an origin search as defined by *K46*. Activating this input will cause the motor to stop and search for the origin even if there is a program bank or motion currently running. For more information on the origin search and associated parameters, see Origin Search.

Jog CW/CCW or Execute Bank 2/3

Depending on the programming of K36, this input type could have two different functions. By default, activating this input will cause the motor to travel the respective direction by the number of pulses programmed in *K50*. The speed of this jog is not programmable and will always be the maximum possible travel speed.

If K36=2, this will change the operation of the input. Much like the Execute Bank 1 function, this will set the inputs to activate program banks two or three respectively.

Enable Motor

If the Enable Motor input is activated, this will re-activate the motor if it has been disabled via a Motor Free input, a serial command to disable the motor, or an error such as an over torque condition which caused the motor to disable.

7 Origin Search

The cool muscle motor is capable of performing an origin search and home itself automatically, or on command. Homing can be performed to an origin switch or to a hard stop. There are six different K parameters that define the operation of the origin search:

Parameter	Description
K42	Origin Search Speed
K43	Origin Search Accleration
K45	Origin Search Direction
K46	Origin Search Method
K47	Origin Stopper Torque
K48	Origin Offset Distance

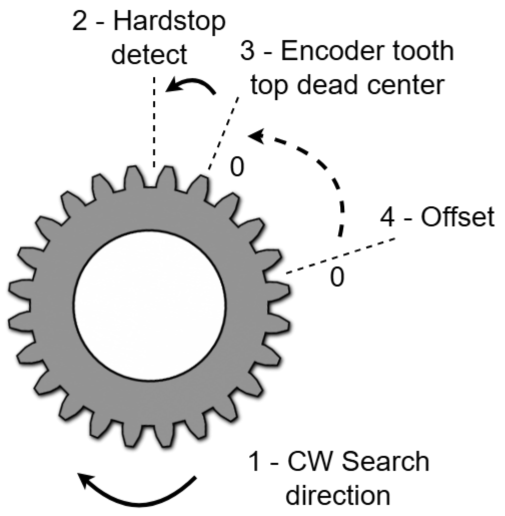
The origin search will timeout after 5 minutes.

7.1 Homing Methods

Two distinct methods are available when executing a home routine.

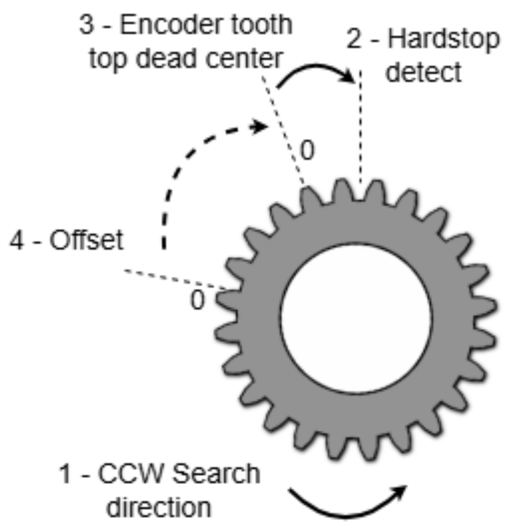
1. Hardstop detect - the motor moves towards a hardstop. When the hardstop is reached the driver reads the position of the magnetic encoder. It then moves back 2 teeth of the encoder gear to set the absolute position. Moving back the two teeth allows for high repeatability as fatigue and variance in the hardstop detect can be accounted for automatically.
2. Sensor detect - the motor moves towards a home sensor. When the sensor is triggered the driver interrupt captures the motor position. The motor decelerates and moves back to the captured position.
    a. If the sensor is detect when the home routine is first initiated the motor will move off the sensor first in the opposite direction to the defined origin search direction.

7.1.1 Hardstop CW -



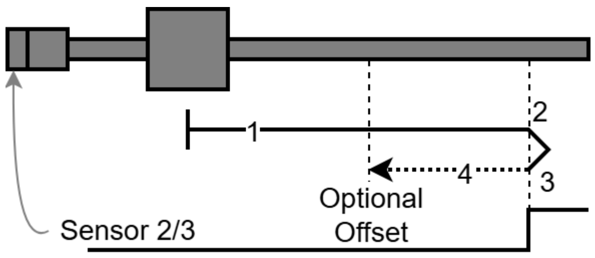
1. Motor runs CW towards a hardstop.
2. The driver monitors the motor current and detects when a hardstop has been reached.
3. Motor backs up CCW to the 2nd previous encoder tooth (improved repeatability) and sets the position to 0.
4. If an offset is set the motor moves the offset distance
    a. Care should be taken not to set a +ve offset as this will move the load into the hardstop
5. Sets the final position to 0.

7.1.2 Hardstop CCW -



1. Motor runs CCW towards a hardstop.
2. The driver monitors the motor current and detects when a hardstop has been reached.
3. Motor backs CW up to the 2nd previous encoder tooth (improved repeatability) and sets the position to 0.
4. If an offset is set the motor moves the offset distance.
    a. Care should be taken not to set a -ve offset as this will drive the load into the hardstop
5. Sets the final position to 0.

7.1.3 Sensor CW -

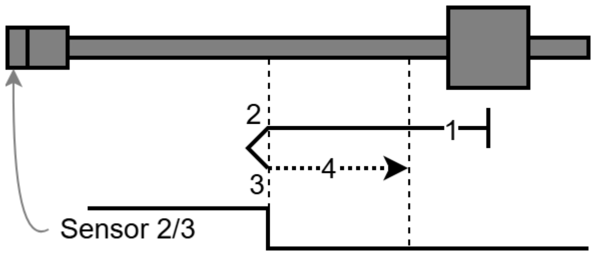


- Diagram shown with motor connected to linear actuator.
    - CW rotation moves the load left to right
- Sensor is connected directly on IN2 or IN3

1. Motor runs CW waiting for sensor input
2. Sensor rising edge triggers the input, motor captures encoder position, decelerates and reverses direction

3. Motor stops on captured position and sets position to 0
4. If an offset is defined the motor continues to the defined offset and sets the final target position to 0.
    a. Offset could be +ve or -ve. The above diagram shows a -ve offset

7.1.4 Sensor CCW -



- Diagram shown with motor connected to linear actuator.
    - CCW rotation moves the load right to left
- Sensor is connected directly on IN2 or IN3

8 RS-485

The Cool Muscle supports communications via a RS-485 type half duplex communications protocol. The actual physical interface still utilizes the regular voltage levels and serial interface hardware, but utilizing an RS-485 style half duplex communications.

In RS-485 mode, when the motor has a message to communicate it will stream a {# where # is its motor ID number. When the motor receives this same command back to acknowledge the request it will then transmit its message followed by {0 to close communications. Parameter K62 will determine whether the motor is communicating in RS-485 mode and what the node ID is. If K62=0, RS-485 mode is off, otherwise that is the node ID of the motor.

8.1 RS-485 Commands

D	Set Node ID
Sets the node ID of a motor with a specific serial number. The Syntax is D# = SN where # is the desired node ID, and SN is the serial number of the motor.	
D12=103490138	Sets the node ID of motor with serial number 103490138 to 12

{0	Broadcast
Opens RS-485 communications to all nodes. Also used to close communications once a message has been issued.	
{0	Begin command broadcast to all motors or end communications.

{#	Address Node
Address node with ID#, where # is any number from 1-255. Only node ID# will communicate until {0 is issued.	

{1	Address node 1

8.2 Disabling RS-485

It is not unusual for someone to accidentally set the motor to RS-485 mode and become confused when the motor will only stream out a series of “{1” and not respond. In this case, the motor is requesting communication access and has something to report. To disable RS-485 and switch back to standard serial communications we must:

1. Give the motor communications access
2. Let the motor report its message
3. Gain communications access to the motor
4. Set K62=0 to disable RS-485 communications

The process for doing this is as follows:

1. Acknowledge the motors request for communications. For instance, if the motor is sending {1 to request communications, we send back a {1 to allow it access.
2. The motor will then report the message it is trying to communicate, followed by a {0 to end communications
3. To gain communications access to the motor, we address that motor ID again, for example if the motor is node 1 as in the above example, we send a {1 again.
4. Now that the correct motor is listening to us, we can send K62=0 to switch off RS-485 and resume normal communications.

9 Direct Mode

All commands are sent to the motor via the motors serial port. The motor accepts commands in TTL level serial ASCII. This means that these commands can be sent from a computer running Control Room, or even a generic terminal software. Commands can also be sent from other devices that communicate via serial communications including PLCs and HMIs.

When controlling the motor in direct motion mode, position, speed, and acceleration must be predefined. A command is then sent to execute a motion based on these parameters. For example:

S.1=50

A.1=10

P.1= 1000

M.1=80

^.1

The number after the period indicates which motor on the daisy chain network you wish to address. For more information on connecting multiple motors, see *Daisy Chain Wiring*.

In the example above, each line is defining a different motion parameter:

S.1 defines the speed for motor 1

A.1 defines the acceleration for motor 1

P.1 defines the target position for motor 1

M.1 defines the torque for motor 1

^.1 is the run command, which commands the motor to run using the defined parameters.

The values of S, A and P that are currently stored in the motor can be queried by sending:

?.1

Motor 1 would then reply with all of the stored values.

9.1 Direct Mode Commands

There are a number of commands which can be used to have the motor perform certain functions in direct mode. These commands can be sent manually using the serial communications, or called from within a logic bank.

P0, S0, A0, M0	Dynamic Motion
The zero location in each motion register allows you to set the current values of target position, speed, acceleration, and torque. S0, M0, and A0 may be changed while the motor is running. P0 will change the target for the next move initiated with the execute command.	

S0.1=250
A0.1=100
M0.1=80
P0.1=10000
	Sets the motor 1 target position to 10000 with a speed of 250, acceleration of 100Kpps^2, and torque of 80%.
	These values will be used when the next move is executed, unless overwritten

^	Execute Direct Motion
The caret command will execute a predefined dynamic motion. The movement will utilize the parameters stored in the respective 0 registers (i.e. P0, S0, A0, M0).	
^.1	Motor 1 begins a motion using the values programmed in P0, S0, A0, and M0.

^#	Execute Direct Motion (# = 1-8)
The caret followed by a number will execute a motion using the parameters specified in the corresponding number registers. Valid entries are from 1 to 8.	
^5.1	Motor 1 begins a motion using the values programmed in P5, S5, A5, and M5.

~	Continuous Point Motion
The tilde command will increment the current position by the value of P0. ⚠ For continuous point motion, P0 is assigned at the full resolution of 50 000 pulses per revolution, regardless of the resolution setting in K37	
P0.1=10000 ~.1	For each time the ~ character is sent, the motor will increment the current position by 10000 pulses, or 1/5 of a full rotation.
|	Origin Search
The vertical bar command will begin an origin search routine according to the current origin search settings. ⚠ This is a pipe, or vertical bar, ASCII 124 (7Ch). Not the upper case letter I.	
|.1	Motor 1 begins an origin search according to the current settings.
|1	Move to position 0 (origin)
The bar 1 command makes the motor move to the current origin position (position 0). The speed, acceleration and deceleration values are set by parameters K42, and K43	
|1.2	Motor 2 will begin a move to its respective origin position.
|2	Assign Current Position as Origin
The bar 2 command will assign whatever the motors current position is, as the origin (position 0). For example, is the motor has rotated a few times and the current position is 17250, when you sent the |2 command this position changes to become 0.	

|2.3	The current position of motor 3 becomes 0.

|4	Soft Reset
The bar 4 command will execute a soft reset of the motor. This means that the motor will restart as if being re-powered, but without physically removing power from the motor.	
|4.1	Restart motor 1.

|11	Clear Revolution Counter
Clears the position of the motor, but leaves position information from under one revolution. For example, if the motor resolution is set to 1000 pulses per revolution and we are at position 23764, this means we've travelled 23 revolutions and 764 pulses. When we send the |11 command, the position becomes 764, as the previous 23 revolutions are cleared.	
|11.1	Clear motor 1 revolution counter.

(	Enable Motor
The open bracket will enable the motor. If the motor is set to power up disabled in K46, if the motor has previously been disabled due to an error, or if the motor was manually disabled through an input or command the open bracket will re energize the motor and return it to the ready state.	
(.1	Enable motor 1.

)	Disable Motor
The close bracket will disable the motor. When the motor is disabled, the shaft will turn freely but the position will still be tracked.	

).2	Disable motor 2.
O#	Output Signal On
The letter O followed by an output number will activate the respective output. The output must be programmed as either 3 or 4 in parameter K34.	
O1.2	Activate output 1 on motor 2.
F#	Output Signal Off
The letter F followed by an output number will deactivate the respective output. The output must be programmed as either 3 or 4 in parameter K34.	
F1.2	Deactivate output 1 on motor 2.
$	Save Data
The dollar symbol will save program data from the motors RAM to non-volatile EEPROM memory. All registers, program banks, and logic banks are saved using this command. When data is successfully saved, a message is returned which reads "Saved.n" where n is the motor ID number.Once saved, all data is maintained when power is removed from the motor.	
$.1	Save all data on motor 1.
[	Execute Bank
The open square bracket command will execute the requested program bank. The bracket is followed by the program bank number you wish to execute. Likewise, by following the bracket by a letter L and the bank number, a desired logic bank can be executed as well.	

[1.1	Execute program bank 1 on motor 1.
[L1.1	Execute Logic bank 1 on motor 1.

]	Pause Bank

The close square bracket command will stop a motor immediately and pause a program bank which is currently running. By sending the [command, the bank can be resumed. By sending the] command twice, the bank will be stopped and cannot be resumed. Logic banks cannot be paused and resumed in this manner; sending the bracket followed by an L will stop the currently running logic bank.

].1	Pause program bank on motor 1.
].1	Terminate program bank on motor 1.
].1,].1	Terminate program bank on motor 1.
]L.1	Terminate Logic bank on motor 1.

}	Pause Bank After Current Line

The close brace command will perform much the same as the close square bracket, however the motor will wait until the current line in the program is completed before pausing the bank. a close square bracket must still be used to terminate the program completely.

}.1	Pause program bank on motor 1 after the current line.
}.1	A subsequent } will terminate the bank.

>	Execute Next Program Line
The forward chevron will allow you to step through a paused program. If you pause a program with the] or } commands, you can then step to the next line using the >. The program will automatically pause again after the line is executed. ⚠ After executing the last line of a program bank, the motor will reply with "End!" and no further stepping may be done.	
].1 >.1 >.1	Pause program bank on motor 1. Step one line forward in the program. Step one line forward in the program.

<	Execute Previous Program Line
The backward chevron will allow you to step through a paused program. If you pause a program with the] or } commands, you can then step to the previous line using the <. The program will automatically pause again after the line is executed. ⚠ You may only step back one line. You can not step back a line if the program has ended and the "End!" has been sent from the motor. When you can not step back, the motor will reply with the message "Can't Back!".	
].1 <.1	Pause program bank on motor 1. Step one line backward in the program.

*	Emergency Stop
The asterix will immediately commands all motors on a daisy chain to stop movement with the maximum deceleration. All programs will cease and the motor will not respond to any commands to move until the emergency stop state is removed.	
*	Command all motors to perform an emergency stop.

*1	Cancel Emergency Stop
The asterix followed by a number one will cancel the emergency stop state in a motor. When the emergency stop is cancelled, the motor will return to the ready state, but any programs or movements that were running when the emergency stop was engaged will not automatically restart.	
*1	Cancel emergency stop.

?	Query
The question mark will query the motor for various parameters and values, depending on the number following the question mark.	
?	Current values of P0, A0 and S0
?x	All values in x register (P,S,A,etc.)
?0-16	Program Bank
?1000	All program and Logic Banks
?51	OUT1 Status
?52	OUT2 Status
?70	Input Status
?71	Drive Temperature (°C)
?74	Analog Input Value
?76	Counter Value
?79	Push mode Timer

?85	Motor ID and Firmware Version
?90	All K Parameters
?91	All Position Data
?92	All Speed Data
?93	All Acceleration Data
?94	All Timer Data
?95	Current Position Error
?96	Current Position
?97	Current Speed
?98	Current Average Motor Current
?99	Current Motor Status

10 Program Mode

In Program mode, positions, speed, and accelerations are called and executed inside predefined programs called program banks and logic banks. The logic and program bank systems are independent, but can offer enormous flexibility when used together. Advantages and use cases for each bank type will be discussed in the following sections.

Banks allow for intelligent operation of the cool muscle motor. The banks can be programmed and stored in the motors memory and executed in a number of ways. Programming the motor in this way eliminates the need for an external control mechanism, and as such improves the performance and timing of the system.

The Cool Muscle can store up to 30 different program banks, and 30 different logic banks, but there can only be a combined total of 200 program steps. You can check the current number of program steps by sending the motor a ?1000 command which will reply with all of the programming in the motor as well as the number of steps currently used.

10.1 CML

The cool muscle motors are programmed using a language called Cool Muscle Language, or CML. Both logic and program banks have access to all CML commands, registers, and states.

10.1.1 Syntax

Syntax in logic and program banks is similar. If there is an error in syntax, the bank will cease to load from that point on. If you read back the bank from the motor, you can compare this to what is expected to assist in pinpointing any syntax errors. It is always good practice to read back the programmed banks in a motor using the ?1000 command to verify that everything has loaded correctly.

Start and End

Each bank must begin with the start command, B# for program banks, and L# for logic banks where # is the bank number. Each bank must be ended by the END command. Anything between these statements is taken to be a part of that bank.

Separating Commands

All commands on the same line are grouped; these may be part of an equation or branch operation. Commands to be called consecutively should be placed on a new line, separated by a carriage return. You can place some consecutive commands on the same line, separated by a comma but the maximum number of characters allowed before a carriage return is 40.

10.2 Branch Operations

All Branch Operations follow the same format:

QUERY, TRUE, FALSE

The first statement is the query which can be checking an input, comparing a variable, etc. The next statement is what you want the program to do if the query is true, and finally the last part is what you want the program to do if the query is false.The query can be any of the following operations. For example

V1>=V2, ?99, ?96

If V1 is greater or equal than V2 then send ?99 query else send ?96 query

I	Input
If an input is used alone, it is not necessary to compare, as it is implied that the statement is true if the input is true	
I3,W0,T0	If input three is active, wait, else do nothing and move to next line.

&&	And
The And query is true if both operands are true.	
I1.1 && I2.1, C2.1, W0	If inputs one and two are both active, call bank two, else wait.

||	Or
This query is true if either operand is true.	
I1.1 || I2.1, C2.1, W0	If either input one or two are active, call bank two, else wait.

!!	Not
This query is true if the operand is 0.	
V1 = !! I1.1	If input one is active, V1 is 0. If input one is deactivated, V1 is 1.

>	Greater Than
This query is true if the first operand is greater than the second.	
V1>V2, C2, T0	If variable one is greater than variable two, call bank 2, else do nothing.

>=	Greater Than or Equal To
This query is true if the first operand is greater than or equal to the second.	
V1 >= V2, C2, T0	If variable one is greater than or equal to variable two, call bank 2, else do nothing.

==	Equal To
This query is true if the first operand is equal to the second.	
P1 == V1, J2, T0	If position one is equal to variable one, jump to bank 2, else do nothing.

<	Less Than
This query is true if the first operand is less than the second.	
V1<V2, C2, T0	If variable one is less than variable two, call bank 2, else do nothing.

<=	Less Than or Equal To
This query is true if the first operand is less than or equal to the second.	
V1<=V2, C2, T0	If variable one is less than or equal to variable two, call bank 2, else do nothing.

10.3 Registers

All registers should be set or initialized before they are called in a bank. A register is set by sending the register and register number you with to set followed by and equals sign and the value you wish to set, e.g.: P1.1=100, S10.1=500, V1.2=-50, V2.1=”Ux”.

It is possible to set a register inside a bank in CML, but it cannot be set directly with a number. Any register set inside of a bank must be set from another register:

Correct CML	**Incorrect CML**

`V1.1=100`
 	
`B1.1`
`A1.1=V1.1`
`END.1`
	`B1.1`
	`A1.1=100`
	`END.1`

However in Control Room when using **CMP script** it is possible to assign a constant value directly in a bank. This is because Control Room will compile a CMP script to a CML script. The compiler will assign the constant to an unassigned register and replace it in the CML script before sending to the motor. For example:

CMP Script	**Compiled CML Output**

`B1.1`
`A1.1=100`
`END.1`
	`N1.1=100`
	
	`B1.1`
	`A1.1=N1.1`
	`END.1`

Any register can be queried by sending only the register number, E.g. to query P15.1 simply send:

P15.1

A response will be returned from the motor in the format P15.1=# where # is the value stored in that register.

10.3.1 List of Registers

P	Position
Unit: Pulses (p) P1-P25 Min: -1000000000 Max: 1000000000	P registers define target positions as measured in pulses.
P5.1=1000	Set position 5 in motor 1 to 1000.
S	Speed
Unit: p/s (see *K37*) S1-S15 Min:-5000000 Max: 5000000	S registers define the target speed in a unit defined in parameter *K37* .
S2.1=430	Set speed 2 in motor 1 as 430.
A	Acceleration
Unit: 1000p/s^2 A1-A8 Min:-32767 Max:32767	A registers define the target acceleration in thousands of pulses per second squared.
A8.1=50	Sets acceleration 8 in motor 1 as 50.

T	Timer
Unit: millisecond T1-T8 Min: 0 Max: 32767	T registers define a timer in milliseconds. This timer can be called in program or logic banks to wait for the specified amount of time.
T1.1=25000	Sets timer 1 in motor 1 to 25000 (25s).

M	Torque
Unit: % Peak torque M1-M8 Min: 0 Max: 100	M registers define the torque limit for a move, as measured in a percentage of the motors peak torque.
M2.1=80	Sets the torque register 2 in motor 1 as 80%.

V	Variable Data
Unit: - V1-V15 Min: -2147483648 Max: 2147483647	Variable registers serve a number of purposes. Variable can be set to use current motor states as well as predefined numbers and text strings. When using the motor state variables or text strings, these must be defined using quotations. A comprehensive list and description of the internal state variables can be found under *Internal Variables*.

V2.1=”Px”
V3.1=”abcd”
V8.1=5432
	Sets variable two in motor one as current position.
	Sets variable three in motor one as text string “abcd”.
	Sets variable eight in motor one as 5432.

N	Center Point / General Use
Unit: Pulses N1-N25 Min: -2147483648 Max: 2147483647	N registers are used to define the center point of a circle when using two motors in a coordinated motion system. For more information see *coordinated motion.* Can also be used as general purpose variable if coordinated motion is not used.
N12.1=1375	Sets center point data 12 in motor 1 to 1375.

R	Radius / General Use
Unit: Pulses R1-R25 Min: -1000000000 Max: 1000000000	R registers are used to define the radius of a circle when using two motors in a coordinated motion system. For more information see *coordinated motion.* Can also be used as general purpose variable if coordinated motion is not used.
R5.1=25000	Sets radius data five in motor one to 25000.

