MC3B
3-channel stepper motor controller/driver

USER'S GUIDE

Rev 1.0
June 2007

RW Automation, LLC

www.rwautomation.com

Table of Contents

1) Legal Notices

1.1) Copyright

All material presented in this user's manual is subject to the copyright laws of the United States of America.

RW Automation, LLC reserves the right to make changes and improvements to its products and to this document without notification.

1.2) License agreement

The software source code and executables packaged with the MC3B motor controller board are intended for use only with the MC3B motor controller board. Any other use of the software is strictly forbidden without written permission of RW Automation, LLC. The source code and executables packaged with the MC3B motor controller board may be altered and recompiled into new executables. Such executables based on the original software packaged with the MC3B motor controller board may be freely distributed for resale along with the MC3B motor controller board, but may otherwise not be distributed without the MC3B motor controller board.

1.3) Limited warranty

RW Automation, LLC warrants the MC3B motor controller board and associated software to be free from defects in material and workmanship and to perform to applicable published RW Automation, LLC specifications for a period of one year from the date of shipment to the purchaser.

RW Automation, LLC will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. RW Automation, LLC shall have the right of final determination as the existence and cause of defect.

In no event shall RW Automation, LLC be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, RW AUTOMATION, LLC MAKE NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. RW AUTOMATION, LLC WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEROF.

1.4) Third-party trademarks

“Microsoft”, “Microsoft Visual C++ 6.0” and “Microsoft Windows” are registered trademarks of Microsoft Corporation.

2) Introduction

Congratulations on your purchase of the MC3B motor controller board. Please take the time to familiarize yourself with the material and features of the MC3B motor controller board package before proceeding to the next section.

2.1) Packing list

When you purchase the MC3B motor controller board, you should receive the following items:

a) The MC3B motor controller board
b) Three 4-connector Phoenix terminal blocks (one for each motor)
c) One 2-connector Phoenix terminal block (for power)
d) One 8-connector Phoenix terminal block (for limit switches)
e) CD-ROM (with motor control library and sample application)
f) RS-232 serial cable

2.2) Features

a) RS-232 port to connect to a host computer. The baud rate is fixed at 19200.
b) Concise ASCII command set.
c) Controls three independent stepper motors (bipolar or unipolar).
d) User programmable current limits (0.0 to 2.0 Amperes per phase, independent for each motor).
e) Up to 5000 steps/second.
f) Full stepping or half stepping.
g) Free-running modes allow on-the-fly changes to velocity, direction, and acceleration, enabling manually controlled operation.
h) Relative moves up to $\pm$ 4,294,967,295 steps.
i) Fully synchronous 3-axis linear motions. Also allows up to 100 synchronous motions to be buffered to create complex continuous motions.
j) Six user-definable limit switches with homing algorithm.
k) User-definable independent acceleration and deceleration for each motor.
l) Detachable terminal block connectors for quick connect/disconnect.
m) Single supply input variable from 7 – 40VDC.
n) Power and status LEDs.