10.3.2 Internal State Variables

Internal state variables are special assignments for variables that give the user access to important motor specific data. The following tables list available state variables

Motor Status Variables

Name	Description	R/W	Unit

"Ux"	Status • Read the motor operating status	RO	-

Value	Status Description
0	In motion
4	Over torque error
8	In position
16	Motor disabled

A complete list of status values can be found at Status and Error States

"Px"	Position • Reading this variable will return the motor position • Writing this variable will set the motor position to the assigned value It is recommended a write only occurs when the motor is stationary.	R/W	Pulses
"Sx"	Speed • Read motors speed	RO	Pulses/second

"Ix"	Current • Read motors current The current limit depends on the motor size. The following table shows the range for each motor	RO	-

Motor Size	Range
11S30	±50
11L30	±65
17S30	±60
17L30	±110
23S30	±120
23L30	±180

"Tx"	Target Current • Read motors target current See "Ix" for each motors value range	RO	-
"Ex"	Error Status • Read the value of any error/alarm. This variable differs from "Ux" in that it will only include status values that indicate a motor error/alarm/fault	RO	-
"Pe"	Position Error • Read the error between the target position and the actual position	RO	Pulses

Inputs and Outputs

General Variables

Name	Description	R/W	Unit

"AIN4"	Analog Input Value Read the value of the analog on input 4 Range: 0 to 1023	RO	-
"Fx"	INPUT2 Frequency Read the frequency of the signal on input 2 Range: 0 to 2000	RO	1KHz
"Cx"	INPUT2 High Speed Counter Count rising edge triggers in IN2. This counter uses the processor's high speed counter and will read pulses up to a frequency of 2MHz Write to the counter to set a clear/set a starting value. This counter is very sensitive and will pick up noise and bounce in the signal	R/W	-
"Cx2"	INPUT2 Counter Counts pulses every 1ms on IN2. Useful on counting inputs use with relays or mechanical switches. The internal debounce algorithm eliminates unwanted noise and counts.	R/W	-
"Cx3"	INPUT3 Counter Counts pulses every 1ms on IN3. Useful on counting inputs use with relays or mechanical switches. The internal debounce algorithm eliminates unwanted noise and counts.	R/W	-
"Cx4"	INPUT4 Counter Counts pulses every 1ms on IN4. Useful on counting inputs use with relays or mechanical switches. The internal debounce algorithm eliminates unwanted noise and counts.	R/W	-
"AO2"	Analog Output Value Set the analog value on OUT2. K34 and K34 must be set correctly to use. Range: 0 to 255 (0V to 4.5V)	R/W	-

Digital Inputs

Name	**Description**	**R/W**	**Unit**

"IN"	Input Value
	
	Read the combined value of the inputs. Value is calculated as a binary combination
	
	IN1=1
	IN2=2
	IN3=4
	IN4=8
	
	Example: IN1 and IN4 are ON, IN2 and ON3 are off
	
	IN=9
		RO	-

"INMK"	Input Mask Mask the inputs so only the required inputs return a value in IN	R/W	-
"IN1F" **"IN2F"** **"IN3F"** **"IN4F"**	IN falling edge Read the falling edge for the given input. The variable returns the number of falling edges since the last read. The value is set to 0 on a read.	RO	-
"IN1R" **"IN2R"** **"IN3R"** **"IN4R"**	IN rising edge Read the rising edge for the given input. The variable returns the number of rising edges since the last read. The value is set to 0 on a read.	RO	-

Timers and Counters

Name	**Description**	**R/W**	**Unit**
"Tmr"	1ms count down timer Writing a value into the countdown timer will start the count down. The count down timer will decrement every 1ms until 0. Range: 0 to +32,767	R/W	1ms

"Cnt"	1ms up counter The "Cnt" counter is a free running up counter. The counter can be set to any +ve value by writing to it. It will count up from that value every 1ms. When the counter reaches its limit it resets to 0 and continues counting up. Range: 0 to +2,147,783,647	R/W	1ms

Pushmode Variables

Name	**Description**	**R/W**	**Unit**
"PTmr"	Set the pushmode timer • Set the pushmode time (same as K61) dynamically. • Value is in milliseconds.	R/W	1ms
"PV%"	Set the pushmode value • Set as as a value of peak torque • Can be changed dynamically during a push mode Range: 0 to 800	R/W	0.1% of peak torque
"PVal"	*This variable has been deprecated. It is recommended to use "PV%" in new designs* • Read or write the pushmode target value. This value can be changed during a push • Maximum value is 80% of the current limit. See "Ix" for limits.	R/W	-

Motion Variables

Name	**Description**	**R/W**	**Unit**

"DPos"	Switch on/off dynamic positioning ⚠ This function has been deprecated. It is recommended to use Direct Control for changing the target position during motion. When using the direct mode positioning command (^) the target position is not updated when a new P0 is set. "DPos" sets the behavior of the ^ function	R/W	-

"DPos" Value	**Description**
0	Direct position operates as standard. Writing P0 does not immediately change the target position
1	Writing to P0 will immediately update the target position. If there is no motion ^ must be sent to start motion
2	Writing to P0 will immediately update the target position, If the motor is stationary it will immediately start motion and does not require ^ to start motion.

"MOP"	Writing to "MOP" enables the direct control modes of operation	R/W	-

"MOP" Value	**Description**
0	Off - CML mode (Normal Operation)
2	Profile mode
4	Dynamic position mode
10	Torque mode

See Direct Control in the CM1-T user guide for information on how to use the modes.

The motor controller uses an internal motion generator that calculates the 1ms position target. Writing to the variables will overwrite the next 1ms target. Caution should be taken if it cannot be guaranteed to write every 1ms as this could cause intermittent motion

Name	**Description**	**R/W**	**Unit**
"Pt"	Read the 1ms position target generated by the motion generator	RO	pulses

"Dx"	Set the next position target • Writing to this variable will overwrite the motion generators next position target. • To effectively use this the logic must be able to scan and write to "Dx" every 1ms. It is not recommended to use this variable unless it is guaranteed the value can be updated every 1ms. Unless this value is written every 1ms the internal motion generator will overwrite the value.	R/W	pulses
"St"	Set the next speed target. • This value is set in pulses and is the amount of pulses the motor will increment in the next 1ms. It is not recommended to use this variable unless it is guaranteed the value can be updated every 1ms. Unless this value is written every 1ms the internal motion generator will overwrite the value.	R/W	pulses

Data Retention

Two variables can be used to save data to non-volatile memory. Data will be written on a change.

Name	**Description**	**R/W**	**Unit**
"EEP1"	Memory 1 32 bit	R/W	-
"EEP2"	Memory 2 32 bit	R/W	-

The memory has a limited amount of writes. Caution should be taken to not excessively write to memory

10.4 Math Functions

10.4.1 Basic Math Functions

Both program and logic banks allow for the following mathematical operations:

Operation	Format	Description

=	P1.1=P2.1+V1.1	Equals. Sets P1.1 to the result of the following operation
+	P1.1=P2.1+V1.1	Addition. Sets P1.1 to the sum of P2.1 and V1.1
-	P1.1=P2.1-V1.1	Subtraction. Sets P1.1 to the difference of P2.1 and V1.1
*	P1.1=P2.1*V1.1	Multiplication. Sets P1.1 to the product of P2.1 and V1.1
/	P1.1=P2.1/V1.1	Division. Sets P1.1 to the quotient of P2.1 and V1.1

10.4.2 Advanced Math Functions

In addition to the above basic math functions, there are some advanced math functions available as well:

U1	Sine
This function will calculate the sine of the operand. $U1(x)=\sin(x*2\pi)$ The functions is scaled such that: $$U1(x)=10000\sin((x\div 10000)2\pi)$$	
`B1` `X0` `S1.1=U1(V1.1)` `V1.1=V1.1+1` `X-` `END`	This bank will set speed S1 to the sine of V1. V1 is then incremented. The bank is then looped indefinitely.

U2	Cosine

This function will calculate the cosine of the operand.

$$U2(x)=\cos(x\cdot 2\pi)$$

The functions is scaled such that:

$$U2(x)=10000\cos((x\div 10000)2\pi)$$

```
B1
X0
S1.1=U2(V1.1)
V1.1=V1.1+1
X-
END
```

This bank will set speed S1 to the cosine of V1. V1 is then incremented. The bank is then looped indefinitely.

U3	Square Root

This function will calculate the square root of the operand.

$$U3(x)=\sqrt{x}$$

The value returned by this function is an integer value. For example √7=2. For higher accuracy it is suggested to scale the solution by orders of 10.

```
P1.1=P2.1+U3(V1.1)
```

Position one is equal to position two plus the square root of variable one.

U5	Polynomial Using N

Performs a polynomial calculation on the desired operand using the N register as the polynomial coefficients.

N0 is the polynomial order, to a maximum of 25

N1-N25 are the polynomial coefficients

N0=n

$$U5(x)=(N1)x^{0}+(N2)x^{1}+(N3)x^{2}\ldots+(Nn)x^{(n-1)}$$

`N0=4` `V1.1=U5(P2.1)`	V1.1=(N1)+(N2)(P2.1)+(N3)(P2.1)^2 +(N4)(P2.1)^3
U6	Polynomial Using R
Performs a polynomial calculation on the desired operand using the R register as the polynomial coefficients. R0 is the polynomial order, to a maximum of 25 R1-R25 are the polynomial coefficients R0=n $U6(x)=(R1)x^0+(R2)x^1+(R3)x^2...+(Rn)x^{(n-1)}$	
`N0=4` `V1.1=U5(P2.1)`	V1.1=(R1)+(R2)(P2.1)+(R3)(P2.1)^2 +(R4)(P2.1)^3

10.5 Program Banks

Cool Muscle program banks are the main method to execute a programmable motion in the motor. Program Banks handle the day to day running of the motor for operations that are stored within the motor and implements by the motor independently. Under some circumstances, the execution of motion can be done from a logic bank, it is typically best practice to execute any motion from a program bank.

10.5.1 Usage – When and why

Program banks will call position (P), speed (S), acceleration (A), and other registers. The registers are called within the banks but are defined before the program bank is run.

Program banks will run through all of the commands in the bank only once unless a loop command is used. This is contrary to a logic bank which will continue to scan through the bank unless stopped. A program bank will also execute each command one at a time and will wait for the previous command to be completed before moving to the next line. This is important when the motor is moving to a position, as the next line will not be executed until that position has been reached.

Program banks make it easier to run motion commands, as by simply calling a speed, acceleration, and position value in the manner of:

A1,S1,P1

Will execute a move command in the motor, and we can be sure that the motion has been completed before we execute another command. This is much more difficult to try and do using a logic bank.

For these reasons, program banks are the most common form of program used in the cool muscle motor and are typically the core of any application.

10.5.2 Program Bank Commands

B#	Program Bank Start
This command marks the beginning of the program bank. # represents the program bank number.	

`B1`
`A1,S1,P1`
`END`
	Program bank one contains a single move and then ends.

P#	Move to Position
This will begin a move to the position located in the # location of the P register. The program will move to the next line when the target position is reached. This can also be used as an incremental move by appending with a +. This will increment whatever the current position is by the value of the specified P register.	
`P4.1` `P4.1+`	Begin move to Position stored in P4.1 using last loaded speed, acceleration, and torque. Increment the current position by the value of P4.1
S#	Set Speed
Call the value located in the # location of the S register. This will set the speed for the next called move.	
`S6.1`	Set the speed for the next executed move to the value of S6.1
A#	Set Acceleration
Call the value located in the # location of the A register. This will set the acceleration for the next called move.	
`A1.1`	Set the acceleration for the next executed move to the value of A1.1

M#	Set Torque
Call the value located in the # location of the M register. This will set the torque for the next called move.	
`M8.1`	Set the torque for the next executed move to the value of M8.1
X# / X-	Loop
The X# command begins a loop in a program bank. The following lines in the program will loop # number of times. For an infinite loop, # will be 0. To mark the end of the looped section, use X-.	

`X4.1`
`A1.1,S1.1,P1.1`
`P2.1`
`X-`
	Move motor 1 to position 1 at speed 1 and acceleration 1. Then move to position 2 with the same speed and acceleration. Execute these two moves 4 times and then move on.

Y#	P Without Wait
Execute the same as a P# command, but immediately move to the next line of the program and do not wait until the target position is reached.	
`Y4.1`	Begin move to P4.1 and immediately continue to the next line in the program.
Q#	Push Move
Execute a *push move* to position P#.	
`Q4.1`	Begin push move to position stored in P4.1. Move to the next program line when the push mode timer expires.

Z#	Q Without Wait
Execute a push move to position P#, proceed immediately to the next program line.	
`Z4.1`	Begin a push move to position stored in P4.1 and immediately continue to the next program line.

,	Command Concatenation / Simultaneous Motion
When multiple commands are listed on a single line, they must be separated by a comma. This also allows for multiple motors to be issued move commands at the same time. When issuing commands to multiple motors, they should be sent in descending order, e.g. motor 2 than motor 1.	

`A1,S1,P1`
 	
`P1.2,P3.1`
	Move to P1 with acceleration A1 and speed S1.
	
	Motor two moves to P1 and motor one moved to P3 at the same time.

;	Merge Motion
This command will merge two separate motions together, allowing for a smooth transition without stopping at the first target position. NOTE: This command is different if used within a logic bank. See Logic Bank Commands.	

`B1`
`A1,S1,P1;`
`A2,S2,P2`
`END`
	Motion one will merge into motion two, without pausing when target position one is reached.

C#	Call Program Bank
Call another program bank. Once the called bank is complete, the first program bank will continue. # represents the program bank number to be called.	

`B1`
`C2`
`A2,S2,P2`
`END`
 	
`B2`
`A1,S1,P1`
`END`
	Bank one calls bank two, where it will perform the first move. Once the move in bank two has been completed, bank one will continue to execute the second move, and then end.

J#	Jump to Program Bank
Jump to another program bank. We will not return to the first bank once we have jumped away. # represents the program bank number to be jumped to.	

`B1`
`I1,J2,T0`
`A2,S2,P2`
`END`
 	
`B2`
`A1,S1,P1`
`END`
	In this example, if input one is true we jump to bank two. If input one is not true, we do nothing. In this way the motor will execute one of the two motions depending on the status of input one.

B100	Clear all Program Banks
This command will erase all data in all program banks. Registers and Logic banks are not affected. It is good practice to begin most programs with this command to ensure you are starting with a clean slate.	

`B100`
 	
`B1`
`A1,S1,P1`
`END`
	This example clears all program banks from the motor, and then writes bank one.

For more applications and examples, see Application Notes.

10.6 Bank Commands

I	Input Logic Branch
This command will check whether the specified input is activated or not. Used with I1-I4.	
`I1.1,C2.1,C3.1`	If IN1 is active, call bank 2, else call bank 3. For more info, see Branch Operations.
W0	Wait
This command will cause the program to wait at the current line until the condition is satisfied. Additionally, you can specify a T register and the wait command can be used to wait for that amount of time before moving on.	
`I1.1,C2.1,W0`	If input one is active, call bank 2, else wait. In this example, the program will wait and do nothing until input one is active, at which point it will call bank two.
`I1.1,C2.1,W1`	Complete first motion, wait for amount of time set in T1 and if condition not satisfied, do nothing and continue to the next step in the program.
T0	No-op / Timer
Do nothing. This is a no-operation command. Additionally, you can specify a T register to call and the program will wait the programmed amount of time.	

`I1.1,V1.1=1,T0`	If input one is active, set V1.1 to one, else do nothing.
`I1.1,V1.1=1,T3`	If input one is active, set V1.1 to one, else wait for amount of time programmed in T3 and move to next line.

`B1`
`A1,S1,P1`
`T2`
`A2,S2,P2`
`END`
	Execute motion one, wait for amount of time programmed in T2 and then execute motion two.otion, wait for amount of time set in T1 and if condition not satisfied, do nothing and continue to the next step in the program.

END	End Bank
The END command marks the end of a program or logic bank.	

`B1.1`
`A1,S1,P1`
`END`
	This bank contains one move, and then an END command to mark the end of the program bank.

:	Junction Command
This command will allow you to execute two separate commands at the same time.	
`I1.1,?96:?97,W0`	If input 1 is active, query current position and query current speed, else wait. For more info on query commands, see *query commands*. In this example, the program will wait on this line until input one is active, and then it will return the current position and speed values.

/	Comment
⚠ This comment style is deprecated. It is now recommended to use the comment functions built into Control Room. This command allows you to write comments in program files. Anything between this command and the carriage return is not recognized as commands. Comments are not stored in the motor and are for your reference in the program file only. These are limited to 40 characters. This style commenting is an older legacy comment.	
`/comment for B1` `B1` `A1,S1,P1` `END`	The comment following the / will not be saved in the motor, but everything following the comment will.

10.7 Logic Banks

Cool Muscle logic banks are analogous to ladder logic. Logic banks run concurrently with program banks, allowing complete control in monitoring the changing states of the motor. Logic banks are typically used to monitor states such as position, speed, torque, and input status. Math and branch operations allow for further manipulation and monitoring of states, values, and motion parameters.

10.7.1 Usage – Why and When

In most standard motion applications, a program bank will be enough to accomplish the required functions. Logic banks are typically used when multiple states or inputs need to be monitored, such as performing a specific action when a certain torque or position is reached. These can be run concurrent with a program bank, so while a program bank can be executing motion commands and some other simple functions, the logic bank can be continuously scanning inputs or motor states and executing a command based on that information. A good example of this would be if you have a program bank executing a motion, and a logic bank running which monitors the motor torque and stops or reverses the motion when the torque reaches a certain limit.

Another unique feature of logic banks is that they can be programmed to run as soon as the motor starts up (see *K85*). In this way you can begin scanning the motor state as soon as the motor powers up, or this can even be used to begin execution of a program bank at a certain time without input from a user.

Contrary to program banks, a logic bank will continue to run over and over unless there is a jump to another logic bank, or the stop command is sent.

10.7.2 Logic Bank Commands

L#	Logic Bank Start
This command marks the beginning of the logic bank. # represents the logic bank number.	

`L1`
`I2,V1.1=1,T0`
`END`
	Logic bank one scans input 2 repeatedly.

;	Echo Off
This command will prevent the result of a mathematical operation from being echoed to the serial port. NOTE: This command is different if used within a program bank. See Program Bank Commands.	

`L1`
`V1=V2+V3`
`END`
 	
`L1`
`V1=V2+V3;`
`END`
	In this example, the result of V2 + V3 will be echoed to the serial port when the operation is run.
	
	In this example, nothing is sent to the serial port when this operation is run.

CL#	Call Logic Bank
Call another logic bank. Once the called bank is complete, the first program bank will continue. # represents the logic bank number to be called.	

`L1`
`I2,CL2,T0`
`END`
 	
`L2`
`V1.1=1`
`END`
	If input two is activated, logic bank one will call logic bank two to set V1.1 and then continue to scan input two.

JL#	Jump to Logic Bank
Jump to another logic bank. We will not return to the first bank once we have jumped away. # represents the logic bank number to be jumped to.	

`L1`
`I1,J2,T0`
`END`
 	
`L2`
`I3,V1=1,T0`
`END`
	In this example, if input one is true we jump to bank two. If input one is not true, we do nothing.

L100	Clear All Logic Banks
This command will clear all data in all logic banks. Registers and program banks are not affected. It is good practice to begin most programs with this command to ensure you are starting with a clean slate.	

`L100`
 	
`L1`
`I2,V1=1,T0`
`END`
	This example will clear all logic banks and then create logic bank one.

For applications and examples, see Application Notes.

11 K Parameters

11.1 Introduction

K parameters are setup parameters. They are used to setup a number of functions such as inputs, outputs, motor resolution, s-curve, baud rate and alarm tolerances.

K14	Power up Delay	Unit: ms
Set a delay for the power up of the motor. If K14 is set to 0, as soon as power is applied to the motor, the power up sequence begins. If you need to delay this so that the motor powers up after other equipment, you can program up to a 32s delay. Min: 0 Max: 32000 Default: 0		

K20	Baud Rate	Unit: bits/s
Sets the baud rate for the main communications of the motor. There are four available baud rates. Additionally, K20 allows you to set the motor to communicate using Modbus. Values 0-3 are for standard serial communications, values 10-13 set Modbus mode. Valid Entries: 0:38400 1:9600 2:19200 3:57600 10:38400 Modbus 11:9600 Modbus 12:19200 Modbus 13:57600 Modbus Default: 0		

K21	Semi / Full Closed Loop Operation	Unit: 0.1 degree

Allows you to set an area around the target position in which the motor will revert to an open loop mode. If the current position leaves this area, the motor will resume closed loop operation and attempt to return to the target position. This has the effect of reducing any slight servoing or vibrations as the motor attempts to hold the target position

0 = full closed loop

1 – 36 = angle in 0.1deg

Default: 0

K22	Time Delay for Semi Closed Loop Operation	Unit: ms

Sets the time delay between when the target position is reached, and when the motor goes in to open loop mode, if K21 is set to use semi-open loop mode.

Min: 10

Max: 1000

Default: 0

K22	Time Delay for Semi Closed Loop Operation	Unit: ms

Sets the time delay between when the target position is reached, and when the motor goes in to open loop mode, if K21 is set to use semi-open loop mode.

Min: 10

Max: 1000

Default: 0

K23	Event Status	Unit: -

Allows certain events to be communicated automatically by the motor on the serial port. These options can be combined; for example, setting K23=2 will make the motor communicate only when the input states are changed, but setting K23=6 will communicate both the input and output status.

Valid Entries:

0: No Status

1: All Alarm and Status Codes (*See Motor Status and Error States*)

2: Input Status

4: Output Status

8: Disable Echo

16: Enable Warning and Messages (*See Motor Status and Error States*)

32: Merge Motion Event *(See merge motion)*

Default: 1

K24	Quadrature Output Interval	Unit: pulses

Sets the output interval, or pulse width, of the quadrature encoder output. For more information see *quadrature encoder output*.

Min: 4

Max: 32767

Default: 1000

K25	Time Delay for Slow Signal Response	Unit: 0.1s

Sets the time delay for the slow response input functions for each input. K25 consists of four digits, one for each input in the following format:

K25 = $N_4N_3N_2N_1$

N_4= Input 4 Time

N_3= Input 3 Time

N_2= Input 2 Time

N_1= Input 1 Time

For example, a value of 3333 will provide a 0.3s delay for the slow signal activation on each input.

For more information, see *Input Activation*.

Min: 1

Max: 9

Default: 3333

K26	Invert Input Signal	Unit: -

Invert the operation of the input operation.

The format is the same as K25.

0 = Normal Operation

1 = Inverted

Default: 0000

K27	Input Function at Quick Response Logical High	Unit: -

Sets the function of the input at the quick response logical high function. This function will trigger within 1ms of the input being active. For more information on the individual functions, see *Input Functions*.

Each input is set individually in the following format:

K27 = $N_4N_3N_2N_1$

N_4= Input 4 function

N_3= Input 3 function

N_2= Input 2 function

N_1= Input 1 function

Default: 0000

0: No Action

1: General Use

2: Origin Sensor

3: Manual Feed CW

4: Manual Feed CCW

5: N/A

6: CW Limit/Origin Switch

7: Emergency Stop

8: Full Stop

9: CCW Limit/Origin Switch

K28	Input Function at Quick Response Rising Edge	Unit: -

Sets the function of the input at the quick response rising edge.

The format is the same as K27.

Default: 0000

0: No Action

1: Alarm Reset / Pause

2: Disable Motor

3: Reset Position Counter

4: Execute Next Step

5: Execute Previous Step

6: Run Program Bank 1

7: Begin Origin Search

8: Jog CW or Execute Program Bank 2 (See K36)

9: Jog CCW or Execute Program Bank 3 (See K36)

K29 Input Function at Quick Response Falling Edge

Unit: -

Sets the function of the input at the quick response falling edge.

The format is the same as K27.

Default: 0000

0: No Action

1: Alarm Reset / Pause

2: Enable Motor

3: Reset Counter

4: Execute Next Step

5: Execute Previous Step

6: Run Program Bank 1

7: Begin Origin Search

8: Jog CW or Execute Program Bank 2 (See K36)

9: Jog CCW or Execute Program Bank 3 (See K36)

K30	Input Function at Slow Response Logical High	Unit: -

Sets the function of the input at the slow response logical high.

The format is the same as K27.

Default: 0000

0: No Action

1: General Use

2: Origin Sensor

3: Manual Feed CW

4: Manual Feed CCW

5: N/A

6: CW Limit/Origin Switch

7: Emergency Stop

8: Full Stop

9: CCW Limit/Origin Switch

K31	Input Function at Slow Response Rising Edge	Unit: -

Sets the function of the input at the slow response rising edge.

The format is the same as K27.