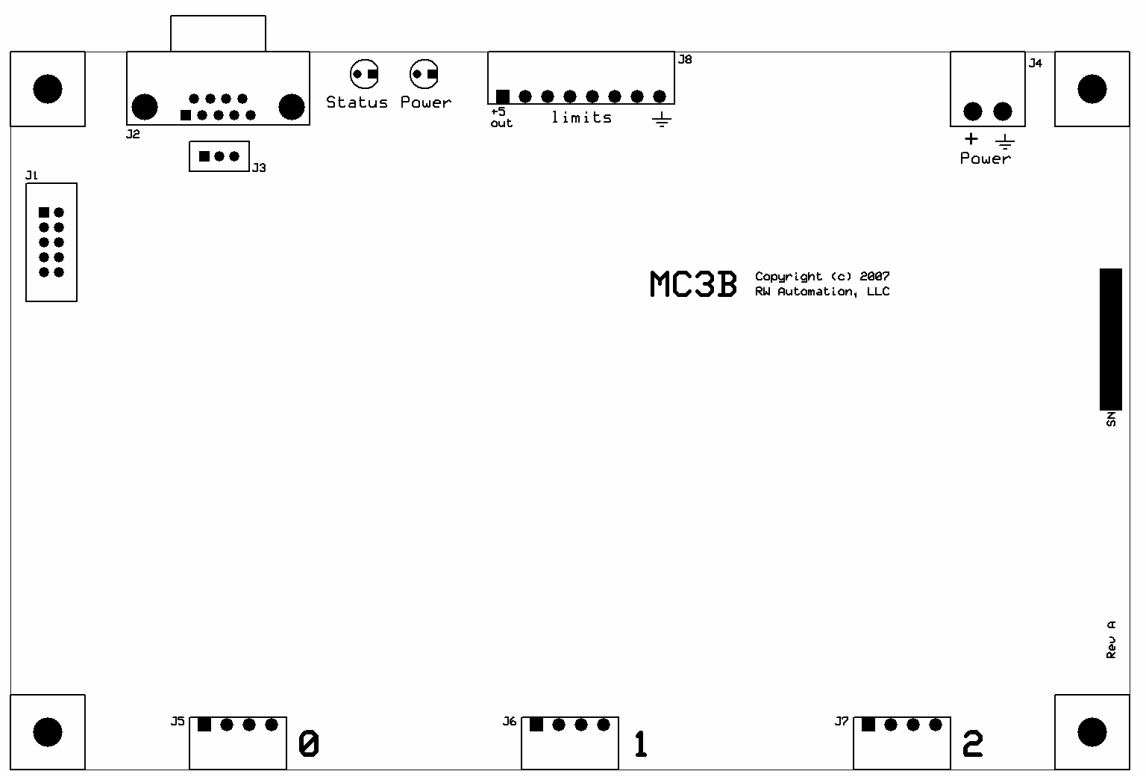
In addition, a software library and motor control application program (including source code) that can be built under Microsoft Visual C++ 6.0 is included at no additional charge.

3) Configuration

This section describes the steps needed to set up the MC3B motor controller board before you will be able to use it.

Connector placement is show in figure 1. Note that connector faces are flush with the perimeter of the board.

Figure 1 – Overview of connector locations



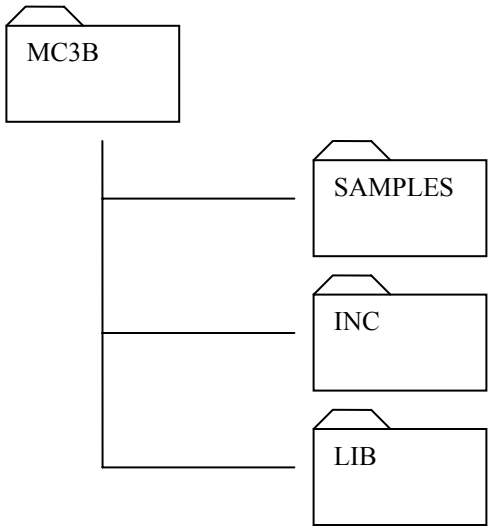
The connectors and LEDs shown in figure 1 are listed in the following table:

Component	Description
J1	Reserved
J2	RS-232 serial port connector
J3	Reserved, jumper from pins 2-3 to enable RS-232 communications.
J4	7 to 40 VDC connector
J5	Motor 0 connector
J6	Motor 1 connector
J7	Motor 2 connector
J8	Limit input connector
Power	Power LED
Status	Status LED

3.1) Installing the software

The MC3B software runs under Microsoft Windows 98/2000/XP operating systems. It is distributed as a single self-extracting setup executable file. Simply run the setup file and follow the instructions. The MC3B library software, sample application, this document, and an uninstall utility are all provided in the distribution file.

Once installed, the software will have the following directory structure:



The SAMPLES sub-directory holds the sample application executable, as well as the full source code and project files for building the application under Microsoft Visual C++ 6.0. These projects may be imported to Visual Studio 2005 with minimal effort.

The INC sub-directory holds the header files user applications will need to include with their projects to use the library.

The LIB sub-directory contains the .lib (static library) source files and project files for the motor control library files. Pre-built static libraries for the normal and multi-threaded builds of the motor control library are included.

Refer to sections 4 and 5 for more details regarding the use of the software after it has been installed.