Default: 0000

0: No Action

1: Alarm Reset / Pause

2: Disable Motor

3: Reset Counter

4: Execute Next Step

5: Execute Previous Step

6: Run Program Bank 1

7: Begin Origin Search

8: Jog CW or Execute Program Bank 2 (See K36)

9: Jog CCW or Execute Program Bank 3 (See K36)

K32 Input Function at Slow Response Falling Edge

Unit: -

Sets the function of the input at the slow response rising edge.

The format is the same as K27.

Default: 0000

0: No Action

1: Alarm Reset / Pause

2: Enable Motor

3: Reset Counter

4: Execute Next Step

5: Execute Previous Step

6: Run Program Bank 1

7: Begin Origin Search

8: Jog CW or Execute Program Bank 2 (See K36)

9: Jog CCW or Execute Program Bank 3 (See K36)

K33	Output Logic	Unit: -

Sets the function of the output logic. If the output is programmed a 0 it will be active high. This means that the output level will be floating when the output is triggered and pulled to ground when inactive. If the output is programmed as a 1 it will be active low. This means that the output will be pulled to ground when triggered and floating when inactive.

Each output is set individually in the following format:

K33 = N_2N_1

N_2= Output 2

N_1= Output 1

Default: 11

0: Active High

1: Active Low

K34	Output Function	Unit: -

Sets the function of each output. For more information on the output types, see *Output Functions*.

Each output set individually in the following format:

K34 = N_2N_1

N_2= Output 2 function

N_1= Output 1 function

Default: 21

0: AO2

1: In Position

2: Alarm

3: CML O1/F1

4: CML O2/F2

5: Analog Output

6: Merge Motion

7: Quadrature Output (See *quadrature encoder output* for additional information)

8: Motor Free

9: Push Mode Torque Limit Reached

K35	Analog Output Function	Unit: -

Sets the function of any output programmed as *analog output.*

K35 = N_2N_1

N_2= Output 2 function

N_1= Output 1 function

Default: 21

0: Target Position

1: Target Position x8

2: Current Position

3: Current Position x8

4: Position Error

5: Position Error x8

6: Current Velocity /16

7: Current Velocity /2

8: Motor Current

9: Motor Current x8

K36	Pulse Interface	Unit: -

If the CM1 motor is configured as a P type (pulse interface type), this parameter will configure the operation of the motor to be either step and direction type, or a simple clockwise and counter clockwise type.

If the motor is configured as a standard C type motor, this parameter will set whether additional banks can be triggered from an input. See K28.

Default: 0

0: CW/CCW

1: Step/Direction

2: Enables Bank 2 and 3 activation

K37	Resolution and Speed Unit	Unit: Pulses

Sets the resolution of the motor in pulses per rotation, and the speed unit of the motor in pulses per second. For more information on the speed unit see *Speed*.

Default: 3

Speed Unit (pps)	Motor Resolution (ppr)	
100	0:200 1:400 2:500 3:1000 (default) 4:2000 5:2500 6:5000 7:10000 8:25000 10:50000	40:300 42:600 43:800 44:1200 45:1500 46:3000 47:4000 48:6000 49:8000 50:12000
10	20:200 21:400 22:500 23:1000 24:2000 25:2500 26:5000 27:10000 28:25000 30:50000	60:300 62:600 63:800 64:1200 65:1500 66:3000 67:4000 68:6000 69:8000 70:12000
1	100: 50000	

K38	Analog Interface	Unit: -

Determines the function of the analog input if it the motor is set to analog control only (K64=9). For more information on the analog input functions, see *Analog Input.*

Default: 1

0: Speed Control

1: Position Control

K39	Voltage Filter Gain	Unit: 5 rad/s

Set the cut off frequency for the analog input. Use this to reduce noise on the analog input.

Default: 128

Min: 0

Max: 1024

K40	Analog Control Speed Limit	Unit: RPM

When using the analog input control (K64=9) set to speed control (K38=0), this sets the maximum speed of the motor when the analog input voltage is at 4.8VDC.

Default: 200

Min: 0

Max: 3000

Max speed is dependent on the specifications of your particular motor model.

K41	Analog Control Travel Limit	Unit: Pulses

When using the analog input control (K64=9) set to position control (K38=1), this sets the maximum travel range of the motor. When the motor powers up, the initial position is 0. The motor will move between position 0 and the maximum position value by increasing the analog input voltage between 0.2VDC and 4.8VDC respectively.

Default: 2000

Min: -32767

Max: 32767

K42	Origin Search Speed	Unit: 100pps

Sets the speed of the motor in 100 pulses per second, any time an origin search is performed.

Default: 10

Min: 1

Max: 5000

K43	Origin Search Acceleration	Unit: Kpps^2

Sets the acceleration of the motor in 1000 pulses per second squared, any time an origin search is performed. This acceleration is also used for the manual feed function.

Default: 100

Min: 1

Max: 5000

K44	Deceleration Ratio	Unit: %

Sets the deceleration as a percentage of the acceleration of the current move. This ratio will apply to the deceleration of all moves.

Default: 100

Min: 10

Max: 500

K45	Origin Search Direction	Unit: -

Sets the direction for the origin search. You are also able to change the direction that is considered "positive" by the motor.

Default: 1

0: Clockwise

1: Counterclockwise

2: Clockwise with reverse coordinates

3: Counterclockwise with reverse coordinates

For example: if K45=0, the motor will run clockwise until it finds the origin. At this point, any positive positions will be clockwise from the origin. If K45=2, the origin search will still run clockwise, but any positive positions will then be counterclockwise of the origin.

K46	Origin Search Method	Unit: -

Sets the method by which the motor will search for the origin. This parameter also allows you to set the motor to power up with the motor disabled, or free.

Default: 0

0: Hard Stop

1: Hard Stop Search Immediately on Power Up

2: Origin Switch

3: Origin Switch Search Immediately on Power Up

16: Hard Stop and Power Up with Motor Disabled

18: Origin Switch and Power up with Motor Disabled

K47	Origin Stopper Torque	Unit: %

Sets the percentage of the total motor torque which is required to detect a hard stop.

Default: 30

Min: 10

Max: 100

K48	Origin Offset Distance	Unit: 100 Pulses

Allows you to set an offset from the mechanical origin for where you want the motors position 0 to be.

Default: 0

Min: -32767

Max: 32767

K49	Manual Feed Speed	Unit: 100 Pulses

Sets the speed for a Manual Feed motion if programmed in K27.

Default: 30

Min: 10

Max: 100

K50	Manual Jog Travel Distance	Unit: Pulses

Sets the distance in pulses for a Manual Jog move if programmed in K28.

Default: 10

Min: 10

Max: 100

K52	I/O 1&2 Digital or Serial	Unit: -

Since inputs and outputs 1 and 2 can be used as either a digital I/O or for serial communications, this sets the behaviour of these I/O.

K52 = N_2N_1

N_2= I/O2 function

N_1= I/O1 function

Default: 00

0: Auto Detect

1: Force Serial Communications

2: Force Digital I/O

IN/OUT1 cannot be forced to digital

K53	Brake Output	Unit: -

Allows you to select a motor output to be used as a brake. This output will switch any time the motor is disabled/enabled. This included being manually disabled, or being disabled through an error or alarm.

In order to set an output as the Brake, the output must be set as 0 in K34. You may then enter the output number you wish to be a brake output in K53. If the output has a setting in K34, then K53 will be ignored.

e.g.

Making output 2 a brake output:

K34=01

K53=2

If you wish to invert the output, simply make the output number negative:

K53=-2

Default: 0

Min: -6

Max: 6

K54	Quadrature Output Offset	Unit: pulses

Sets an offset from the motors 0 position for the first pulse on the quadrature output.

See *quadrature encoder output* for additional information.

Default: 0

Min: 0

Max: 32767

K55	In Position Tolerance	Unit: pulses

Sets the position tolerance for when the motor will send the in position signal, and move to the next step in the program if applicable. For example, if you tell the motor to move to position 1000 and K55=10, the motor will send the in position signal when the current position reaches 990. If the motor stops on this position however, it will continue to try and hold the position at 1000, not 990.

Default: 5

Min: 1

Max: 100

K56	Position Error Overflow Alarm	Unit: 1000 pulses

Sets the maximum value for the position overflow error. If the position error reaches above the value set in K56 in thousands of pulses, the motor will generate a position overflow error and will enter the disabled state.

Default: 50

Min: 1

Max: 32767

K57	Overload Alarm Delay	Unit: ms

Sets the time delay between when an overload condition is detected, and when the motor faults and becomes disabled. An overload condition is any time the motors current/torque exceeds the rated maximum torque.

Default: 3000

Min: 100

Max: 10000

K58	+ Software Position Limit	Unit: 100 pulses

Sets the positive position limit for the motor. When the motor reaches the programmed position it will stop. In a program bank, any motion that would instruct the motor to go beyond this limit is instead ended at the limit and the program will then move to the next step. A value of 0 will disable this limit.

Default: 0

Min: 0 (off)

Max: 32767

K59	- Software Position Limit	Unit: 100 pulses

Sets the negative position limit for the motor. When the motor reaches the programmed position it will stop. In a program bank, any motion that would instruct the motor to go beyond this limit is instead ended at the limit and the program will then move to the next step. A value of 0 will disable this limit.

Default: 0

Min: 0 (off)

Max: -32767

K60	Pushmode Current Limit	Unit: %

This is the amount of torque used when running a push move. This torque is entered as a percentage of 80% of max torque. For example, K60=30 will cause the push move to use 30% of 80% max torque, or 24%.

Default: 0

Min: 10

Max: 80

K61	Push Mode Holding Time	Unit: ms

Sets the length of time to push for when using a push move. If a time of 3001 is entered, this will result in an indefinite push time.

Default: 200

Min: 10

Max: 3001

K62	RS-485 Node ID	Unit: -

Allows you to set the motor in to RS-485 mode and set the node ID. When using MODBUS mode, set K65 first. For more information, see *RS-485*.

Default: 0

0: RS-232 Mode

1...256: RS-485 Node ID

-1...-256: RS-485 Node ID, No auto-report

K63	External Encoder Input	Unit: -

Allows you to set the inputs to accept the output from an external encoder. Also allows you to enable the high speed counter variables Fx and Cx.

Default: 0

0: None

1: Phase A Only

2: Phase A and B

3: Enable “Fx” and “Cx” Counter Variables

K64	Analog Input Function	Unit: -

Sets the function of the analog input. The digital value of the analog input will be applied to the selected register.

Default: 0

0: None

1: NA

2: P0 (Target Position)

3: S13

4: P24

5: S14

6: P25

7: Proportional Speed Control between 0 and current Target Speed

8: Position Multiplier - Scales the current target position based on analog input value as a percentage. For example if the analog input value is 100 the target position will remain unchanged. If the analog input value is 1000 the target position will be 10x greater.

9: Analog Control Only (see K38)

When using the analog input to set the P registers (P0,P24,P25), the calculation is as follows:

Px = Analog Input * K41 / 10

For example, if the analog is set to maximum value of 1024, and K41 is programmed as 500:

Px= 1024 * 500 / 10

Px=51200

When using the analog input to set the S registers (S13,S14), the calculation is as follows:

Sx = Analog Input * (K40 *5/6) *1000 / (pulse factor * speed factor) /1179

Pulse factor is the maximum resolution of the motor divided by the current set resolution. For example, by default the motor resolution is set to 1000 (see K37). This means that the pulse factor will be 50000/1000 = 50.

Speed factor is set in K37 and is either 100, 10, or 1. By default the speed factor is 100.

For example, if the analog is set to the maximum value of 1024, and k40 is programmed as 200:

Sx = 1024*(200*5/6) *1000 / (50 * 100) / 1179

Sx = 28

K65	Slave Motor Baud Rate	Unit: bits/s

Sets the baud rate for communication to other motors downstream on a daisy chain. If setting the baud rate for Modbus mode, set the last motor only.

Default: 0

0: 38400

1: 9600

2: 19200

3: 57600

4: 76800

5: 129000

6: 173000

7: 515000

Modbus Mode:

10: 38400

11: 9600

12: 19200

13: 57600

14: 76800

15: 129000

16: 173000

17: 515000

K66	Data Streaming	Unit: -

When set, this will cause the motor to continually stream out the requested data at the timing programmed in K67. This is useful for obtaining data to analyze a movement.

Default: 0

0: Disable Streaming

1: Target Speed

2: Real Position

3: Real Speed

4: Real Motor Current

5: Real Position in Full 50k Resolution

6: Real Velocity in Full 50k Resolution

For more details, see K66 Parameters

K67	Data Streaming Sample Time	Unit: ms

Sets the data streaming sample time if there is data programmed to be streamed in K66.

Default: 0

Min: 0

Max: 3000

K69	S-Curve Gain	Unit: -

Sets the gain of the S-curve functionality. By setting a higher gain, the motor will attempt to produce a more aggressive S-curve.

Default: 128

Min: 0

Max: 1024

K70	Data Delimiter	Unit: -

Sets the way in which the motor will delimit the end of any replied data.

Default: 1

0: Carriage Return Only

1: Carriage Return and Line Feed

K71	Temperature Alarm Limit	Unit: °C

Sets the temperature alarm limit. When the temperature in the driver case at the back of the motor exceeds the programmed limit, the motor will enter a disabled state and output the “Ux=128” temperature alarm status. This alarm status can be reset once the temperatures is at least 10°C lower than the programmed alarm temperature.

Default: 150

Min: 0

Max: 150

K72	-	N/A
K72 is not implement and has no affect		

K73	Merge Motion Signal Output Length	Unit: ms
Sets the length of time that the merge motion output is active for, if there is an output programmed for the merge motion event in K34. For more Information about merge motion, see *merge motion*. *Default: 10* Min: 0 Max: 1000		

K74	External Torque Feedback P-Gain	Unit: -
Sets the gain of external proportional torque feedback. *Default: 0* Min: 0 Max: 1000		

K75	External Torque Feedback I-Gain	Unit: -

Sets the gain of external integral torque feedback.

Default: 0

Min: 0

Max: 500

K83	Digital Input Filter	Unit: 200us

Allows you to set a scan time delay for the digital inputs. This delay is set in 200us increments. The CM1 will only respond to an input activation when it has been asserted for the entire time of the delay.

For example, if K83=5 the input will need to be activated for 1ms before the motor responds to the input activating. If the input is on for 500us but then turns off again, no input activation will be registered.

Default: 0

Min: 0

Max: 45

K85	Logic Bank to Start at Power Up	Unit: -

Allows you to set the motor to start a particular logic bank when the motor powers up. This way you may have the motor execute a certain program without the need for manual intervention. Only logic banks can be activated in this way, though program banks can be activated from inside logic banks. A value of 0 will disable any automatic starts.

Default: 0

Min: 0

Max: 30

K86	Coordinated Motion Synchronization	Unit: -

When using coordinated motion with multiple motors, turning on synchronization will cause the motor to stream a sync bit to ensure both motors are moving together with perfect timing. This may cause a reduction in the smoothness of motion in the secondary motor and is not recommended for most applications.

Only enable this bit on motor one, as communication issues could arise if more than one motor is trying to send the sync bit.

Default: 0

0: Off

1: On

K87	Logic Bank Scan Time	Unit: ms

Sets the time it takes for the motor to scan through an entire logic bank. This will be the minimum time, as depending on the number of steps in the bank, it could take longer.

Default: 0

Min: 1

Max: 32767

K88	External Encoder Resolution	Unit: pulses

If you are using external encoder feedback, this will program the resolution of the encoder.

Default: 0

Min: 1

Max: 50000

K89	Modbus Input Register Address	Unit: -

Sets the modbus address for the input register.

Default: 640

Min: 0

Max: 65535

K90	Modbus Output Register Address	Unit: pulses

Sets the Modbus address for the output register.

Default: 2048

Min: 0

Max: 65535

11.2 Saving of K Parameters

K parameters are automatically saved to non volatile memory when they are changed. Typically K parameters are only used during setup but occasionally are changed during runtime by an application. If K parameters are repeatedly changed the lifetime of non-volatile memory will be reduced. The _SKH command can be used to switch off automatic saving of K parameters. This command is only available in the command line and is reset on a power cycle.

_SKH Value	**Description**
_SKH=0	K parameters are not saved automatically
_SKH=1	K parameters are saved automatically • Default value

The value can be queried by sending _SKH.

Note: All commands use a carriage return as a terminating character.

11.3 K66 Parameters

K66 Value	Output Values	Streaming	Notes
1	Target Speed	When Running	
2	Current Position	When Running	
3	Current Speed	When Running	
4	Current Torque	When Running	
5	Current Position	When Running	(Always in full resolution)
6	Current Speed (Same as 3?)	When Running	
7	Current Torque	When Running	Tx.1=xxx
8	Motor status, Overload alarm delay	When Running	Overload Alarm Delay=K57/32*5 (Alarm: XX, XX)
9	Current Iq, Iq Target, Current Id, Id Target	When Running	Iq=Torque Generating current Id= Non-torque generating current
10	ADIU0, ADIV0, ADSIN, ADCOS	When Running	Encoder Feedback
11	AnglerealComp, angleReal, Position at full resolution	Always	AngleRealComp = Position Sensor Angle after calculation of sensor phase offset AngleReal = Position Sensor Angle before offset

K66 Value	Output Values	Streaming	Notes
12	SinOffset, CosOffset, Position Offset	Always	SinOffset/CosOffset = Position sensor level offset Position Offset = Encoder Calibration Offset
13	Rotation Count, Distance Target, Distance Real, Position Target, Current Position, Schedule Ready, Distancereal memo, vgenerated	Always	Rotation Count = Current number of full rotations of full resolution Distance Target = Distance Real = Position Target = Target Motor Position Position Real = Current Motor Position Schedule Ready = Ready to schedule new motion Distance Real Memo = Vgenerated = Speed Target
14	vGenerated, VelocityReal	Always	

15	VoltageFilter4, OS_input420, ADAIN0	Always	Voltage Filter 4 = IN4 Analog Input Voltage full resolution followed by motor ID
			OS_Input420 = Filtered Analog Input Value
			ADAIN0 = Filtered Analog Input Value

16	H infinity motor current , Maximum Current Limit	Always	
17	H infinity motor current	Always	
18	Integrator State	Always	
19	Torque.torMeasure.1 Torque.torU.1	Always	
20	Position error	Always	
21	Rolling Average of Torque Squared for Overload detection	Always	
22	Current Torque Squared	Always	Used in Overload Detection
23	Manual Feed Speed	Always	

K66 Value	Output Values	Streaming	Notes
24	External Encoder count	Always	
25	Slow speed display	Always	10s rolling average speed at full resolution
26	Current position at full resolution, CurrentAngleReal	Always	AngleReal = Position Sensor Angle before offset
27	NA	Always	
28	ADCOS, ADSIN, Current AngleReal, Current position at full resolution	Always	
29	TaskCounter, tm4, positionFilterRes	Always	
30	OUT1 status, OUT2 status	Always	

12 Motor Tuning

Cool Muscle motors use an H-Infinity control algorithm, which is fundamentally different than any PID control system many are familiar with. The control algorithm is actively trying to achieve stability at all times. This makes it a very robust and dynamic controller when applied to motion control. The controller is very forgiving giving the user a large envelope to work in while still achieving optimal and stable motion. The controller is formulated using a state-space model where each of the 8 user parameters are elements in a matrix. The matrix as a whole defines the controller and is used to solve the control algorithm. As such there is no direct cross over of the controller gains to a PID controller. However, some parameters will act in a similar fashion even though they are fundamentally different.

Due to the stability of the controller the majority of applications will not require any changing/tuning of parameters. Most often large inertia mismatches (>50:1) or very light loads with accurate slow speeds targets will require tuning. Each of the parameters are discussed below and approaches to common control problems.

Parameter	**Range [min, max]**	**Motor Defaults**					
		11S	**11L**	**17S**	**17L**	**23S**	**23L**
H0	[20 , 100]	100	100	100	100	100	100
H1	[0, 512]	46	46	46	46	46	46
H2	[-5000, 5000]	-2427	-2427	-2427	-2427	-2427	-2427
H3	[0, 5000]	1747	1454	2100	455	376	204
H4	[0, 3000]	127	74	61	54	70	90
H5	[0, 1000]	46	31	20	34	43	65
H6	[0, 1000]	38	20	21	10	13	15
H7	[0, 32767]	98	64	19	8	21	10

Parameter	Description

H0	H0 is an overall gain on the controller. This is set to 100% by default. It will affect the following gains
	H4_CONTROLLER = H4 * H0 / 128
	H5_CONTROLLER = H5 * H0 / 128
	H6_CONTROLLER = H6 * H0 / 128
	H7_CONTROLLER = H7 * H0 / 128
	H0 is commonly used to detune the controller as a whole.

H1	H1 does not have any specific function. Changing H1 will have little effect and should not be changed
H2	H2 does not have any specific function. Changing H2 will have little effect and should not be changed
H3	H3 is a filter in the control system. Increasing H3 will stiffen the controller increasing response time.
H4	H4 acts most similar to a proportional gain
H5	H5 acts most similar to a differential gain
H6	H6 acts most similar to an integral gain
H7	H7 is a filter on the control system. Increasing H7 will make the controller sluggish. It gives the effect of making the system feel more "spongy".

12.1 Common Gain Tuning Examples

12.1.1 Harmonic resonance

When coupled to a long ballscrew a harmonic resonance may be induced. Reduce H0 to detune the system as a whole. H0 should be reduced in increments of 10 until the resonance is eliminated.

- You may notice that that the system response is reduced below acceptable levels. Increasing H4, H5, and H6 can help improve response.

12.1.2 Large inertia mismatch

The default gains will often function well on mismatches all the way to 50:1. Once going beyond 50:1 it may be required to change a few parameters. The following steps can be followed to easily obtain stability.

1. Set H4=200
2. Set H6=9
3. Continue to increase H4 in increments of 50 and decrease H6 in decrements of 2 until the load is stable.

   a. If H4 is required to get very large (>600) a humming sound (resonance) may appear. Reduce H0 slightly to account for this.
   b. H6 can be set as low as 0 but typically you would want to keep this at minimum 1.
4. H3 can be increased slowly if the response of the system is lagging. Often this is not required unless the load is accelerating quite quickly.

12.1.3 Oscillating Load

If the load is coupled to the motor with a flexible helical style coupling there could be oscillations induced. Increasing H3 will stiffen the controller and help stabilize the oscillating.

12.1.4 Slow speed, low ripple, light load

The default gains are quite responsive and will quite obviously react to dynamics in the motor system at lower speeds. This can be accounted for by treating the system as if there is a large inertia mismatch. This will dampen the controller reducing speed ripple. Follow the steps described above in Large inertia mismatch.

13 Push Move

A push move is a special type motion profile usually used with actuators that will allow you to use the motor to push forward at a certain torque for a certain amount of time.

The following parameters are used to set up a push move:

Parameter	Description
K60	Push Mode Current Level
K61	Push Mode Time

The force you want to apply is programmed in K60. This is programmed as a percentage of 80% of the total torque of the motor. The max value for this is 80, this means that the highest force you can apply to a push move is 80% of 80% of the motors full torque.

The push mode time is the amount of time you want the motor to apply this torque for. This is programmed in milliseconds and can go as high as 30000. To program an infinite push time, you can program this as 30001.

To use a push move, it must be programmed in a program bank, using one of the following commands:

Q - Push Move

Z - Push move without wait

14 Modbus-RTU

14.1 Available Registers

14.1.1 Register Table Table

Included below is a table which details the location and function of the CM1 Modbus registers.
The addresses included below are referenced to the Modbus data model (PLC address for read/write to holding registers)

Holding Register Address	Motor Parameter	Read Access	Write Access
Motor Information			
40001	Position Error (?95)	Yes	No
40003	Motor Position (?96)	Yes	No
40005	Motor Speed (?97)	Yes	No
40007	Motor Torque (?98)	Yes	No
40009	Motor Status (?99)	Yes	No
Variables			
40011	V0	Yes	Yes
40013	V1	Yes	Yes
40015	V2	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40017	V3	Yes	Yes
40019	V4	Yes	Yes
40021	V5	Yes	Yes
40023	V6	Yes	Yes
40025	V7	Yes	Yes
40027	V8	Yes	Yes
40029	V9	Yes	Yes
40031	V10	Yes	Yes
40033	V11	Yes	Yes
40035	V12	Yes	Yes
40037	V13	Yes	Yes
40039	V14	Yes	Yes
40041	V15	Yes	Yes
Direct Registers			
40043	P0	Yes	Yes
40045	S0	Yes	Yes
40047	A0	Yes	Yes
40049	V0	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40051	R0	Yes	Yes
40053	M0	Yes	Yes
I/O			
40055	Analog Output	Yes	Yes
40057	Input Status (?70)	Yes	No
40059	Output Status (?50)	Yes	Yes
CML Port			
40103	CML Port [ASCII]	No	Yes
P Registers			
40201	P0	Yes	Yes
40203	P1	Yes	Yes
40205	P2	Yes	Yes
40207	P3	Yes	Yes
40209	P4	Yes	Yes
40211	P5	Yes	Yes
40213	P6	Yes	Yes
40215	P7	Yes	Yes

Holding Register Address	**Motor Parameter**	**Read Access**	**Write Access**
40217	P8	Yes	Yes
40219	P9	Yes	Yes
40221	P10	Yes	Yes
40223	P11	Yes	Yes
40225	P12	Yes	Yes
40227	P13	Yes	Yes
40229	P14	Yes	Yes
40231	P15	Yes	Yes
40233	P16	Yes	Yes
40235	P17	Yes	Yes
40237	P18	Yes	Yes
40239	P19	Yes	Yes
40241	P20	Yes	Yes
40243	P21	Yes	Yes
40245	P22	Yes	Yes
40247	P23	Yes	Yes
40249	P24	Yes	Yes
40251	P25	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
R Registers			
40301	R0	Yes	Yes
40303	R1	Yes	Yes
40305	R2	Yes	Yes
40307	R3	Yes	Yes
40309	R4	Yes	Yes
40311	R5	Yes	Yes
40313	R6	Yes	Yes
40315	R7	Yes	Yes
40317	P8	Yes	Yes
40319	P9	Yes	Yes
40321	R10	Yes	Yes
40323	R11	Yes	Yes
40325	R12	Yes	Yes
40327	R13	Yes	Yes
40329	R14	Yes	Yes
40331	R15	Yes	Yes
40333	R16	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40335	R17	Yes	Yes
40337	R18	Yes	Yes
40339	R19	Yes	Yes
40341	R20	Yes	Yes
40343	R21	Yes	Yes
40345	R22	Yes	Yes
40347	R23	Yes	Yes
40349	R24	Yes	Yes
40351	R25	Yes	Yes
N Registers			
40401	N0	Yes	Yes
40403	N1	Yes	Yes
40405	N2	Yes	Yes
40407	N3	Yes	Yes
40409	N4	Yes	Yes
40411	N5	Yes	Yes
40413	N6	Yes	Yes
40415	N7	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40417	N8	Yes	Yes
40419	N9	Yes	Yes
40421	N10	Yes	Yes
40423	N11	Yes	Yes
40425	N12	Yes	Yes
40427	N13	Yes	Yes
40429	N14	Yes	Yes
40431	N15	Yes	Yes
40433	N16	Yes	Yes
40435	N17	Yes	Yes
40437	N18	Yes	Yes
40439	N19	Yes	Yes
40441	N20	Yes	Yes
40443	N21	Yes	Yes
40445	N22	Yes	Yes
40447	N23	Yes	Yes
40449	N24	Yes	Yes
40451	N25	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
S Registers			
40603	S0	Yes	Yes
40605	S1	Yes	Yes
40607	S2	Yes	Yes
40609	S3	Yes	Yes
40611	S4	Yes	Yes
40613	S5	Yes	Yes
40615	S6	Yes	Yes
40617	S7	Yes	Yes
40619	S8	Yes	Yes
40621	S9	Yes	Yes
40623	S10	Yes	Yes
40625	S11	Yes	Yes
40627	S12	Yes	Yes
40629	S13	Yes	Yes
40631	S14	Yes	Yes
40633	S15	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
A Registers			
40635	A0	Yes	Yes
40637	A1	Yes	Yes
40639	A2	Yes	Yes
40641	A3	Yes	Yes
40643	A4	Yes	Yes
40645	A5	Yes	Yes
40647	A6	Yes	Yes
40649	A7	Yes	Yes
40651	A8	Yes	Yes
M Registers			
40653	M0	Yes	Yes
40655	M1	Yes	Yes
40657	M2	Yes	Yes
40659	M3	Yes	Yes
40661	M4	Yes	Yes
40663	M5	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40665	M6	Yes	Yes
40667	M7	Yes	Yes
40669	M8	Yes	Yes
K Registers			
40671	K0	Yes	Yes
40673	K1	Yes	Yes
40675	K2	Yes	Yes
40677	K3	Yes	Yes
40679	K4	Yes	Yes
40681	K5	Yes	Yes
40683	K6	Yes	Yes
40685	K7	Yes	Yes
40687	K8	Yes	Yes
40689	K9	Yes	Yes
40691	K10	Yes	Yes
40693	K11	Yes	Yes
40695	K12	Yes	Yes
40697	K13	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40699	K14	Yes	Yes
40701	K15	Yes	Yes
40703	K16	Yes	Yes
40705	K17	Yes	Yes
40707	K18	Yes	Yes
40709	K19	Yes	Yes
40711	K20	Yes	Yes
40713	K21	Yes	Yes
40715	K22	Yes	Yes
40717	K23	Yes	Yes
40719	K24	Yes	Yes
40721	K25	Yes	Yes
40723	K26	Yes	Yes
40725	K27	Yes	Yes
40727	K28	Yes	Yes
40729	K29	Yes	Yes
40731	K30	Yes	Yes
40733	K31	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40735	K32	Yes	Yes
40737	K33	Yes	Yes
40739	K34	Yes	Yes
40741	K35	Yes	Yes
40743	K36	Yes	Yes
40745	K37	Yes	Yes
40747	K38	Yes	Yes
40749	K39	Yes	Yes
40751	K40	Yes	Yes
40753	K41	Yes	Yes
40755	K42	Yes	Yes
40757	K43	Yes	Yes
40759	K44	Yes	Yes
40761	K45	Yes	Yes
40763	K46	Yes	Yes
40765	K47	Yes	Yes
40767	K48	Yes	Yes
40769	K49	Yes	Yes

Holding Register Address	**Motor Parameter**	**Read Access**	**Write Access**
40771	K50	Yes	Yes
40773	K51	Yes	Yes
40775	K52	Yes	Yes
40777	K53	Yes	Yes
40779	K54	Yes	Yes
40781	K55	Yes	Yes
40783	K56	Yes	Yes
40785	K57	Yes	Yes
40787	K58	Yes	Yes
40789	K59	Yes	Yes
40791	K60	Yes	Yes
40793	K61	Yes	Yes
40795	K62	Yes	Yes
40797	K63	Yes	Yes
40799	K64	Yes	Yes
40801	K65	Yes	Yes
40803	K66	Yes	Yes
40805	K67	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40807	K68	Yes	Yes
40809	K69	Yes	Yes
40811	K70	Yes	Yes
40813	K71	Yes	Yes
40815	K72	Yes	Yes
40817	K73	Yes	Yes
40819	K74	Yes	Yes
40821	K75	Yes	Yes
40823	K76	Yes	Yes
40825	K77	Yes	Yes
40827	K78	Yes	Yes
40829	K79	Yes	Yes
40831	K80	Yes	Yes
40833	K81	Yes	Yes
40835	K82	Yes	Yes
40837	K83	Yes	Yes
40839	K84	Yes	Yes
40841	K85	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40843	K86	Yes	Yes
40845	K87	Yes	Yes
40847	K88	Yes	Yes
40849	K89	Yes	Yes
H Registers			
40901	H0	Yes	Yes
40903	H1	Yes	Yes
40905	H2	Yes	Yes
40907	H3	Yes	Yes
40909	H4	Yes	Yes
40911	H5	Yes	Yes
40913	H6	Yes	Yes
40915	H7	Yes	Yes
T Registers			
40951	T0	Yes	No
40953	T1	Yes	Yes
40955	T2	Yes	Yes

Holding Register Address	Motor Parameter	Read Access	Write Access
40957	T3	Yes	Yes
40959	T4	Yes	Yes
40961	T5	Yes	Yes
40963	T6	Yes	Yes
40965	T7	Yes	Yes
40967	T8	Yes	Yes

14.2 Getting Started

14.2.1 Introduction

Modbus on the CM1 can be implemented over Modbus RTU or Modbus TCP. The main motor remains the same over both variants but require different interface modules. Modbus is not available on motors with firmware RT3.12 but requires versions RT3.13 or higher.

Modbus TCP

⚠ This Modbus TCP interface has been deprecated. Please see CM1-T Modbus TCP for the latest Modbus TCP version of the CM1.

Modbus TCP requires the Modbus Ethernet module and has the following part numbers for the 4 motor options.

Part Number	Description
CM1-C-17S30-MBT	NEMA 17 single stack CM1 Cool Muscle motor with Modbus TCP
CM1-C-17L30-MBT	NEMA 17 double stack CM1 Cool Muscle motor with Modbus TCP*
CM1-C-23S30-MBT	NEMA 23 single stack CM1 Cool Muscle motor with Modbus TCP*
CM1-C-23L20-MBT	NEMA 23 double stack CM1 Cool Muscle motor with Modbus TCP*

*For torque and speed characteristics please see the CM1 data sheet.

Modbus RTU

Modbus RTU is available on the standard motor as it uses serial as its communication protocol. The standard motor can then be coupled with different interface modules (-SRLM, -SRLS, -EIO) to allow for a wider range of connectivity to a PLC, HMI, PC or embedded controller. It can be ordered with standard motor part numbers but should include the firmware version to ensure the correct version is ordered.

Part Number	**Description**
CM1-C-11S30-RT3.13	NEMA 11 single stack CM1 Cool Muscle motor**
CM1-C-11L30-RT3.13	NEMA 11 double stack CM1 Cool Muscle motor**
CM1-C-17S30-RT3.13	NEMA 17 single stack CM1 Cool Muscle motor with Modbus RTU*
CM1-C-17L30-RT3.13	NEMA 17 double stack CM1 Cool Muscle motor with Modbus RTU*
CM1-C-23S30-RT3.13	NEMA 23 single stack CM1 Cool Muscle motor with Modbus RTU*
CM1-C-23L30-RT3.13	NEMA 23 double stack CM1 Cool Muscle motor with Modbus RTU*

*For torque and speed characteristics please see the CM1 data sheet.

**The 11L and 11S motors have a different packaging and additional interface modules cannot be mounted and integrated directly onto the motor.

14.2.2 Modbus Registers and Usage

All motor parameters are available in read/write access through Modbus holding registers. All registers are 32bit little endian registers. CML code is required in the motor to execute a move. All Modbus TCP motors have a generic point-to-point program written in the motor and this can be modfied to suit the application. As Modbus RTU is a user select-able option it does not come preloaded, however, the program is shown and described further in this documentation and can easily be loaded onto the motor.

The supplied program contains 5 main write variables.

1. Position (P0 register)
2. Speed (S0 register)
3. Acceleration (A0 register)
4. Torque (M0 Register)
5. Control Word (R0 register)

The control word allows the user to start, stop, home, enable and disable the motor. Starting the motor will run the motor to the defined position, speed and acceleration. Position feedback, speed and motor status are available for read through the relevant registers. Please see the supplied code example for a complete description.

14.3 Configuring Modbus

14.3.1 Configuring Modbus TCP

⚠ This following guide is to setup the legacy -MBT addon module. Please see CM1-T User Guide to setup the CM1-T.

The user will want to setup the network settings for the Modbus TCP motor. This could include setting a static/dynamic IP and/or a password.

Requirements

To configure a CM1 with Modbus TCP you should have the following

1. CM1 motor with -MBT module (e.g. CM1-C-23L20-MBT)
2. Control Room (which can be found here)

Configure Network Settings

Two main network settings can be changed

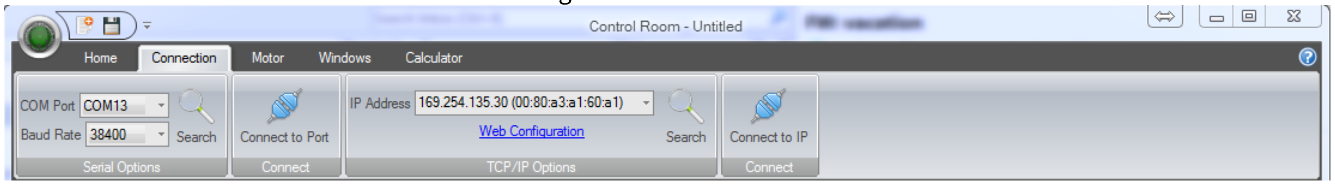
1. Static or dynamic (default) IP address
2. Network password

When you logon to the Modbus TCP web configuration page there are other Modbus related settings. These should be left as they are. If you reset the module to defaults please refer to the Configuring Modbus section at the bottom of this page.

Logon to the Configuration page

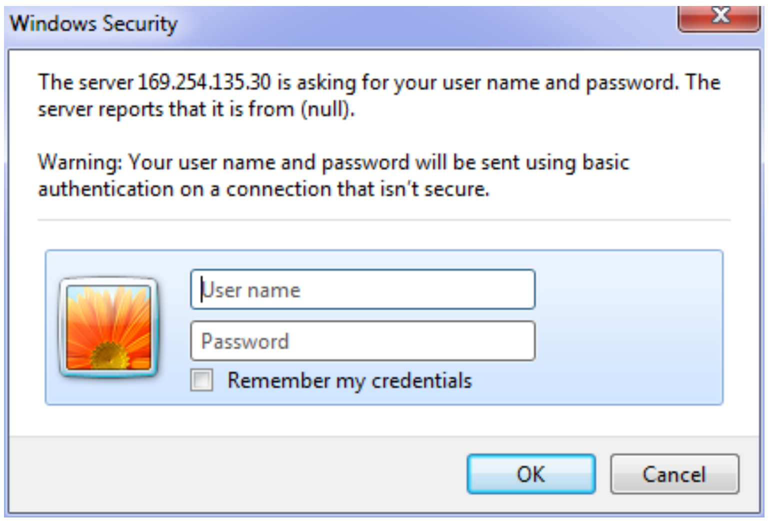
When logging onto the configuration page for the first time there is no password. Follow the steps below to logon.

1. Open Control Room and search for the module under the TCP/IP options
2. Once the module has been found click "Web Configuration"



COM Port COM13
Baud Rate 38400
IP Address 169.254.135.30 (00:80:a3:a1:60:a1)

3. If using for the first time and you haven't set a password click okay when the security window pops up.


Windows Security

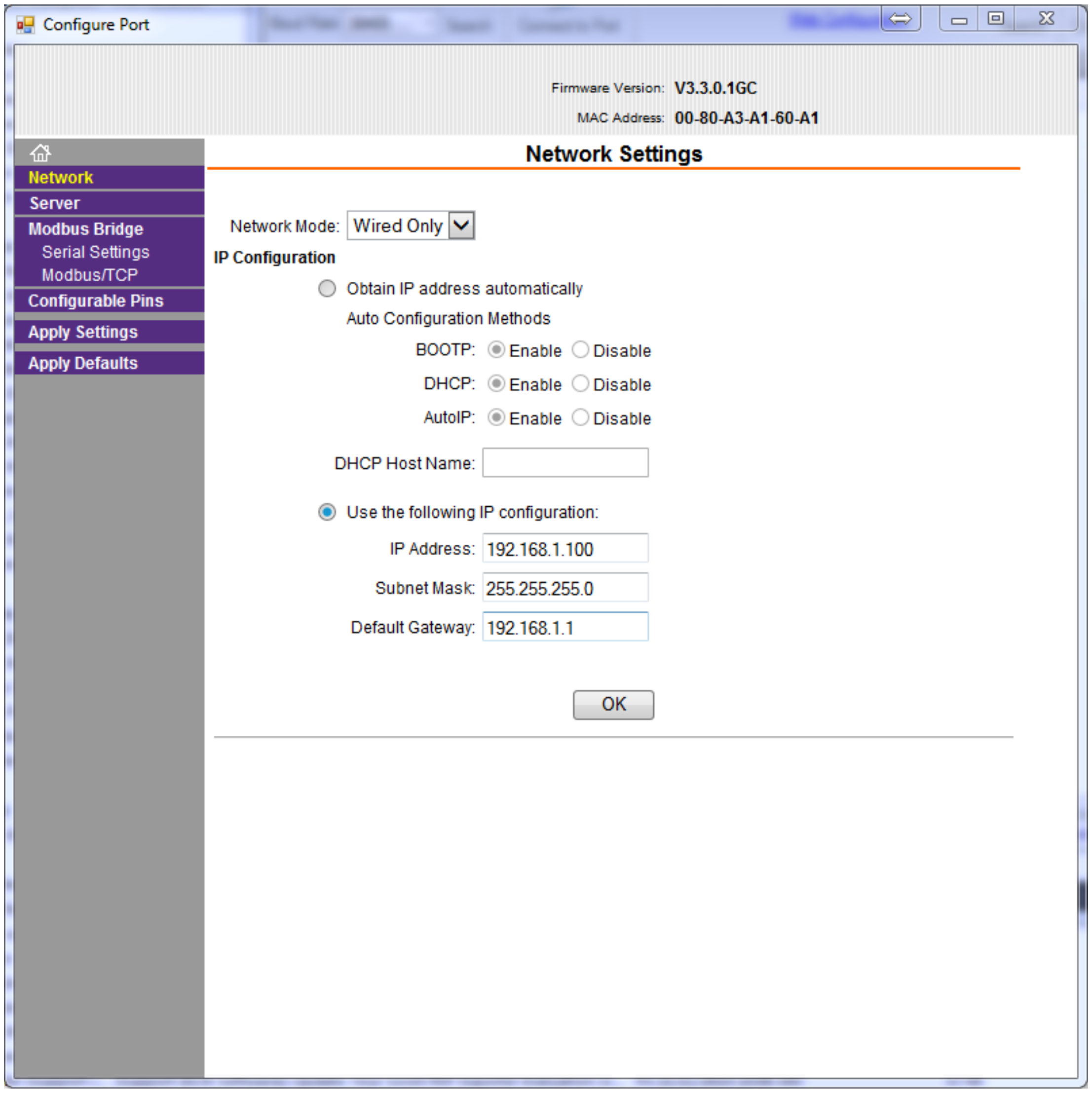
The server 169.254.135.30 is asking for your user name and password. The server reports that it is from (null).

Warning: Your user name and password will be sent using basic authentication on a connection that isn't secure.

Set to Static IP

The motor comes standard with a dynamic IP looking for a DHCP server. If no server is found the module will assign itself and address in the 169.254 range. If the module is plugged directly into a computer it will typically get set in this manner. To assign a static IP use the following step-by-step guide.

1. Logon as described above in the logon guide
2. Click "Network" in the left panel to open up the Network settings



Firmware Version: V3.3.0.1GC
MAC Address: 00-80-A3-A1-60-A1

Network Settings

Network Mode: Wired Only

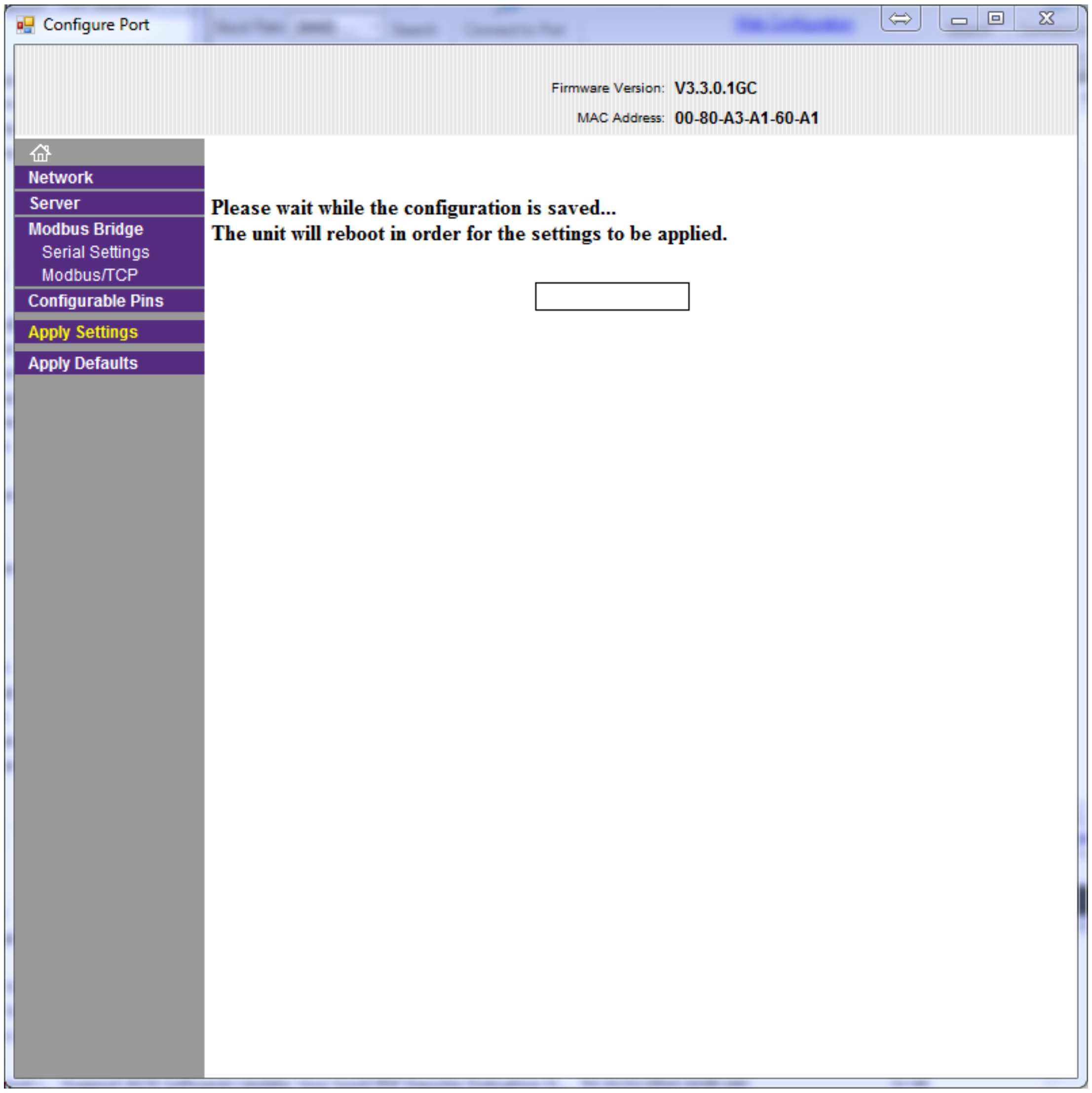
IP Configuration

Obtain IP address automatically
Auto Configuration Methods
BOOTP: [✓ Enable] Disable
DHCP: [✓ Enable] Disable
AutoIP: [✓ Enable] Disable
DHCP Host Name:

[✓ Use the following IP configuration:]
IP Address: 192.168.1.100
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.1.1

a. Select "Use the following IP configuration" and set your required network settings
b. Click the OK button

3. Click "Apply Settings" in the left column. The unit will now reboot.



Please wait while the configuration is saved...
The unit will reboot in order for the settings to be applied.