3.2) Host computer communications

Figure 2 below illustrates an example of how to connect the MC3B motor controller board (the “slave") to a host PC (the “master"). This particular example shows the use of a 9-pin female-male cable wired straight through. This cable configuration is appropriate for most recently manufactured PCs, however, older PCs and other types of computers may have different connectors (often 25-pin) and pinouts. In such cases, it is up to the user to purchase whatever additional null modem, gender changer, and adapter cables are needed to suit the requirements of the MC3B motor controller board.

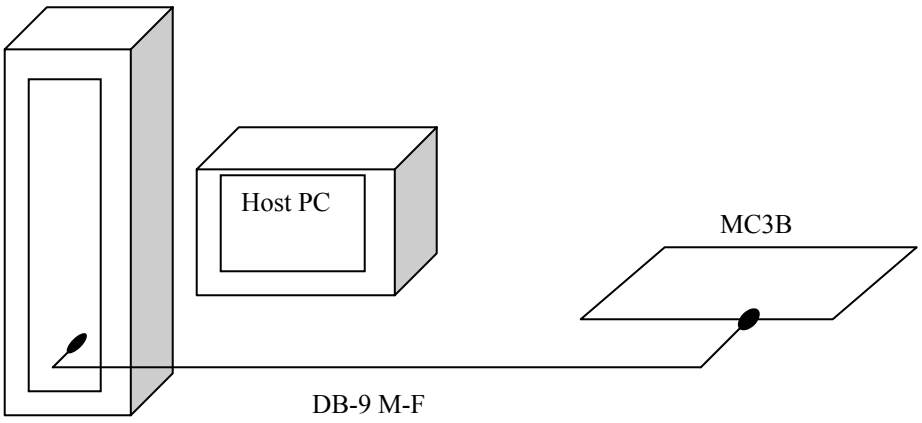
Below is the pinout of the RS-232 connector of the MC3B motor controller board used to connect the MC3B board to a host PC. Note that flow control lines are not supported by the MC3B motor controller board.

RS-232 “master” connector pinout

Pin	Function
1	Loopback to pins 4 and 6
2	Receive
3	Transmit
4	Loopback to pins 1 and 6
5	Ground
6	Loopback to pins 1 and 4
7	Loopback to pin 8
8	Loopback to pin 7
9	Not connected

The RS-232 parameters are: 19200 baud, 8-bits, no parity, one stop bit.

Figure 2 – Host PC to MC-3B connection



3.3) Connecting power

The MC3B motor controller board requires a single voltage source in the range 7 to 40 VDC. In addition to supplying power to the stepper motors, on-board regulators are used to supply the +5 VDC and +3.3 VDC needed for the microprocessor and other digital logic integrated circuits. The output of the +5VDC regulator is also available to the user in limited quantities via the limit switch connector.

Power connector pinout

Pin	Function
1	Ground
2	+7 to +40 VDC

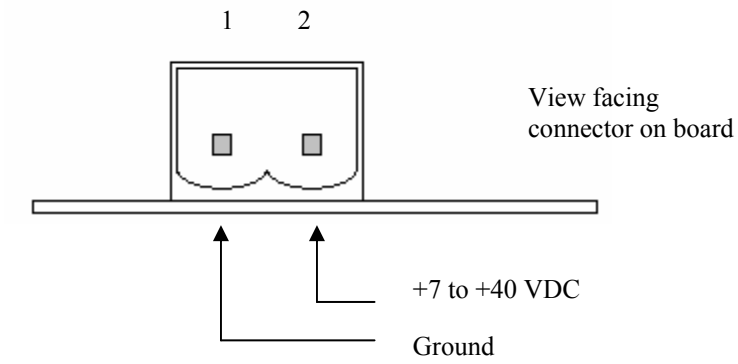
If the three stepper motors are to be run simultaneously at the maximum current (2 Amperes per phase), then the supply must be able to support a minimum of 13 Amperes of continuous current.

Figure 3 below shows the 2-pin power connector illustrating how to connect power to the MC3B motor controller board.

WARNING: Please note carefully the required polarity of the power supply connector. Applying power of the incorrect polarity can damage the MC3B motor controller board and will void the warranty.

WARNING: Applying voltages outside of the required 7-40 VDC range can damage the MC3B motor controller board and will void the warranty.