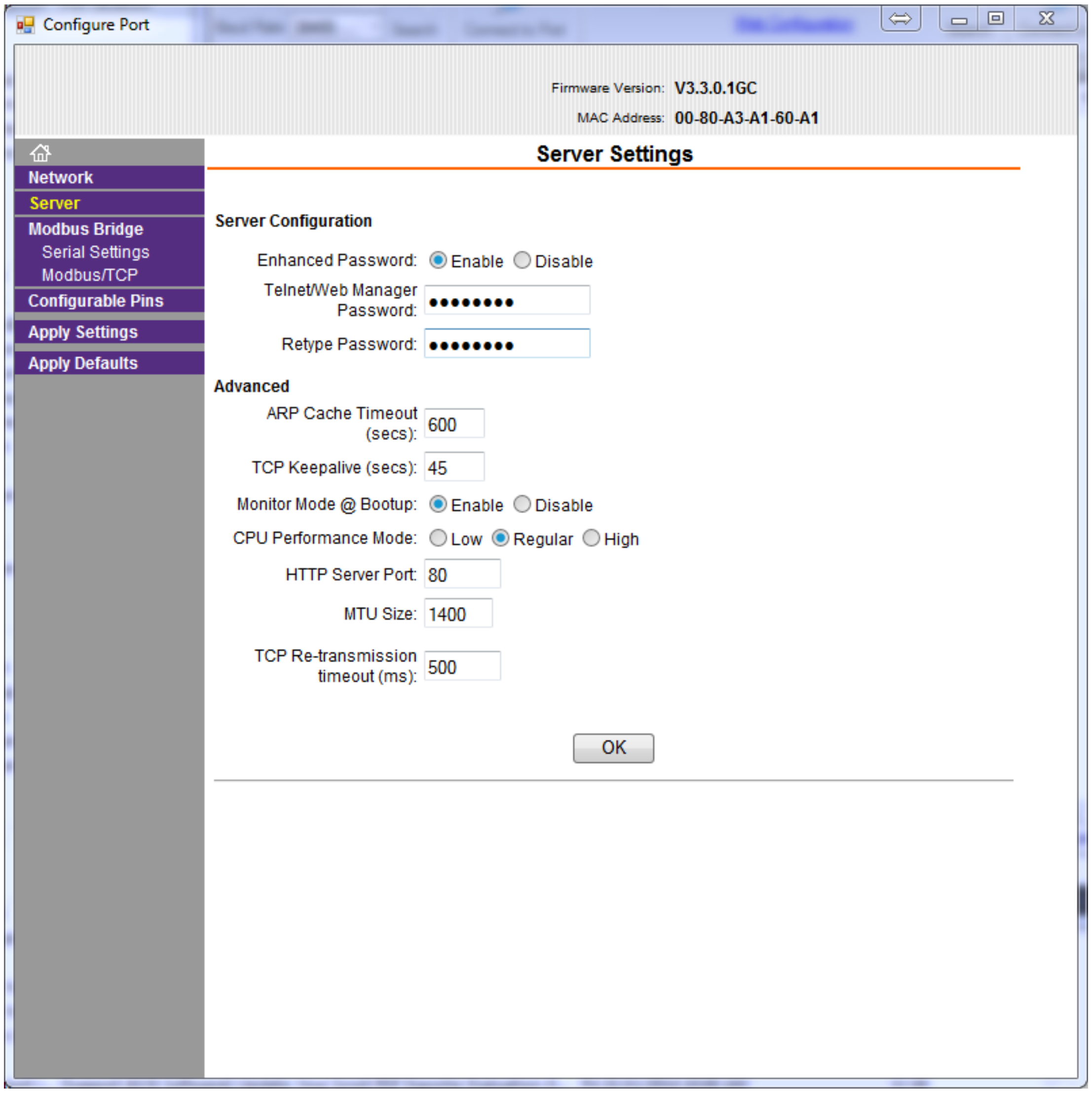
To set back to DHCP or another configuration for a dynamic IP select the "Obtain IP address automatically" radio button.

Set Network Password

The web configuration can be protected with a password.

There is no way to reset the password if it is forgotten without returning the module to the factory.

1. Logon to the web configuration as described above
2. Click "Server" in the left column



Firmware Version: V3.3.0.1GC
MAC Address: 00-80-A3-A1-60-A1

Server Settings

Server Configuration

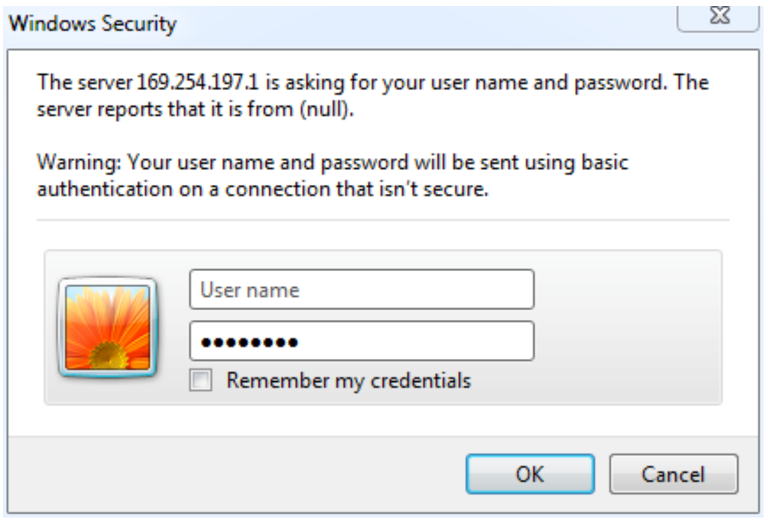
Enhanced Password: [✓ Enable] Disable
Telnet/Web Manager Password: ••••••••
Retype Password: ••••••••

Advanced

ARP Cache Timeout (secs): 600
TCP Keepalive (secs): 45
Monitor Mode @ Bootup: [✓ Enable] Disable
CPU Performance Mode: Low [✓ Regular] High
HTTP Server Port: 80
MTU Size: 1400
TCP Re-transmission timeout (ms): 500

a. Click the Enable radio button on "Enhanced Password"
b. Enter a password
c. Click OK

3. Click "Apply Settings" in the left column.
4. The module will reboot. To login again use the password that has just been set.


Windows Security

The server 169.254.197.1 is asking for your user name and password. The server reports that it is from (null).

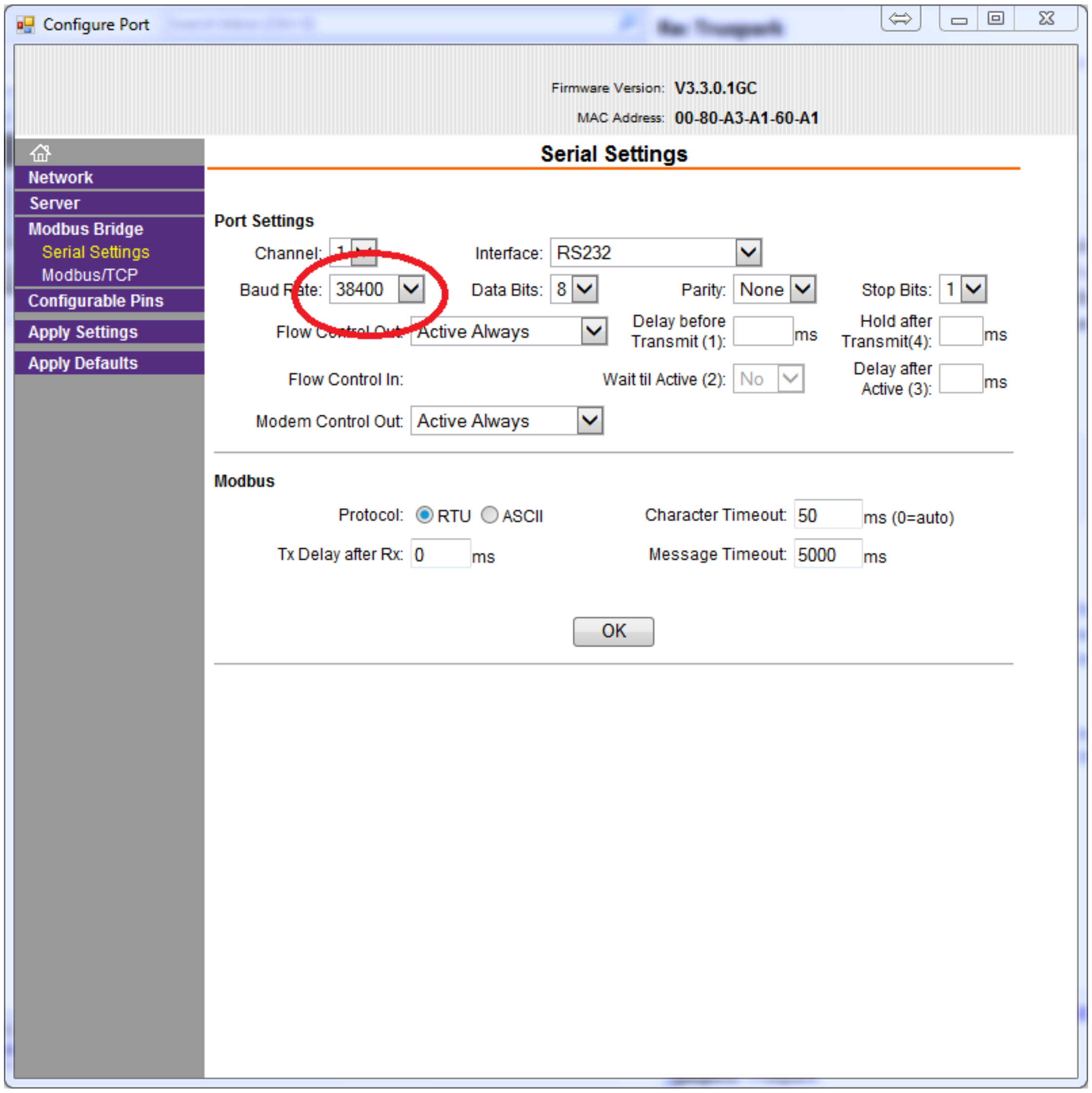
Warning: Your user name and password will be sent using basic authentication on a connection that isn't secure.

User name: User name
Password: ••••••••
☐ Remember my credentials

Configuring Modbus

This section is relevant if a user has clicked "Apply Defaults" on the web interface. The module will have arrived with the correct Modbus TCP settings. Do not change Modbus settings unless instructed to my a Myostat engineer.

1. Click "Serial Settings"
    a. changed the Baud Rate to 38400
    b. Click OK



Firmware Version: V3.3.0.1GC
MAC Address: 00-80-A3-A1-60-A1

Serial Settings

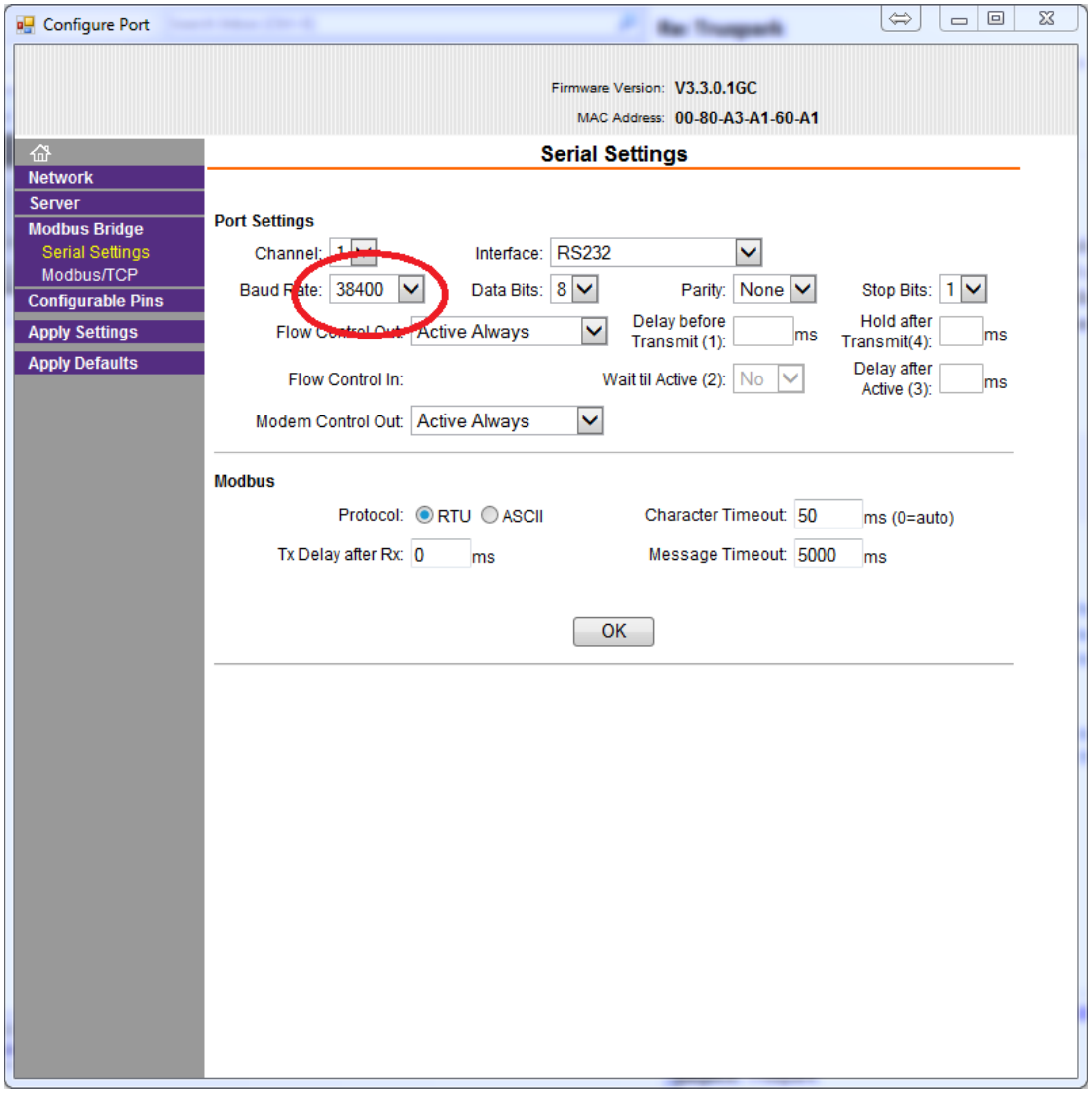
Port Settings

Channel: 1 Interface: RS232
Baud Rate: 38400 Data Bits: 8 Parity: None Stop Bits: 1
Flow Control Out: Active Always Delay before Transmit (1): ms Hold after Transmit(4): ms
Flow Control In: Wait til Active (2): No Delay after Active (3): ms
Modem Control Out: Active Always

Modbus

Protocol: [✓ RTU] ASCII Character Timeout: 50 ms (0=auto)
Tx Delay after Rx: 0 ms Message Timeout: 5000 ms

2. Select "Modbus/TCP"
   a. Change "Fixed Slave Address" to 1
   b. Click Okay



Firmware Version: V3.3.0.1GC
MAC Address: 00-80-A3-A1-60-A1

Serial Settings

Port Settings

Channel: 1
Interface: RS232
Baud Rate: 38400
Data Bits: 8
Parity: None
Stop Bits: 1
Flow Control Out: Active Always
Delay before Transmit (1): ms
Hold after Transmit(4): ms
Flow Control In:
Wait til Active (2): No
Delay after Active (3): ms
Modem Control Out: Active Always

Modbus

Protocol: [✓ RTU] ASCII
Character Timeout: 50 ms (0=auto)
Tx Delay after Rx: 0 ms
Message Timeout: 5000 ms

3. Click "Apply Settings"

The Modbus TCP module has now been set to correctly communicate with the motor.

14.3.2 Configuring Modbus RTU

Requirements

To switch the motor into Modbus RTU you will need the following

1. CM1 motor with RT3.13 firmware (Send "?85" to query the version if you are not sure)
2. Control Room (which can be found here)
3. Communication to the motor from the PC running Control Room. This can be achieved with a number of cables and/or interfaces.
    a. If you are unsure of how to communicate with the motor please see the quick start guide.

Set to Modbus RTU

Once the motor is connected and communicating with Control Room it can be switched from standard ascii communication to Modbus RTU. The following parameters are used to make the change. Read through the descriptions of them and then follow the step-by-step instructions

K20 - COM1 Communication Baud Rate

K20 sets the communication baud rate between the Modbus master and the motor. Using the standard baud rate setting +10 will switch it to Modbus mode with that baud rate

K20 Value	**Baudrate (bps)**
K20=10	38400
K20=11	9600
K20=12	19200
K20=13	57600

The motor default baudrate is 38400.

K62 - Modbus station ID

K62 sets the Modbus station ID. This ID is also used for the RS485 protocol. If the motor starts streaming { with the ID that has been set then it is in RS485 (the software protocol) mode. See this Application Note to switch out of RS485 mode.

K65 - COM2 Communication Baud Rate

COM2 on the motor can be used for Modbus communication. This is not typically used and requires a special cable. If you need to use COM1 for standard communication and COM2 for Modbus communication please contact a Myostat engineer for assistance.

FFFFFFFFF - 9 x Fs function

"FFFFFFFFF" (9xF) is used to temporarily switch the motor out of Modbus mode. This will allow you, until a power cycle, to communicate with the motor using standard ASCII and regular CML. To switch back into Modbus the motor will need to be power cycled or K20 set to normal ASCII and then back to Modbus.

Step-By-Step Guide

Use the following steps to set a motor into Modbus mode. In this example we are setting the motor to ID=1 with a baud rate of 38400bps.

1. Set the motor into modbus mode with a baud rate = 38400

   ```
   K20=10
   ```

2. Send 9 x F to get out of Modbus mode

   ```
   FFFFFFFFF
   ```

3. Set the station ID to 1

   ```
   K62=1
   ```

4. Cycle power on the motor.

The motor is now in Modbus RTU mode and can be communicated with a Modbus master.

14.4 Running the motor in Modbus Mode

14.4.1 Introduction

Modbus gives read/write access to all Cool Muscle registers such as K-parameters, positions, speeds and accelerations. A CML program needs to reside in the motor to execute functions depending on the status of these registers. Below you will find an example program that is used in all Modbus TCP motors. It uses a control word in the R0 register to execute a number of functions. This code is useful for point-to-point motion and speed control. An application may require a significantly more complex program which can replace the example program. The CML is written to be compiled and sent from Control Room. The user does not need to understand the code but only how to use it. The full program code is supplied for those users wanting to change or get a better understanding of how the program works. Modbus TCP motors come standard with the Modbus program loaded.

14.4.2 Running the motor

The following list of holding registers can be used to the read and write move data to the motor.

Holding Register	**Parameter**	**Description**	**R/W**
40201	P0	Target position	R/W

Holding Register	Parameter	Description	R/W
40603	S0	Target speed	R/W
40635	A0	Target acceleration	R/W
40301	R0	Control Word	R/W
40003	Motor Position	Motors current position in pulses	R
40009	Motor Status	Motors current status	R

Position, Speed and Acceleration

The default setting for the motor is K37=3 which sets the following units. This can easily be changed by modifying the value of K37.

Register	Unit/Resolution
P0	1000 pulses/revolution
S0	100 pulse/s
A0	1K pulses/s^2

Control Word

The R0 register is used for the Control Word. It has the following value options by default:

R0 Value	Description
0	Do nothing
1	Start the position move
2	Stop the motor
3	Enable the motor

R0 Value	Description
4	Disable the motor
5	Home the motor

⚠ These functions can be edited or expanded by changing the default program based on user requirements. Please contact our support team if you need assistance with this.

Some things to note when using the control word

1. Changing the value of the control word immediately executes the operation
2. If the Control Word is left with the value 1 then changing the position once the motor has come to a stop will execute the next move. This allows the Modbus master to only change the position and not need to also toggle the control word to execute the next move.
3. The home routine is by default set to a hardstop search in the CCW direction. Please see K42 to K48 for home routine options.

14.4.3 CML Code

The following is the CML code used for motor control in Modbus. It is not required for users to understand the code unless they are looking to change it.

CML Modbus Code

```
//set the logic scan rate to 1ms
K87.1=1
//set logic bank 1 to scan on power up
K85.1=1
//set the modbus register offset to 0
K89.1=0
//switch off all automatic motor event reporting
K23.1=0
//make sure carraige return is not automatic after line feed (legacy setting)
K70.1=0

/*create variables for the old/previous target
control word
position
speed
acceleration
These are used to find a change in the target
Init them to 0
*/
var old_ControlWord R1.1                //old control word
```

```
R1.1=0
var old_TargetPos P1.1      //old position
P1.1=0
var old_TargetSpd S1.1      //old speed
S1.1=0
var old_TargetAcc A1.1      //old acceleration
A1.1=0

/*create variables for the new target
control word
position
speed
acceleration
These are used to find a change in the target
Init them to 0
*/
var ControlWord R0.1                //control word
R0.1=0
var TargetPos P0.1      //position
P0.1=0
var TargetSpd S0.1      //speed
S0.1=0
var TargetAcc A0.1      //acceleration
A0.1=0

/*
Logic L1 scans for a change in the word or any target value
if a change is detected it call the relevant logic bank
*/
L1.1
ControlWord!= old_ControlWord, CL2.1, T0.1  //scan control word
TargetPos!= old_TargetPos, CL3.1, T0.1      //scan position
TargetAcc!= old_TargetAcc, CL4.1, T0.1      //scan acceleration
TargetSpd!= old_TargetSpd, CL5.1, T0.1      //scan speed
END.1
/*
Logic L2 is called if there is a change in the control word
1) it saves the new state into the old state
2) It compares the changed value with defined values to
execute the relavant command
*/
L2.1
old_ControlWord= ControlWord;
old_ControlWord== 1, ^.1, T0.1      //run
old_ControlWord== 2, ].1, T0.1      //stop
old_ControlWord== 3, (.1, T0.1      //enable
old_ControlWord== 4, ).1, T0.1      //disable
old_ControlWord== 5,|.1,T0.1        //home
END.1
/*
Logic L3 executes a change in position
```

```
If the control word eqauls 1 then it executes the move immediately
*/
L3.1
old_TargetPos= TargetPos;
ControlWord== 1, ^.1, T0.1  //execute move is ControlWord equals 1
END.1
/*
The following 2 logic banks set the speed and acceleration
Writing to the value through modbus only changes the register
it does not process the change.
The change must be processed through CML for it to be
executed immediately
*/
//Logic L4 sets the acceleration
L4.1
old_TargetAcc= TargetAcc;
TargetAcc= TargetAcc;
END.1
//Logic L4 sets the speed
L5.1
old_TargetSpd= TargetSpd;
TargetSpd= TargetSpd;
END.1
$.1
```

Download the complete Control Room project here Default Modbus CML program.crp

14.4.4 Sample

Beijer X2 Base

The following sample uses the Beijer X2 Base 7 HMI to communicate to the CM1:

SingleAxisModbus.zip

The following is the same program configured to run on the CM2 motor:

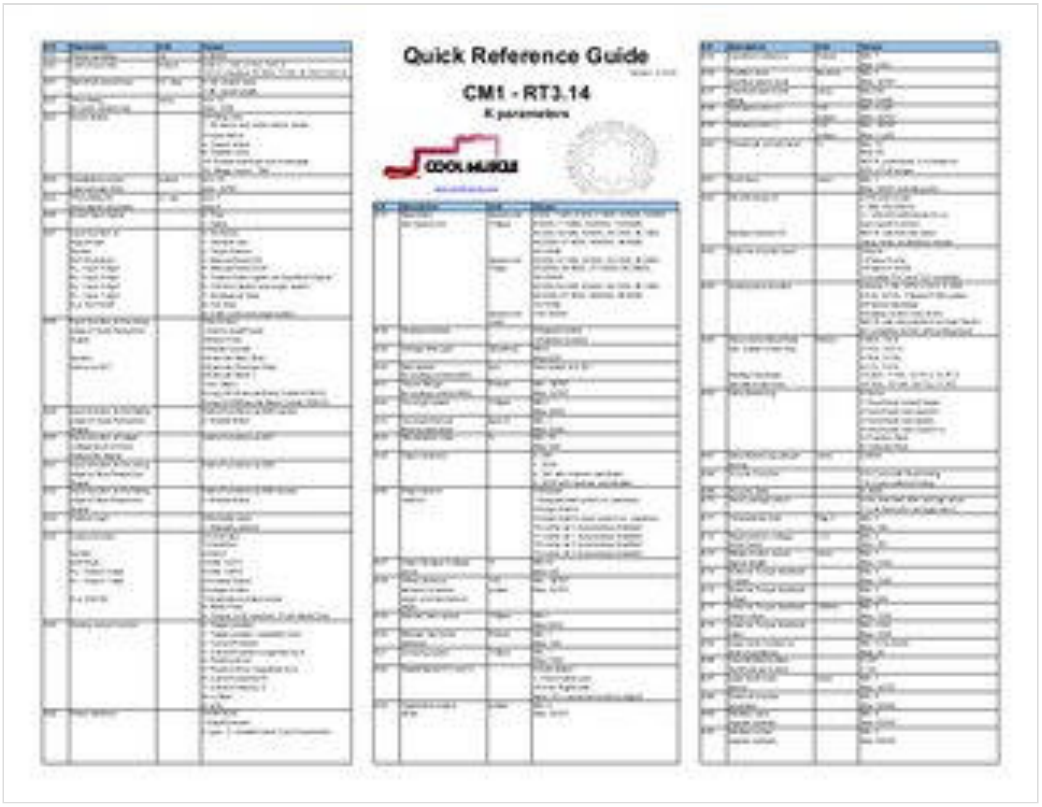
CM2_SingleAxisModbus.zip

Mitsubishi GOT1000

The following program configures a Mitsubishi GOT1000 to run the motor over Modbus RTU. The application is written in GT Designer3.

GOT1000.zip

15 Quick Reference Guide



Quick Reference Guide

CM1 - RT3.14

K parameters

COOL MUSCLE

[illegible]

16 CM1 Firmware Update History

16.1 RT3.14

16.1.1 RT3.14.06

Release date: 31 Mar 2023

Bug fixes

1. RT3D-99 - Ux=0 intermittently not changing to Ux=8 at end of move. This bug was introduced in RT3.14.04. The in-position status Ux=8 was not set at the end of a move. The message Ux=0 was displayed though the motor was within the in-position tolerance (K55).

16.1.2 RT3.14.05

Release date: 01 Feb 2023

Bug fixes

1. RT3D-98. Removed latent debug code that could affect serial communication. The messages "_SBM0", "_SBM1" and "_SBM2" where debug messages that were transmitted on a bank execution.

16.1.3 RT3.14.04

Release date: 25 Nov 2022

Improvements

1. RT3D-73 - V="INxR" and V="INxF". CML variables for input edge detection. A logic or program bank can now use a variable set to detect a rising or falling edge on an input. When the variable is read it returns the number of edges since the last read. The variable is available in all CM1-C and CM1-T
   Variable can be set to the following. E.g. V1="IN1R" for INPUT1 rising edge.
   - IN1R → IN1 rising edge
   - IN1F → IN1 falling edge
   - IN2R → IN2 rising edge
   - IN2F → IN2 falling edge
   - IN3R → IN3 rising edge
   - IN3F → IN3 falling edge
   - IN4R → IN4 rising edge
   - IN4F → IN4 falling edge
2. RT3D-86 - V="IN". CML variable which returns the combined value of all 4 digital inputs. Read as an integer value in logic and program banks. This is useful when inputs are used as a binary combination to perform multiple functions. Input calculation is as follows:
   - "IN" = (8 x IN4) + (4 x IN3) + (2 x IN2) + (IN1)
     The INMK variable is used to mask the inputs and execute the calculation with or without certain inputs
     The INML variable is available on CM1-C and CM1-T motors.
3. RT3D-92 - V="INMK". CML variable used in conjunction with V="IN". This variable masks inputs to be used or not used in the calculation. The variable is set as a digital value but uses the relative bits as a mask. I.e. bit0 = IN1 mask, bit1 = IN2

mask, bit2 = IN3 mask, bit3 = IN4 mask. By default on power up the INMK=15 so all inputs are active in the V="IN" calculation. "INMK" is available on CM1-C and CM1-T motors.
V="INMK" examples
- INMK = 12 → IN4 and IN3 only (bit3=IN4=1, bit2=IN3=1, bit1=IN2=0, bit0=IN1=0 → 0b1100 = 0xC = 12_d)
- INMK = 3 → IN2 and IN1 only (bit3=IN4=0, bit2=IN3=0, bit1=IN2=1, bit0=IN1=1 → 0b0011 = 0x3 = 3_d)
- INMK = 11 → IN4, IN2 and IN1 only (bit3=IN4=1, bit2=IN3=0, bit1=IN2=1, bit0=IN1=1 → 0b1011 = 0xB = 11_d)

Bug Fixes

1. RT3D-72 - V="AO2" not initialised correctly on power up. "AO2" is now initialised correctly and the analog output can be set in a logic or program bank. The analog output is only available on CM1-C motors.
2. RT3D-82 - Motor still moves when S=0 on an absolute position move. When S=0 was set during an absolute move the motor would continue to move very slowly. This was due to legacy code that did not allow a value of S=0 during an absolute position move. A value of S=0 is now accepted during an absolute position move which halts the motor. As the motor is not in-position it will return a status of Ux=0 indicating the move is still "running" though there is no physical movement. A stop command is required to put the motor into a complete stopped state (Ux=8).
3. RT3D-93 - Home routine won't execute if started directly after an error. Prior to RT3.14.04 a small dummy move was required if an error occurred during a home routine. A home routine can now be executed directly after an error or loss of drive power.

16.1.4 RT3.14.03

Release date: 17 Sep 2022

Improvements

1. RT3D-81 - New CML command to switch off automatic saving of K and H parameters. This is useful in an application where K and/or H parameters are changed during runtime. Repeated changing of parameters will damage the EEPROM. The following commands are now available to be sent over the communication port. There is no ability to call the commands from a logic or program bank
   - _SKH=0 switches off save
   - _SKH=1 switches on save (default on power up)
   - _SKH will query and return the current value

Bug fixes

1. RT3D-77 - Quadrature output would stream quadrature output pulses if K34 (digital output option) or K24 (quadrature resolution) changed during runtime. This typically had no affect as the parameters are not changed during runtime. Applications where the parameters are changed during runtime will no longer see a stream of pulse/position when no movement is occurring.

16.1.5 RT3.14.02

Release date: 23 Aug 2022

Bug fixes:

1. RT3D-75 - IN1 and IN4 on CM1-T/E not functioning. Did not affect CM1-C

16.1.6 RT3.14.01

Release date:26 Apr 2021

Improvements

1. Improved startup timing between hardware revision E and revision C when mixed on a daisy-chain network

Compatibility updates

1. Addition inputs used with the CM1-T inverted

Bug fixes

1. UART0 (primary communication port) would stop communicating in specific circumstances. The issue was seen to occur on a small number of daisy-chained motors that included a power-up delay.
2. Merge motion function could hang the motor if the motor is disabled milliseconds prior to executing a merge.

16.1.7 RT3.14.00

Release date:26 Jan 2021

Improvements

1. High speed communication implemented for use with the CM1-T ethernet module
2. Additional modes of operation introduced for more direct control of the motor
   a. Profile mode - change target position, speed and acceleration while in motion.
   b. Torque mode - run a in a torque only mode

16.2 RT3.13

16.2.1 RT3.13.12

Release date:24 Mar 2021

Improvements

1. Improved startup timing between hardware revision E and revision C when mixed on a daisy-chain network

Bug fixes

1. UART0 (primary communication port) would stop communicating in specific circumstances. The issue was seen to occur on a small number of daisy-chained motors that included a power-up delay.

16.2.2 RT3.13.10

Release date:15 Oct 2020

Bug fixes

1. The line feed option (K70=1) was inserting an additional line feed in daisy-chain motors.
2. The Ux=256 pushmode timeout error was not sent on the serial port in a bank.

16.2.3 RT3.13.09

Release date:06 May 2020

This is the initial backwards compatible release for RT3 on the CM1 hardware revision E. Please see the attached engineering change notice for information on the change ECN RT3D-23.pdf

17 Coordinated Motion

17.1 Introduction

Coordinated motion allows 2 cool muscle motors to work together in an X-Y motion to create accurate and complex motions such as lines, arcs, circles, and ellipses. By using merge motion, each move can be merged with the next to create a smooth motion between points without a stop.

The following variables and commands are used for coordinated motion:

- R1-R25: Circle/Arc Radius value
- N1-N25: Circle/Arc Center point
- @0-@25: Execute motion to end point defined by P1-P25 respectively

In order to tell the two motors to move in coordination, we need to define the speed and acceleration as normal, as well as either a radius or center point. We can then use the @ command similar to how we would use the P command in a normal motion to set the target end point. The start point of any coordinated motion move is the current position.

For timing reasons, any coordinated motion move must send the move command to the second motor first. For example:

```
A1.1,S1.1,A1.2,S1.2

R1.1,R1.2

@1.2+,@1.1+
```

In the above, we can call the speeds, accelerations, and radii in any order but the @ move command must be called in the second motor before the first.

⚠ The are a few rules to note when using coordinated motion:

1. Coordinated motion must be run from a program bank, and cannot be used in logic banks or direct commands.
2. Motions must be run in consecutive motors. e.g. you can coordinate motors 1 and 2, or 2 and 3, but cannot do a coordinated motion between motors 1 and 3.
3. The move commands must be given in reverse order from the motor the program is stored on. e.g. if you are using motors 1 and 2, and the program is stored on motor 1 the move command for motor 2 must come first.
4. In the case of running the bank from the third motor, the move command for the farthest motor must come first. e.g. running the same move as rule 3 from a third motor, the move command for motor 1 must come first since it is the farthest from the third motor.

- Arcs
- Circles, Ellipses, and Lines
- Merge Motion

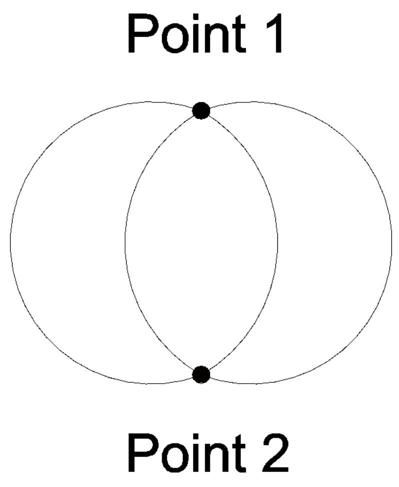
17.2 Arcs

There are two formats you can use to complete an arc: you can define the center point of the arc and the end point, or you can define the radius of the arc and the end point.

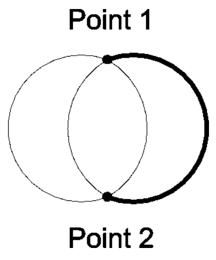
17.2.1 Radius and End Point

If you imagine an arc between two points, there are four possible arcs that can be created if only the radius and the end point is defined. As illustrated in figure 13 below there is a small clockwise arc, a large clockwise arc, as well as a small counter clockwise arc, and a large counter clockwise arc.

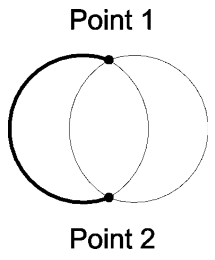
By manipulating the @ and R values, it is possible to obtain all four of these arcs. R, the radius, will define the size of the arc, and @ will define whether we are travelling clockwise or counter clockwise. To make an arc, the radius set for motors one and two must be the same.



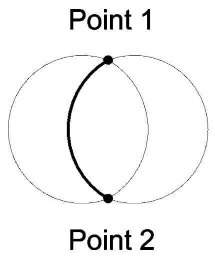
13 Fig 13 - Arcs with only radius and endpoint defined



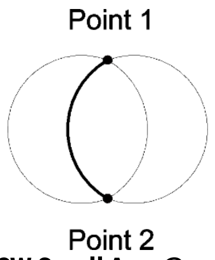
14 CW Large Arc: @+ and R>0



15 CW Small Arc: @+ and R<0

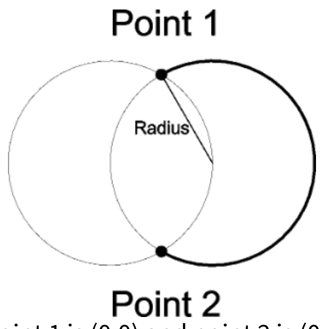


16 CCW Large Arc: @- and R>0



17 CCW Small Arc: @- and R<0

For Example, if we wish to make the following arc:

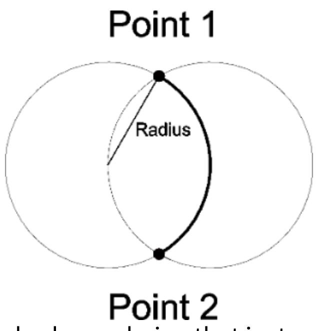


We could use the following program, assuming point 1 is (0,0) and point 2 is (0,-500). This means the radius is 353.

```
P2.1=-0, P2.2=-500
R1.1=353, R1.2=353
A1.1=10. A1.2=10
S1.1=10, S1.2=10

B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
R1.1, R1.2, @2.2+, @2.1+
END
```

In the above example, we are defining point 2 as (0,-500) and the radius as 353. The first line in the bank program sets the current position as 0, we then call the accelerations and speeds, calls the radius in R1.1 and R1.2 and executes the motion.

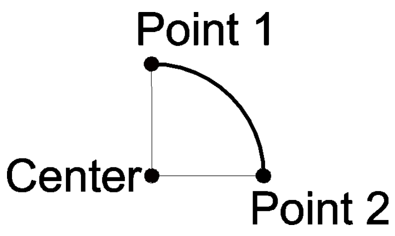
Contrarily, if we wanted to make the following arc:



We could use the exact same program, with the only change being that instead of the radius in R1.1 and R1.2 being set to 353, it would be set to -353.

17.2.2 Center and End Point

By using the center and end point method, the arc is completely defined with regards to the arc size and only the direction need to be specified.



18 Fig. 14 - Center and end point definition

For example, if we wish to draw the arc in figure 14 using the center and end point method, we could do the following:

```
P2.1=-500, P2.2=-500
N1.1=0, N1.2=-500
```

```
A1.1=10. A1.2=10
S1.1=10, S1.2=10



B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
N1.1, N1.2, @2.2+, @2.1+
END
```

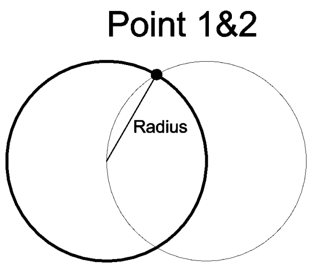
In the above example, we are defining point 2 as (-500,-500) and the center point as (0,-500). The first line in the bank program sets the current position as 0, we then call the accelerations and speeds, set the center point as N1.1, N1.2 and execute the motion.

17.3 Circles, Ellipses, and Lines

Circles, ellipses, and lines are essentially special cases of the arc functionality mentioned above: Circles are an arc with the same start and end point, lines are an arc with infinite radius, and an ellipse is an arc or circle with a different radius called on each axis.

17.3.1 Circles

A circle can be defined by using either of the two above methods for creating an arc, and having the start and the end point the same. Using the example above for radius and end point, we can make this a circle as follows:



```
P2.1=0, P2.2=0
R1.1=-353, R1.2=-353
A1.1=10. A1.2=10
S1.1=10, S1.2=10

B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
R1.1, R1.2, @2.2+, @2.1+
END
```

By simply changing P2.1 and P2.2 to 0, we have made the arc in to a circle.

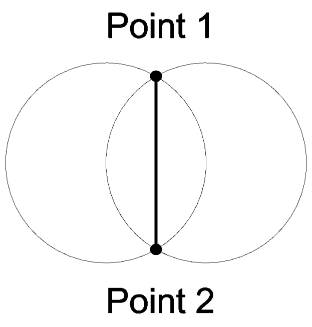
A second option is also available which allows you to run multiple circles continuously if the @ command is used by calling @0.1 and @0.2 instead of @1.1,@1.2, etc. In this case, the value of P0 is used to define the number of circles to run and the end point is automatically assigned to the start point. Using the above example, we can run it as follows:

```
P0.1=5, P0.2=5
R1.1=-353, R1.2=-353
A1.1=10. A1.2=10
S1.1=10, S1.2=10

B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
R1.1, R1.2, @.2+, @.1+
END
```

This will run the same circle as above, but it will run the circle 5 times in a row.

17.3.2 Lines

A Line is defined as an arc where the radius is set to 0.



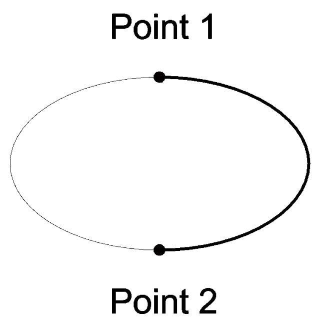
The above line can be created by the following:

```
P2.1=0, P2.2=-500
R1.1=0, R1.2=0
A1.1=10. A1.2=10
S1.1=10, S1.2=10

B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
R1.1, R1.2, @2.2+, @2.1+
END
```

You will notice that this is essentially the same motion as the example for an arc, with only the radius being changed to 0.

17.3.3 Ellipses

An ellipse or elliptical arc can be created in the same manner as a circle or regular arc, but with the radii of the two axes different. Take, for instance, the following elliptical arc:



```
P2.1=0, P2.2=-500
R1.1=353, R1.2=706
A1.1=10. A1.2=10
S1.1=10, S1.2=10

B1
|2.1, |2.2
A1.1, S1.1, A1.2, S1.2
R1.1, R1.2, @2.2+, @2.1+
END
```

The above program is identical to the regular large clockwise arc, except the radius for motor 2 has been doubled. This gives us the exaggerated movement to create an elliptical arc.

17.4 Merge Motion

Motions within a bank can be run discretely, as shown in the above examples, or merged. Merging the motions provides a smooth, continuous movement in which the motors do not stop at the target location, but merely pass through it. This is achieved by appending end of a move with a semicolon ';' character. If a line in a program bank ends with a ';' it will merge the next move in the program in to the current one.

For example:

```
R2.1, R2.2, @1.2, @1.1
R1.1, R1.2, @2.2, @1.1
```

Is a discrete movement, but:

```
R2.1, R2.2, @1.2, @1.1;
R1.1, R1.2, @2.2, @1.1
```

Will merge the two profiles in to a seamless motion. This also works with a normal movement that is not a coordinated motion, and will even work with a single motor. For instance:

`A1.1,S1.1,P1.1;`

`A1.1,S1.1,P2.1`

Is also a valid merge motion command.

18 Application Notes

18.1 Jogging using an Input

The following examples show two different ways of jogging the motor using an input trigger.

The first example uses only K parameters. There is no program running in the motor.

```
K27=0034        //Programs IN1 to Manual Feed CCW and IN2 to Manual Feed CW
K43=100         //Set Manual Feed Acceleration to 100 000 pulses per second squared
K49=2           //Sets Manual Feed Speed to 200 pulses per second (1/5 of a rotation per
second by default)
```

The next example runs a program bank each time an input is triggered to jog the motor. While there is more programming done here, it is possible to use this if there is more that needs to happen than just moving. For example, you may need a variable set to indicate a direction or distance for a controller or PLC, you may want an output to activate under a certain condition, or you may need to be able to dynamically change the acceleration or speed of the move.

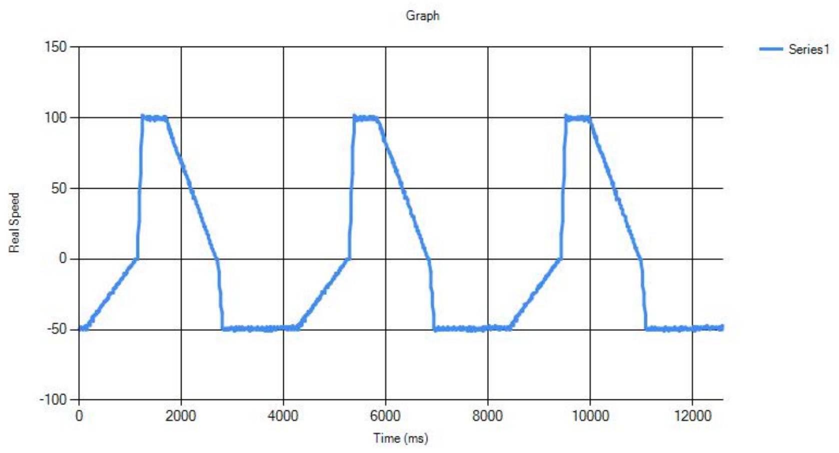
```
B100.1              //CLEAR PROGRAM BANKS
L100.1              //CLEAR LOGIC BANKS

K36.1 = 2           //ENABLES BANK 2&3 EXECUTION
K37.1 = 3           //1000ppr RESOLUTION AND 100pps SPEED
K28.1 = 8060        //INPUT 2 RUN BANK 1, INPUT 4 RUN BANK 2
K29.1 = 1010        //INPUT 2 AND 4 PAUSE BANK WHEN RELEASED

S1.1 = 100          //SPEED FOR FORWARD JOG
S2.1 = -100         //SPEED FOR REVERSE JOG

A1.1 = 80           //ACCELERATION VALUE

P1.1 = 1000000000   //INFINITE MOVE POSITION

B1.1                //PROGRAM BANK 1
S1.1,A1.1,P1.1      //RUN FORWARD TO INFINITE POSITION
END.1               //END BANK

B2.1                //PROGRAM BANK 2
S2.1,A1.1,P1.1      //RUN BACKWARD TO INFINITE POSITION
END.1               //END BANK
```

```
$.1                 //SAVE PROGRAM
```

18.2 Move with Different Acceleration and Deceleration

18.2.1 Description

The required move needs a different acceleration and deceleration. By design the Cool Muscle uses a single acceleration value (a#) for both acceleration and deceleration. The following graph shows motor speed as it repeatedly moves between position A and position B. The positive speed is the motor moving from A to B and the negative speed is the motor moving back from B to A. Both moves show a different acceleration and deceleration.



The required move is achieved using the merge motion command structure.

Merge Motion
Merge motion allows you to merge multiple position points together.
In this case we will have a position halfway between the start position and the end position. The motor will first move to this point with the defined acceleration. It will then merge into the end point with the acceleration changed to the required deceleration value.

e.g.
A1,S1,P3,A2,P1 --> with A1 and S1 move to P3, then switch to A2 and stop at P1.

Caution should be taken that the move is theoretically possible. I.e. the motor can decelerate to the final target position with the desired distance and deceleration value.