Figure 3 – Power connector



3.4) Connecting motors

Each motor is assigned a logical number from 0 to 2 as shown in figure 1. All three motor ports are wired identically. Figure 4a illustrates the motor connector and the pinout is shown on the table below

Motor connector pinout

Pin	Function
1	Phase A+
2	Phase A-
3	Phase B+
4	Phase B-

Figure 4b below illustrates the connection of a bipolar (4-lead) motor to one of the motor ports on the MC3B motor controller board. The direction that the motor will turn in (clockwise or counterclockwise) is a function of how the leads for each coil of the motor are attached to the motor port and the arbitrary mapping between the logical values of 0 and 1 assigned to the direction. If, for instance, your application requires that direction 0 map to a counterclockwise rotation, and you find that, after connecting your motor, direction 1 maps to the counterclockwise rotation, you can resolve the problem by reversing the leads of *one phase* of the motor.

Figure 4c illustrates the connection of a unipolar (6 lead) motor to one of the motor ports on the MC3B motor controller board. As can be seen, the connections are identical to those for a bipolar motor, except that the coil center tap leads are left unused.

Figure 4a – Motor connector

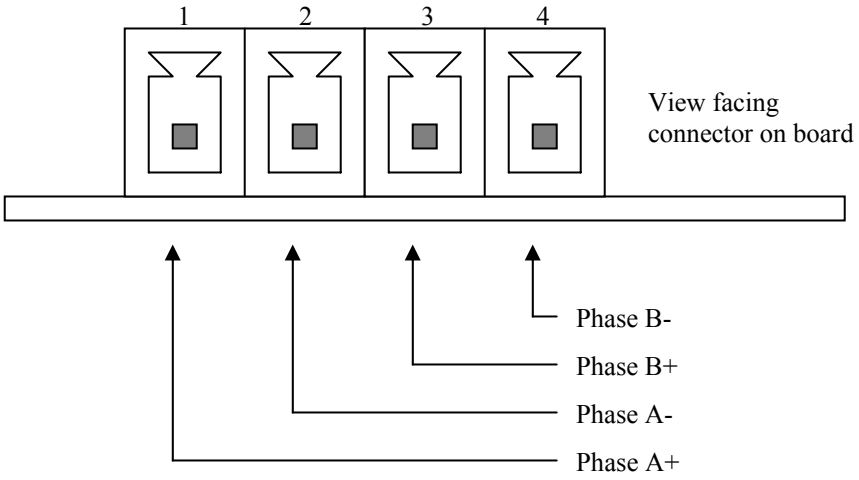


Figure 4b – Bipiloar motor

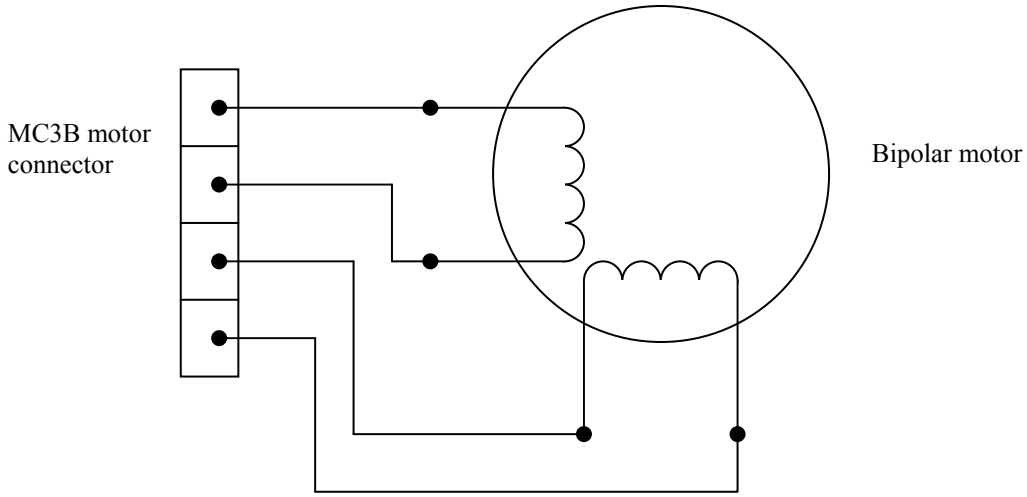
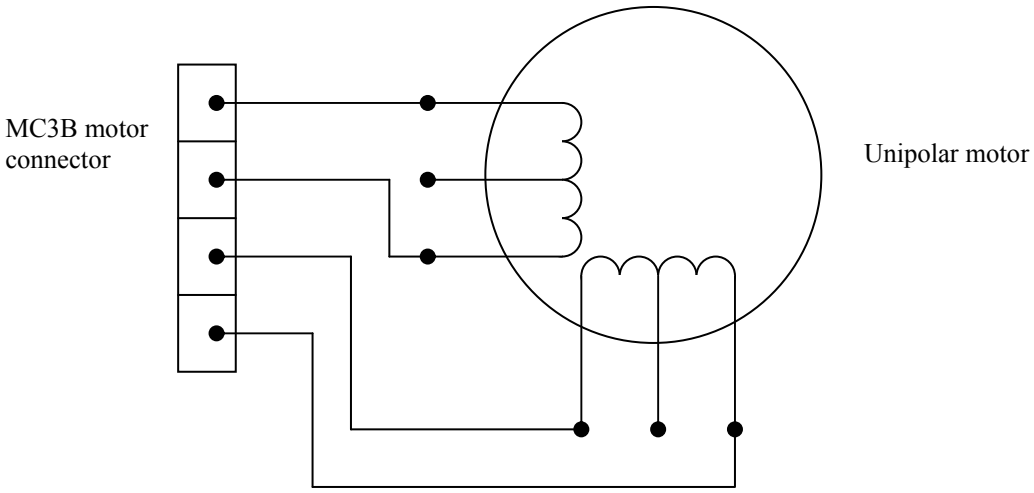