18.2.2 Program

```
//Move 1 parameters
a1=100          //acceleration value
a2=10           //deceleration value
```

```
s1=100          //target speed
p1=10000        //final position

//Move 2 parameters
a3=50           //acceleration value
a4=5            //deceleration value
s2=50           //target speed
p2=0            //final position
p3=5000         //midpoint position used by both moves

//Example bank
B1
X0
A1,S1,P3,A2,P1      //move 1
A3,S2,P3,A4,P2      //move 2
X-
END.1
```

18.2.3 Attachments

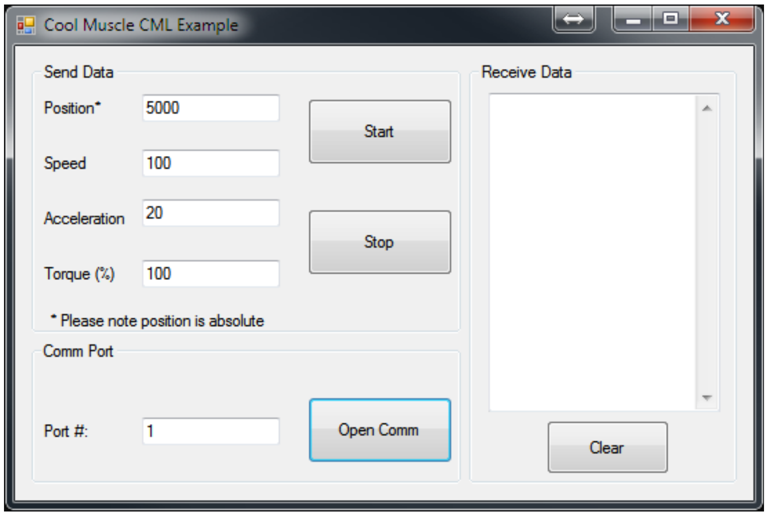
File	Modified
Accel-Decel Example.crp	Apr 08, 2016 by Mark McCann

18.3 C# Coding Example

18.3.1 Introduction

The attached example shows how to execute a point-to-point move with the Cool Muscle. The code will work for both a CM1 and CM2. It uses a standard serial port connection to the motor and executes the direct move commands. The following CML is used

- P0 - Target position
- S0 - Target speed
- A0 - Target acceleration
- M0 - % Peak torque
- ^ - Start move
-] - Stop move



18.3.2 Code

The following code shows the key start and stop function

Start Function

```
        private void button_Start_Click(object sender, EventArgs e)
        {
            /* Run the motor using the dynamic move structure
             * Set P0, S0, A0 and M0. Then start the motor with the ^ command
             * We assume a single motor in this example so use the .1 ID
             * P, S, A and M are separated with a comma (,)
             * All instructions must be appended with a carriage return
             */

            String sSend;

            sSend = "P0.1=" + textBox_Position.Text +
                    ",S0.1=" + textBox_Speed.Text +
                    ",A0.1=" + textBox_Acceleration.Text +
                    ",M0.1=" + textBox_Torque.Text +
                    "\r";

            //write the registers data
            serialPort.Write(sSend);

            //start the motor
            serialPort.Write("^.1\r");

        }
```

Stop Function

```
        private void button_Stop_Click(object sender, EventArgs e)
        {
            /* Stop the motor with the ] command.
             * We assume a single motor in this example so use the .1 ID
             * All commands must be appended with a carriage return
             */

            serialPort.Write("].1\r");
        }
```

18.3.3 Attachments

File	Modified
Cool Muscle CML Example.zip	Feb 14, 2019 by Mark McCann

18.4 Analog Position Output

The following program uses the analog output on OUT2 to output a voltage relative to a user-defined range of positions.

```
B100                    //Clear existing program banks
L100                    //Clear existing logic banks

K34=00                  //Set Output 2 as analog output
K85=1                   //Run logic bank 1 on power up
K87=1                   //Logic bank scan time of 1ms


var Px      V1.1        //Variable V1 is named Px (current Position)
V1.1="Px"               //V1 is set to the current motor position

V2.1="AO2"              //V2 is set to the analog output value

V5.1=0                  //V5 is used as an accumulator in the calculations

var offset  V6.1        //Variable V6 is named (used in calculations)
V6.1=0                  //V6 is an offset in the event of negative position values

var P_Max   V4.1        //Variable V4 is named P_Max (maximum position value)
P_Max=5000              //Set max position to 1000
```

```
var P_Min   V3.1        //Variable V3 is named P_Min (minimum position value)
P_Min=-5000             //set min position to 0

N1.1=-1                 //constant used in calculation

L1                      //Logic bank 1
V5.1=P_Max-P_Min;       //Subtract min position from max position to find range
P_Min<0,CL2,offset=0;   //If the minimum position is negative, call logic bank 2, otherwise
set the offset to 0
V5.1=V5.1/256;          //Scale the range to the range of the analog output
V5.1=Px/V5.1;           //Scale the output to the current position value
V5.1=V5.1+V6.1;         //Add the offset to the current calculated value
V5.1>255,V5.1=255,T0;   //If the value is above the analog range, set the analog output to
max
V5.1<0,V5.1=0,T0;       //If the value is below the analog range, set the analog output to 0
V2.1=V5.1;              //Set analog output to the calculated value
END.1

L2                      //L2 is used to scale the offset value in the event of negative
positions
offset=V5.1/256;
offset=P_Min/offset;
offset=offset*N1.1;
End.1
$                       //save program to EEPROM
```

18.5 Move Based on Binary Combination Input

The following program uses the inputs 1-4 to run a motion based on the binary combination of the inputs. For example if IN1 is triggered, motion 1 is run. If IN1 and IN2 are both triggered, motion 3 is run, and so on.

```
B100.1          //Clear any existing Program and Logic Banks
L100.1

K87=5           //LOGIC BANK SCAN TIME OF 5ms
K85=1           //START LOGIC BANK 1 ON POWERUP

//Assign required acceleration values
A1=100
A2=100
A3=100
A4=100
A5=100
A6=100
A7=100
A8=100
A9=100
```

```
A10=100
A11=100
A12=100
A13=100
A14=100
A15=100

//ASsign required Speed values
S1=100
S2=100
S3=100
S4=100
S5=100
S6=100
S7=100
S8=100
S9=100
S10=100
S11=100
S12=100
S13=100
S14=100
S15=100

//Assign required position values
P1=1000
P2=2000
P3=3000
P4=4000
P5=5000
P6=6000
P7=7000
P8=8000
P9=9000
P10=10000
P11=11000
P12=12000
P13=13000
P14=14000
P15=15000

/*assign N values to be
used as constants to compare
to the input combination*/
N10=0
N11=1
N12=2
N13=3
N14=4
N15=5
N16=6
```

```
N17=7
N18=8
N19=9
N20=10
N21=11
N22=12
N23=13
N24=14
N25=15

//set default values for variables
V0=0
V1=0
V3=0
V2=2
V4=4
V8=8



L1.1
V0=I1;           //CALCULATE DECIMAL VALUE OF BINARY INPUTS
V1=I2*V2;
V0=V0+V1;
V1=I3*V4;
V0=V0+V1;
V1=I4*V8;

V0=V0+V1;
V0!=V3, CL2,T0  //IF VALUE HAS CHANGED, CALL LOGIC BANK 2
END.1

L2.1
V3=V0;           //SAVE NEW VALUE
V0==N10,]:],CL3 //IF VALUE OF INPUTS IS 0, STOP MOTOR, ELSE CALL LOGIC BANK 3
END.1


L3.1
V0==N11,[1.1,T0 //IF THE DECIMAL VALUE OF BINARY INPUTS EQUALS 1, RUN BANK 1
V0==N12,[2.1,T0
V0==N13,[3.1,T0
V0==N14,[4.1,T0
V0==N15,[5.1,T0
V0==N16,[6.1,T0
V0==N17,[7.1,T0
V0==N18,[8.1,T0
V0==N19,[9.1,T0
V0==N20,[10.1,T0
V0==N21,[11.1,T0
V0==N22,[12.1,T0
```

```
V0==N23,[13.1,T0
V0==N24,[14.1,T0
V0==N25,[15.1,T0
END.1

B1.1            //Program Bank 1
A1,S1,P1        //Run to position 1, at speed 1, and acceleration 1
END.1

B2.1
A2,S2,P2
END.1

B3.1
A3,S3,P3
END.1

B4.1
A4,S4,P4
END.1

B5.1
A5,S5,P5
END.1

B6.1
A6,S6,P6
END.1

B7.1
A7,S7,P7
END.1

B8.1
A8,S8,P8
END.1

B9.1
A9,S9,P9
END.1

B10.1
A10,S10,P10
END.1

B11.1
A11,S11,P11
END.1

B12.1
A12,S12,P12
```

```
END.1

B13.1
A13,S13,P13
END.1

B14.1
A14,S14,P14
END.1

B15.1
A15,S15,P15
END.1

$.1         //Save program
```

18.6 Creating a figure-eight using Coordinated Motion

The following program uses the Center and Endpoint method of defining a series of four arcs in order to create a figure-eight motion in a cartesian X-Y system. This program also uses Merge Motion to make a smooth seamless transition from one arc to another.

```
B100.1,L100.1           //Clear any existing program and logic banks
B100.2,L100.2

K85.1=1                 //Run Logic Bank 1 on power up

R1.2=0,R1.1=0           //Radii not needed for this method

A1.1=50,A1.2=50         //Define accelerations and speeds
S1.1=30,S1.2=30

P1.1=0                  //First position of first arc
P1.2=1600               //Because this Figure-Eight is drawn along the motor 2 axis, the
motor 2 end position is always the same
P2.1=3200               //Arc End Position
P3.1=6400               //Arc End Position

N1.1=1600               //Center point of first figure eight
N1.2=1600               //Again, Because this is drawn along the motor 2 axis, the motor 2
center point is always the same
N2.1=4800               //Center point of other side of the figure-eight

l1.1
[1.1                    //Run Program Bank 1
jl2                     //Jump to Logic Bank 2
end.1
```

```
l2.1                    //Empty Logic Bank so that we don't keep executing program bank 1
end.1

B1.1
|2.1,|2.2               //Home both axes
A1.2,A1.1               //call accelerations and speeds
S1.2,S1.1
X0.1                    //Everything between the X0 and X- is looped indefinitely
N1.2,N1.1,@1.2,@2.1;    //Create arc to first position
N1.2,N2.1,@1.2-,@3.1-;
N1.2,N2.1,@1.2-,@2.1-;
N1.2,N1.1,@1.2,@1.1;
X-
END.1
$.1                     //Save Program
```

19 CM1 Interface Modules

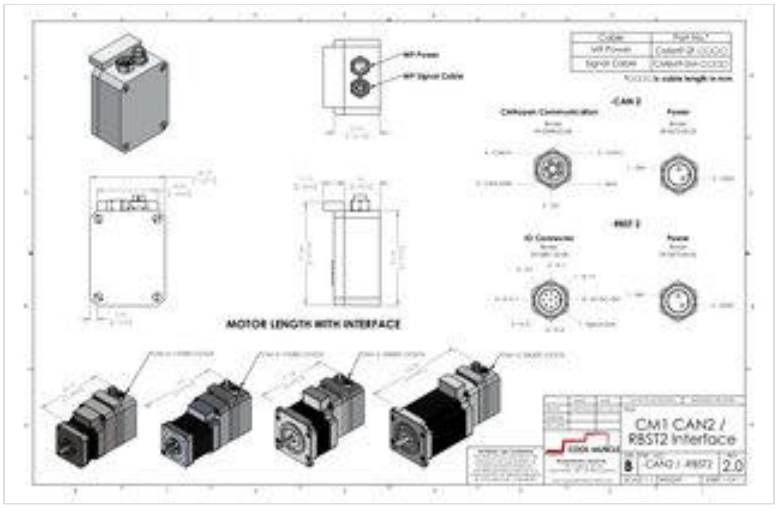
The CM1 motor has a variety of different interface modules that can be fitted for various purposes.

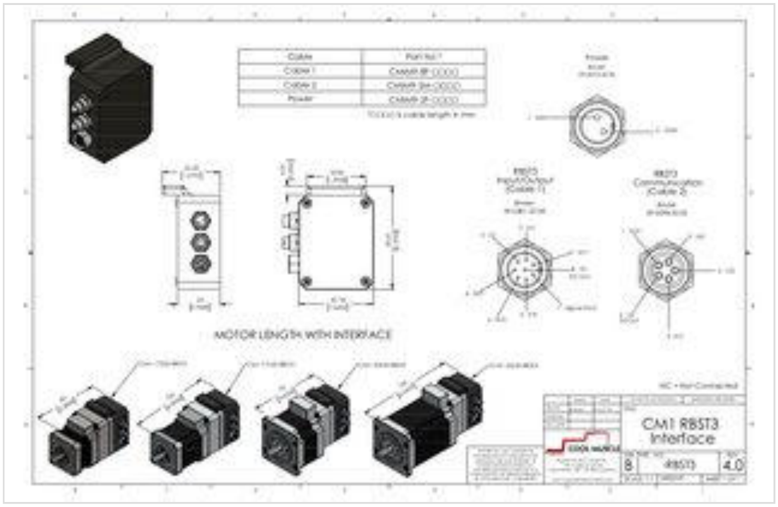
19.1 RBST

The RBST or 'Robust' modules are designed to allow a more robust, industrial style connection method. The RBST module comes in two types: The RBST-2 and RBST-3.

The RBST-2 has two M9 screw down connectors. One connector has two pins for power and ground, the other connector is an 8-pin connector with access to inputs 1,2,4 and Output 1. This can be used for communicating to a PC or for accessing the digital I/O.

The RBST-3 has three M9 screw down connectors. The first two connectors are similar to the RBST-2 and the third allows for connection to the remaining input and output.



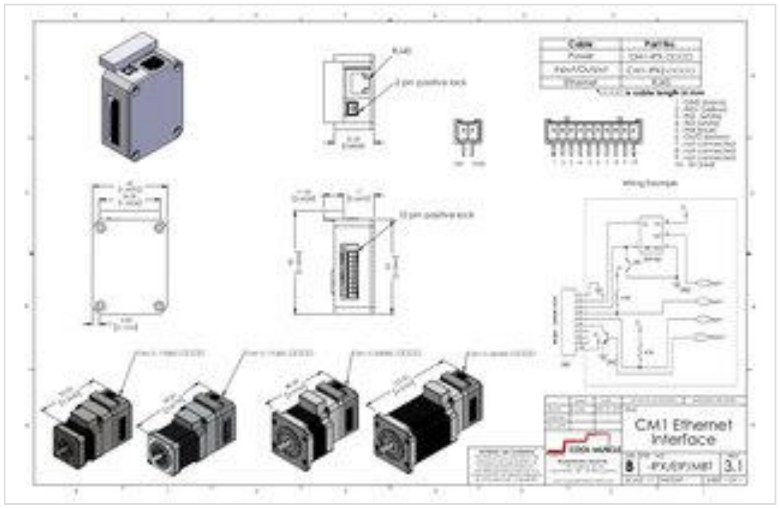


19.2 MBT

This Modbus TCP interface has been deprecated. Please see CM1-T Modbus TCP for the latest Modbus TCP version of the CM1.

The MBT module allows the CM1 motor to communicate to a PLC or other controller using Modbus TCP. The MBT module shares a form factor with the IPX.

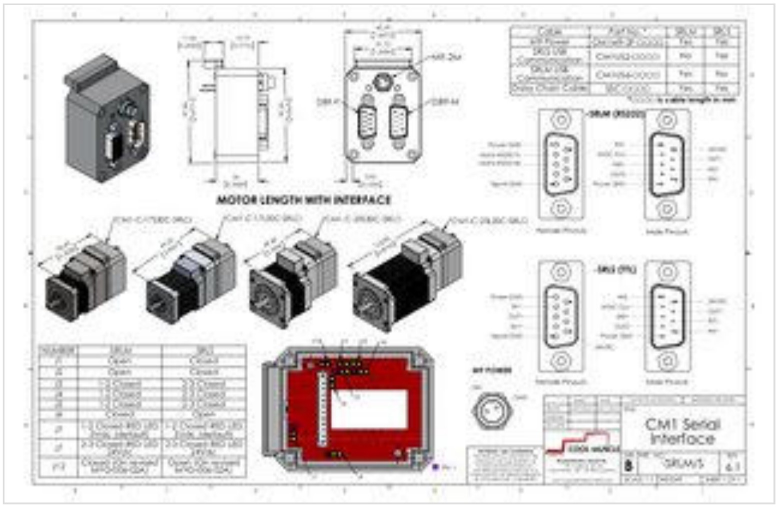
For further information on the CM1 Modbus control scheme, see Modbus-RTU.



19.3 SRL

The SRL module comes in two different configurations, the SRLM and SRLS. These are the master and slave modules respectively. Using this master/slave configuration, multiple motors can be easily daisy chained by using a simple straight through DB9 serial cable. The SRL modules allow for the use of RS-232 serial communications, as well as quick and convenient access to inputs, outputs, and daisy chaining multiple motors.

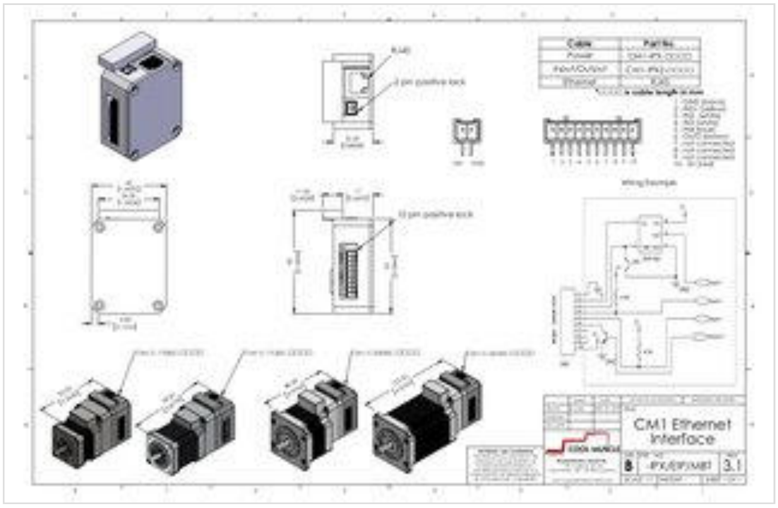
The SRL modules also come equipped with a more robust voltage regulation circuit to help deal with any regenerated voltage problems.



19.4 IPX

⚠ This Ethernet interface has been deprecated. Please see CM1-T CML port for the latest TCP/IP interface for the CM1.

The IPX module allows you to communicate to the motor using TCIP/IP through a virtual COM port. The IPX module has an RJ45 ethernet jack, as well as a 2-pin positive lock power connector and a 10-pin connector for I/O and daisy chaining.

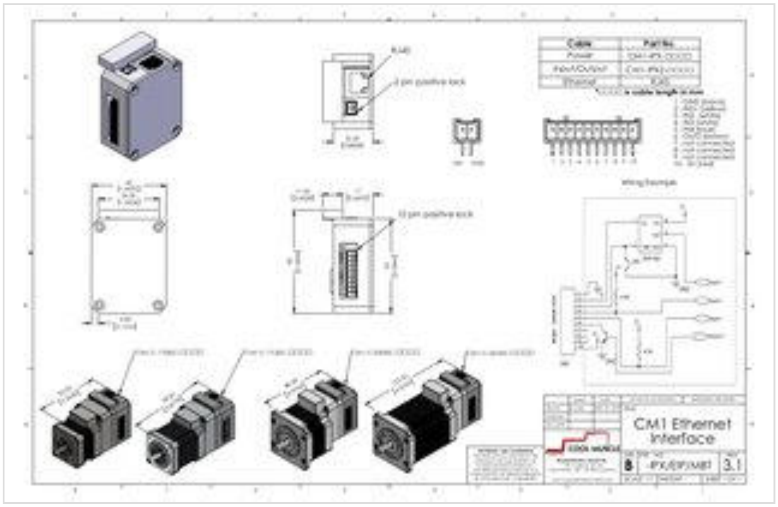


19.5 EIP

This EtherNet/IP interface has been deprecated. Please see CM1-T EtherNet/IP for the latest EtherNet/IP revision.

The EIP module allows the CM1 motor to communicate to a PLC or software using the common industrial protocol, Ethernet/IP. The EIP module shares a form factor with the IPX.

For more information on the EIP module contact Myostat support.



20 Control Room

Control Room is the software used for programming and controlling CoolMuscle motors. Control Room allows you to write and download programs to the cool muscle motor, including some advanced programming options. In addition to programming a motor, Control Room allows you to upload programs and parameters from a motor as well as monitor motor functions and activities in real time.

Title	Size	Revision	Revision Date
Control Room V1027.msi	23.2MB	V1.0.27	14 Feb 2025

20.1 CML Programming Example

The following program is an example of a basic point to point motion executed in a program bank.

```
B100       //CLEAR EXISTING PROGRAM BANKS
L100       //CLEAR EXISTING LOGIC BANKS

A1=50      //ACCELERATION 1 SET TO 50
S1=50      //SPEED 1 SET TO 50
S2=100     //SPEED 2 SET TO 100
P1=0       //POSITION 1 SET TO 0
P2=1000    //POSITION 2 SET TO 1000


B1         //BEGIN PROGRAM BANK 1
A1,S1,P2   //MOVE TO POSITION 2 AT SPEED 1 AND ACCELERATION 1
S2,P1      //MOVE TO POSITION 1 AT SPEED 2 AND ACCELERATION 1
END        //END PROGRAM BANK 1

$          //SAVE PROGRAM TO EEPROM
```

In the program above, we first clear any existing banks to insure we are starting with a clean slate. We then declare all of our motion parameters. We set the necessary acceleration, speed, and position values. In program bank 1 there are two motions, the first is to position 2, then as soon as it reaches this point it moves to position 1 and the bank ends.

20.2 CML Vs. CMP

There are two types of programming available in control room. There are CML programming windows, and CMP programming windows. CML stands for Cool Muscle Language. This is the basic code that the motor understands and stores. CMP stands for Cool Muscle Program, and this is the basic structure of CML with some added features which can be written in control room and are compiled into CML before being sent to the motor. The added features of CMP programming are to make more complicated programs easier and faster to write, maintain, and troubleshoot.

20.2.1 Comments

In a CML program, comments can be added after a '/' character. These are limited to 30 characters and are not downloaded to the motor. The CMP file improves on the commenting structure in two ways: addition of multi-line comments, and removal of a character limit.

A single line comment can be written similar to CML by using a '//' before the comment which will end at the next carriage return. You can also write a multi-line comment by preceding with a '/*' and ending with '*/'. CMP comments are also colored green in order to differentiate them from the rest of the program. Neither of these comment structures are character limited. These comments are not compiled or sent to the motor.

For example:

```
//This is a single line comment. It will only be terminated when the line ends. – we should show that
//because this is no on another line it is not a comment.


/*This is a multi-line comment.
It can have as many lines as
you want until it is terminated */
```

20.2.2 Definition of Constants

A CMP program also allows you to define a constant to be used elsewhere in the program. This allows you to have a name which you can easily reference multiple times in a program, as well as being able to change a constant in one location and have it automatically change throughout the program. The following is an example of a constant definition:

```
#define fast 100

#define slow 10
```

This definition will allow you to use the names 'fast' and 'slow' throughout your program, and when it is compiled these instances will be replaced with the numbers 100 and 10 respectively. The define statements are also colored blue in the CMP window to differentiate them from the rest of the program.

20.2.3 Named Variables

A named variable is a read and write variable which is assigned to a specific motor parameter. This allows you to name specific register values to more easily write and troubleshoot a program. The following is an example of a named variable:

```
var Home P1

var HighSpeed S1
```

In the above example we have defined the P1 register as "Home" and the S1 register as "HighSpeed". These names can be used throughout the program and when compiled they will reference the defined registers.

20.2.4 Numbers Within a Bank

In a normal CML program, a number cannot be used within a bank; for instance:

P2=P1+100

Would not work, you would have to define a different register as 100 and call that from within the bank. By using a CMP file, control room will do this for you automatically. You can in fact write the above statement in a CMP file. When compiled, the software will automatically assign an unused register as the desired number and use that register in the program sent to the motor. If you have used all available register addresses, control room will give you an error.

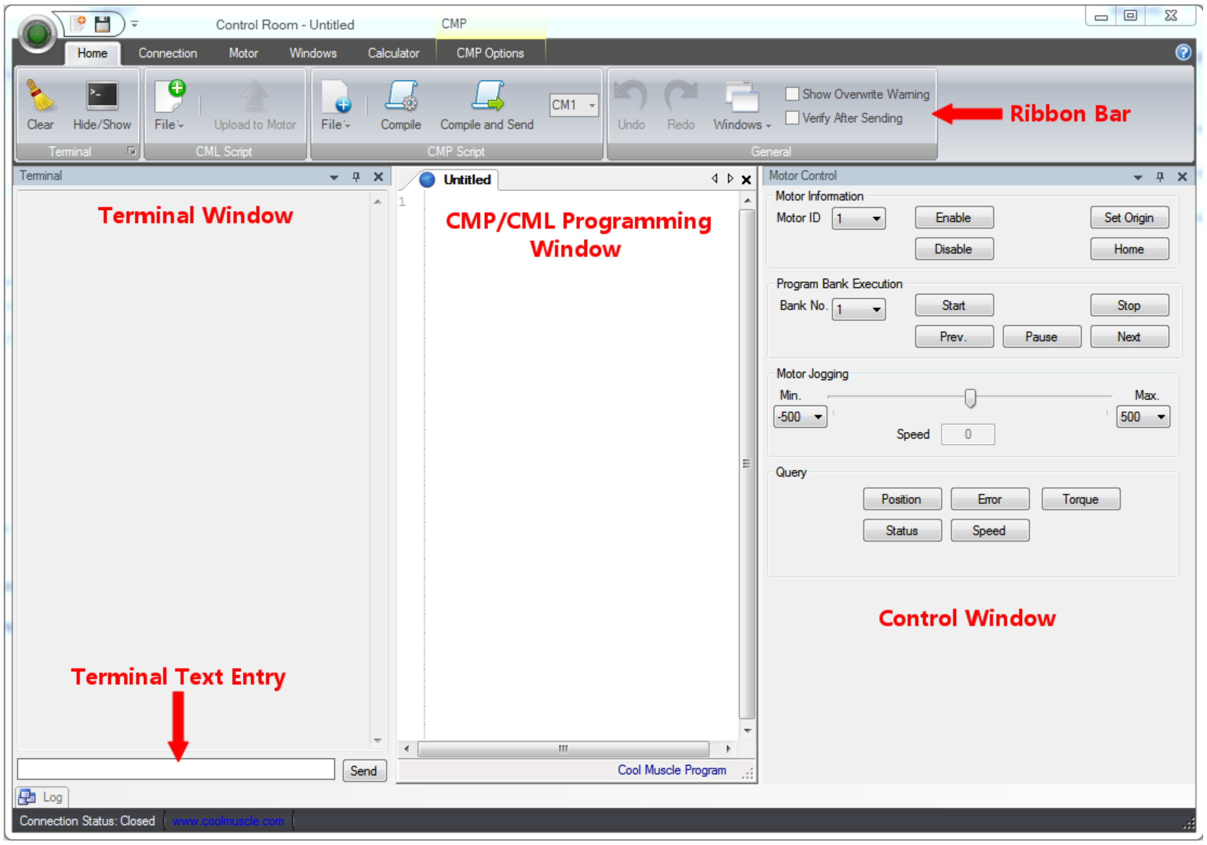
20.3 Remote Support

Support Engineers at Myostat are able to provide remote support sessions via AnyDesk. If you are having trouble and you would like to arrange a support session, please contact Myostat Technical Support at 1-877-696-7828

You can download the support client here: Myostat Client Support.exe

20.4 User Interface

The Control Room interface is made up of only a few elements.

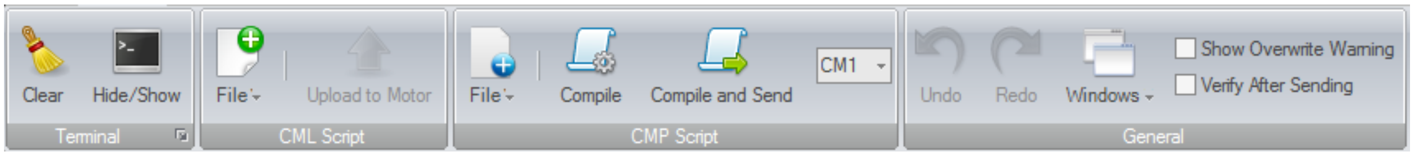


19 Control Room User Interface

20.4.1 Ribbon Bar

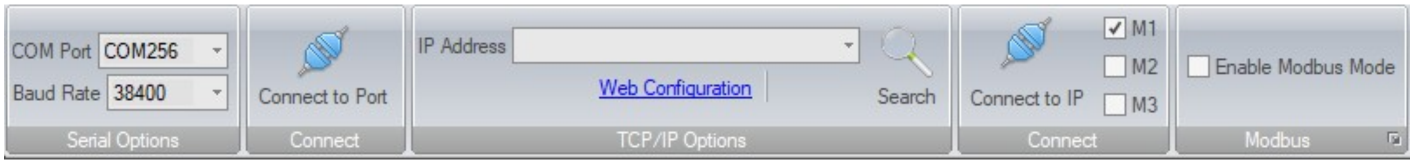
The Ribbon bar along the top of the program is similar to most other Windows software in that this is where you will find the shortcuts to the majority of the functions of the program. This is where most of the important elements will be found. There are 6 different ribbons available from the tabs along the top.

Home



The Home ribbon contains most of the basic functions you will need. The first two buttons allow you to clear any text from the terminal window, or show/hide the terminal window. The next two sections allow you to create a new file or upload a file to a motor, depending on whether you are editing a CMP file or a CML file. For more information on this, see *CML vs. CMP*. The final section are some general options, such as undo and redo, as well as the window button which allows you to open or close windows individually. There are also two selection boxes: the top one allows you to select to show a warning when overwriting a program on a motor, as well as an option to have the software verify a program after it is sent to the motor to ensure that everything was received correctly.

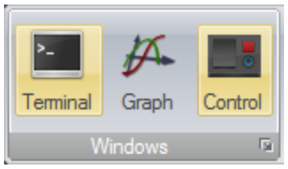
Connection



The connection Ribbon will be the first ribbon you see when starting Control Room. This ribbon contains the options for connecting to a motor. There are two methods for connecting to a motor: via serial port, or via TCP/IP. The first option allows you to select a COM port and baud rate, and connect to port.

In order to connect to a motor via TCP/IP using an IPX module, you can enter the IP address if it is known and press connect to IP. If the IP address is dynamic or unknown, you can also search for the IP as long as it is on the same network as the computer. Selecting web configuration will allow you access to the advanced configuration menu of the IPX module. You should not need to edit these options by default.

Windows

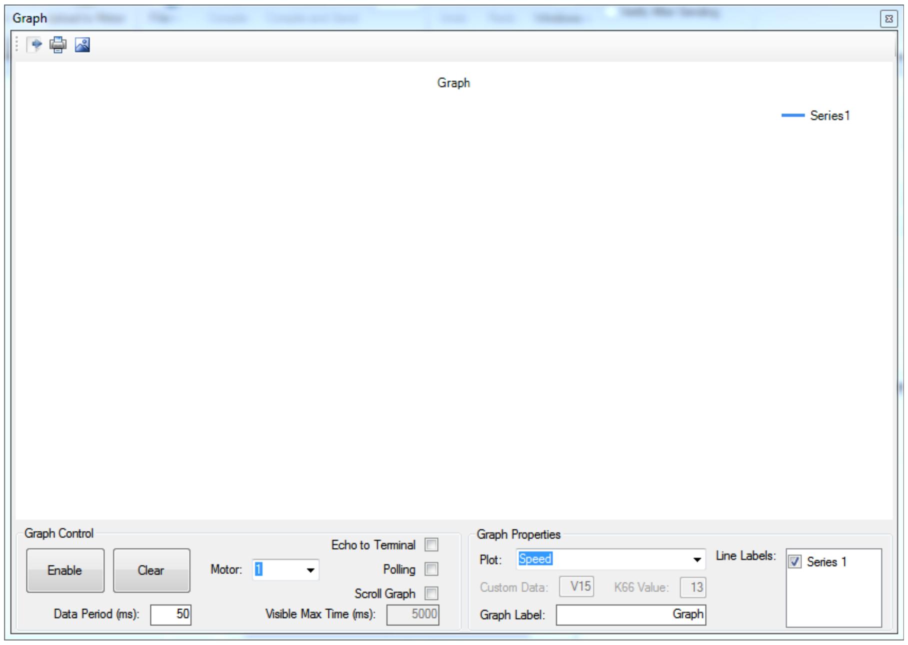


The windows ribbon allows you to turn on or off the various windows.

Terminal

The terminal window is the main path of communication for messages sent from and to the motor. Any information transmitted from the motor will appear in the terminal window. You can also use the terminal text entry to send single line commands to the motor, one at a time. If you want to query anything from the motor, you can send that command here and the response from the motor will be displayed in the above terminal window.

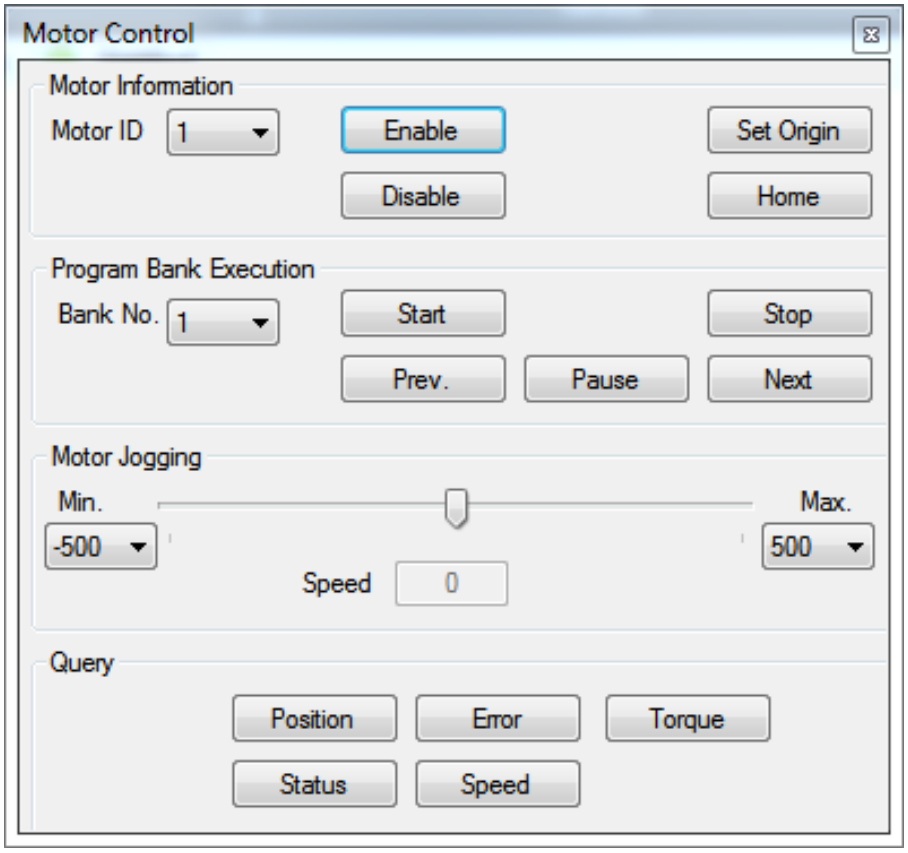
Graph



The graph window allows you to get real time information from the motor. By selecting speed, position, or torque from the plot menu and selecting enable, the software will graph those values from the motor. There is also an option to graph a custom value, such as a variable in the motor which you want feedback on. You can also graph other ranges of values by selecting the K66 option. The time between graph points in milliseconds can be adjusted by changing the value in the data period field. 10ms is the lowest recommended value.

In the top left corner of the graph window, you can also outputs the graph as an image, print the graph, or output the graph as a set of comma separated values for use in a spreadsheet program.

Control

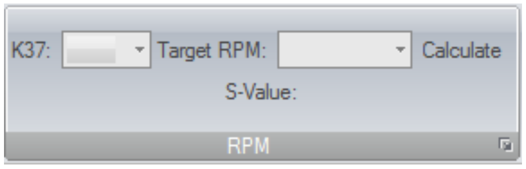


The control window presents a set of commonly used simple controls and queries as easy to use buttons. You can perform functions such as enable/disable the motor, as well as setting the origin, homing, stepping through a program bank or jogging the motor in either direction with the an easy to use interface. The query section also allows you to check the current position, speed, torque, position error, and status. The responses from the queries will appear in the terminal window.

CMP/CML

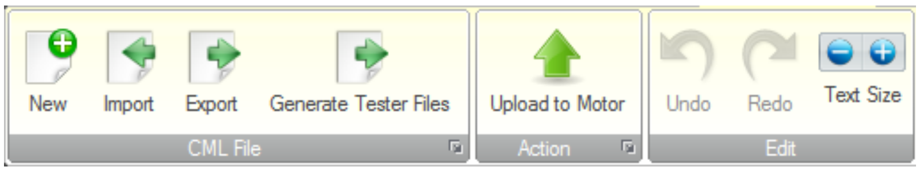
The CMP/CML window is window that you will probably spend the most time in. This is the area in which you will write any programs to be downloaded to a motor. The text size in the window can be changed by holding in the CTRL key and moving the mouse wheel up or down. In the case of a CMP window, this will also allow you to write comments, named variables and other advanced functions of a CMP program. For more details, see *CML vs. CMP*.

Calculator



The calculator ribbon is a tool to help you calculate a desired speed value. First select the value of K37 you are using and then enter the speed value you are looking to achieve in RPM. When you press calculate, the software will output the S parameter value that you should use to achieve this.

CMP/CML Options



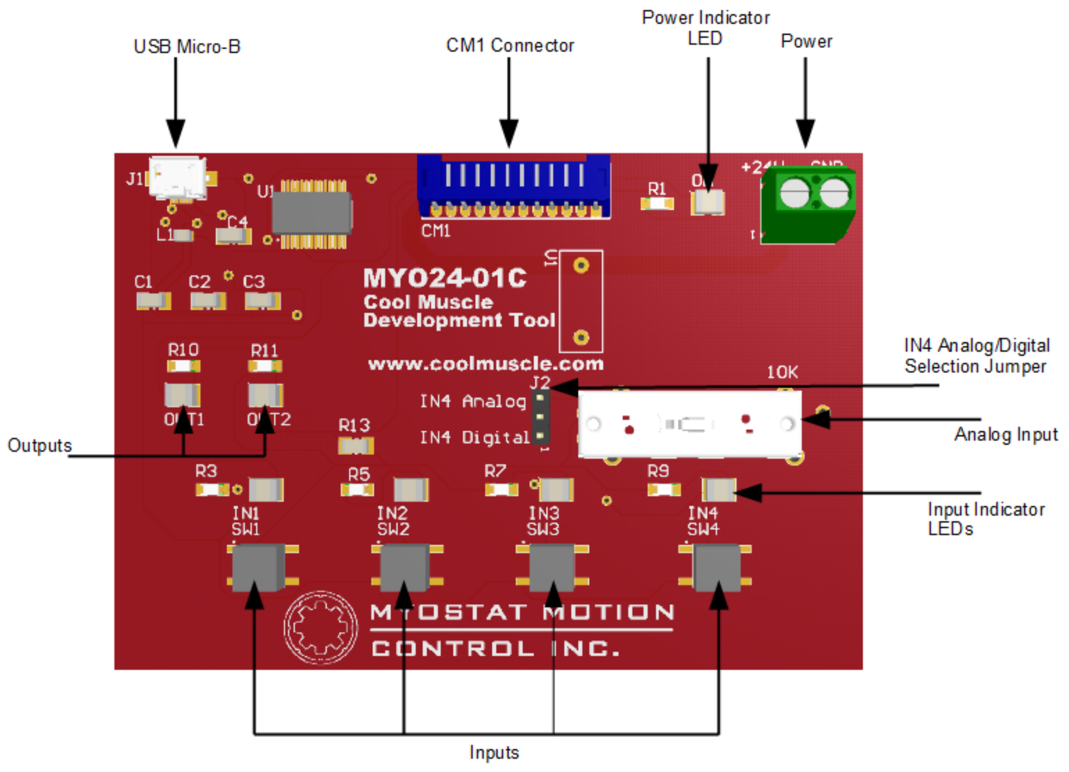
A final ribbon tab will appear if you are editing a CMP or CML file depending on which you are using. There are different options depending on which type of file you are editing. Each ribbon will allow you to import or export a file or create a new file. Undo, redo, and the text size options are also available for each type of file. The CMP ribbon will allow you to compile the file for a CM1 or CM2 motor as well as send it to the connected motor. The CML ribbon will allow you to upload the file to a connected motor, but it will also allow you to generate test files if you are using a CM2 motor diagnostic tool.

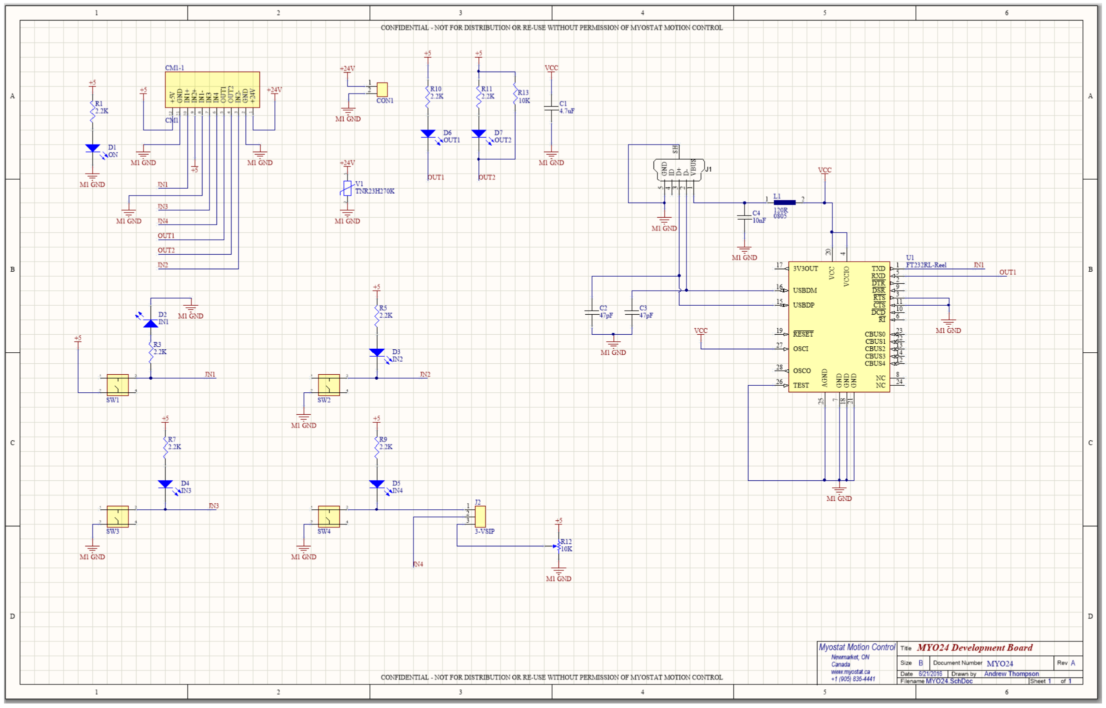
21 CM1 Development Tool - MYO24

21.1 Overview

The MYO24 CM1 development tool is meant to be used as a simple and convenient method for testing programming and simulating inputs and outputs on the CM1 motor. The tool comes equipped with a USB converter for easy communications, four input buttons for simulating sensors and other inputs, a linear potentiometer for simulating an analog input, and two LEDs for showing the state of the motor outputs. The MYO24 connects to the motor easily using a 12-pin straight through cable.

The MYO24 is powered by the same 24VDC power supply as the motor and features a varistor on-board to help reduce or eliminate voltage spikes from the motor. Power is connected to the screw terminal on board and is transferred to the motor through the 12-pin connector. There is no need for a separate power supply or separate wiring for the motor power.




CONFIDENTIAL - NOT FOR DISTRIBUTION OR RE-USE WITHOUT PERMISSION OF MYOSTAT MOTION CONTROL
Myostat Motion Control
MYO24 Development Board
MYO24
Andrew Thompson
FT232RL-Reel

21.2 Connecting the MYO24

21.2.1 Power

The MYO24 is powered off of a 24VDC power supply and will pass power through to the connected motor. The components on the MYO24 are not powered from 24V but from the motor's 5VDC supply, so observe the maximum current draw for the motor you are using for proper power supply sizing.

Connect the output and ground from your power supply to the screw terminals labelled +24V and GND respectively.

It is recommended to not connect and/or disconnect the power connector, or the CM1 connector with power applied. Wait until all connections are made before powering up your power supply.

The Power indicator LED will light when the motor is powered up. This LED is powered from the motor, and not directly from the 24V power supply. This means there may be 24V on the MYO24 board, but the power indicator will not be lit if there is no motor connected, or if the motor is damaged or not functioning.

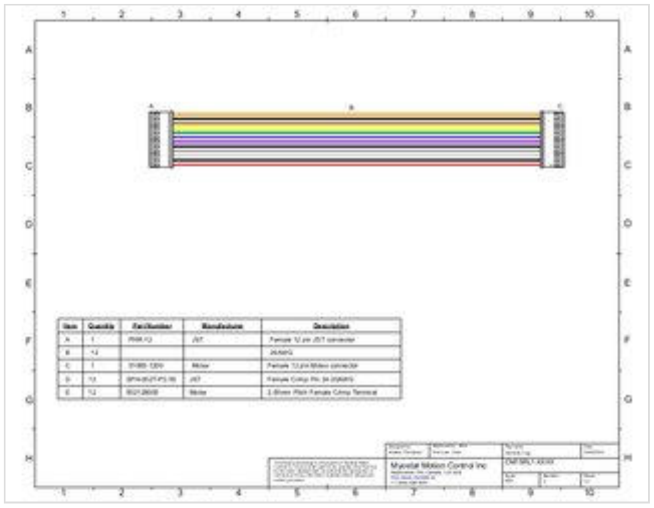
21.2.2 Communications

The MYO24 features a single USB Micro-B plug for connecting to your computer. Plug the cable from a USB plug on your computer to the USB Micro-B connector on the MYO24. The MYO24 should be automatically detected in windows and be assigned a COM port. If the drivers are not automatically installed, you can download the drivers here: http://www.ftdichip.com/Drivers/VCP.htm

In order to communicate to the CM1 motor, you will want to install Control Room.

21.2.3 Connection to CM1 Motor

The MYO24 connects to the CM1 motor through the 12-pin connector on the top of the board. See the drawing for the required CM1SRL1 cable below:



Item	Quantity	Part Number	Manufacturer	Description
A	1	PHR-12	JST	Female 12 pin JST connector
B	12			26AWG
C	1	51353-1200	Molex	Female 12 pin Molex connector
D	12	SPH-002T-P0.5S	JST	Female Crimp Pin 24-28AWG
E	12	[illegible]	Molex	2.00mm Pitch Female Crimp Terminal

21.3 Inputs

The MYO24 features a tactile button for each input. Because Input 1 is used on the CM1 motor for communications, the IN1 button will only function if there is no USB cable connected to the MYO24 board.

Above each button is an LED. This LED will show that the input button is activated. This LED is not an indicator of anything from the motor, but merely that the button is depressed and is functioning. If the LED is lit, the motor should be getting the activation signal from that button.

Input 4 features a jumper pin to select either the tactile button, or a sliding potentiometer. This is used to simulate the analog input on the motor if needed. To select the potentiometer, set the jumper to IN4 Analog. To use the tactile button, set the jumper to IN4 Digital. The potentiometer will vary the voltage in to input 4 from 0 to 4.5V.

21.4 Outputs

The MYO24 features an LED for each of the two outputs on the CM1 motor. Because output 1 is used on the CM1 motor for communications, the OUT1 LED will only function if there is no USB cable connected to the MYO24 board.

Each LED will activate when the corresponding motor output is activated.

22 CM1 Certifications

22.1 Certificates

22.1.1 ISO9001

ISO 2001_2015 Myostat.pdf

22.1.2 CE Declaration

CM1 CE Declaration of Conformity

22.1.3 RoHS Compliance

CM1 Certificate of RoHS Compliance.pdf