Figure 4c – Unipolor motor



3.5) Connecting limit switches

The MC3B motor controller board has a 6-input limit switch connector. The pinout for the connector is shown in below.

Limit switch connector pinout

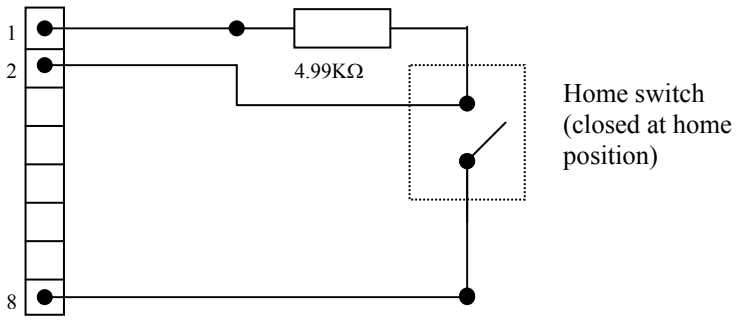
Pin	Function
1	+5V
2	Input 0
3	Input 1
4	Input 2
5	Input 3
6	Input 4
7	Input 5
8	Ground

The limit switch inputs use 3.3V gates (5V tolerant). Do not exceed the 0-5VDC input range of the limit switch inputs.

The limit switches may be logically mapped by the user to any of the motors/directions as described in sections 4 and 6. A simple means of implementing a limit switch is shown in figure 5. In such a configuration, the motor will cause the closure of a switch that will ground the input pin when the device actuated by the motor reaches a home position.

A limited +5 VDC supply is available on this connector for users that need to power optical or hall-effect limit switches. Note: no more than 100 milliamperes should be drawn from the +5 VDC pin of this connector.

Figure 5 – Limit switch example



3.6) Status LEDs

The MC3B motor controller board has two status LEDs (refer to figure 1).

The power LED is activated whenever the on-board +3.3 VDC regulator is active.

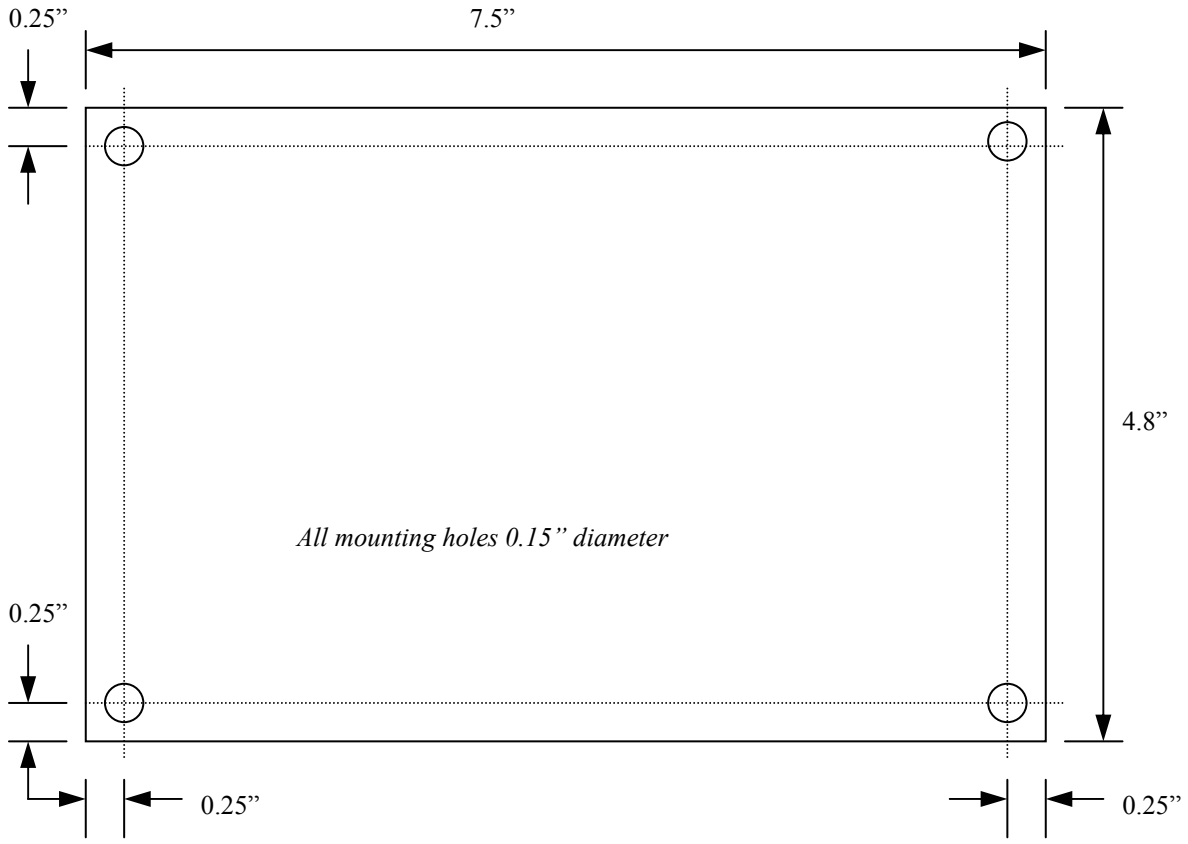
The motor status LED will be on whenever there is a current being applied to any motor (including holding currents). The LED is off when no current is being applied to any motor.

3.7) Mechanical specifications

The MC3B motor controller board is 7.5” x 4.8” (190.5mm x 121.92mm). The height of the heatsinks is 1.5” (38.1 mm).

Mounting hole locations are shown in figure 6 below

Figure 6 – Mounting hole locations (not drawn to scale)



4) Motor Control Library

The motor control library is a software library of C functions that can be compiled under Microsoft Visual C++ 6.0 or visual studio 2007.

These functions provide an applications programming interface (API) for the motor controller without the need to learn the MC3B motor controller board serial communication protocol or the API for the serial port.

The source code is included to allow the interested user to more fully understand how to use the serial communication protocol (see section 6). As well, the source code makes it possible to port the library to platforms other than those supported under Microsoft Visual C++ 6.0.

The subsections that follow give details of each of the library API calls and their usage. Each of the API functions will return an integer status code. The codes are enumerated as follows:

0 NO ERROR

This is the status normally returned by an API call. It indicates that the API call was completed successfully.

1 SERIAL PORT ERROR

This will occur when a low-level serial port error occurs. This will occur, for instance, when an API call specifies the use of a serial port that does not exist, or is in use by another application.

2 NOT INITIALIZED ERROR

This will occur when calling an API function before the library is initialized by calling the `mcl_initialize()` function.

3 TIMEOUT ERROR

This will occur when an attempt to communicate with the MC3B motor controller board fails. This will normally only occur if there is a problem with the connection between the host computer and the MC3B motor controller board.

4 INVALID COMMAND ERROR

This indicates that a command was received that the MC3B did not recognize. This may indicate data corruption in the serial communications.

5 MOTOR RUNNING ERROR

This indicates that a command was rejected by the MC3B motor controller because the motor that the command affects is running.

6 WRONG NUMBER OF PARAMTERS ERROR

This indicates that a command was received by the MC3B that did not have the correct number of parameters. This may indicate data corruption in the serial communications.

7 PARAMETER OUT-OF-RANGE ERROR

This indicates that a value for a parameter has been passed to an API function that is outside of the range expected for that function.

4.1) Get library version

```
int mcl_get_library_version(int *version)
```

The calling function must pass this function an integer pointer. The version number of the library is put into memory at the address held by the pointer. The two least significant digits of the version value are used for the minor version number, the rest of the digits are used for the major version number. For example, library version 1.00 is represented by a value of 100.

4.2) Initialize

```
int mcl_initialize(int com_port,
                   int *number_of_controllers)
```

The user should call this initialization function only once at the start of the program to initialize the motor controller library. Most of the other API functions will return a “not initialized” status if this function is not called first.

The `com_port` parameter specifies the RS-232 serial port, where 1 is used for COM1, 2 for COM2, etc.

The `mcl_initialize()` function will attempt to open the specified serial port and will then send out an initialization command to the MC3B motor controller board connected to the port. If the motor controller board is currently running, it will be reset to its default conditions (see section 6).

The user must pass a pointer, `*number_of_controllers`, to the `mcl_initialize()` function. This function will then return, possibly after a few seconds of delay, and the number of RW Automation controller boards found will be placed into memory at the address passed in the `*number_of_controllers` pointer. The MC3B motor controller board does not support daisy-chaining, so if connected directly to the PC, the number of controllers will always be 1. Note that the controller IDs are assigned sequentially, starting from 0.

Once the `mcl_initialize()` function is called, calling it again will have no effect other than to simply return a “no error” status. If the user wants to specify a different com port at runtime, then the `mcl_deinitialize()` function must be called prior to calling the `mcl_initialize()` function again with the new com port parameter.

4.3) De-initialize

```
int mcl_deinitialize(void)
```

The `mcl_deinitialize()` API function is complementary to the `mcl_initialize()` function. The user's program should call this function when it is being terminated or when it is desired to release the serial port specified in the call to `mcl_initialize()`.

When `mcl_deinitialize()` is called, the MC3B motor controller board will be re-initialized (turning off all motors) and the library will return to its uninitialized state and most other API functions will return a "not initialized" status until `mcl_initialize()` is successfully called.

4.4) Set acceleration

```
int mcl_set_motor_acceleration(int controller_id,
                               int motor_id,
                               int acceleration)
```

This API function sets the acceleration for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

The acceleration is an integer in steps/second^2. It must be in the range 100 to 100000. The acceleration for a motor may not be set while the motor is running (except when it is in free-running mode).

4.5) Set deceleration

```
int mcl_set_motor_deceleration(int controller_id,
                               int motor_id,
                               int deceleration)
```

This API function sets the deceleration for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

The deceleration is an integer in steps/second^2. It must be in the range 100 to 100000. The deceleration for a motor may not be set while the motor is running (except when it is in free-running mode).

4.6) Set maximum velocity

```
int mcl_set_motor_maximum_velocity(int controller_id,
                                   int motor_id,
                                   int maximum_velocity,
                                   int direction_modifier = 0)
```

This API function sets the maximum velocity for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

The maximum velocity is an integer in steps/second. It must be in the range 1 to 5000. The maximum velocity for a motor may not be set while the motor is running (except when it is in free-running mode).

The optional direction modifier parameter is normally set to 0, but if set to 1 will cause free-running and homing motions to be run in the opposite of their specified direction. The direction modifier is useful for performing on-the-fly changes in direction while a motor is stepping in free-running mode (see section 4.12).

4.7) Set motor half-steps control

```
int mcl_set_motor_half_steps_control(int controller_id,
                                     int motor_id,
                                     bool half_steps)
```

This API function sets the half/full stepping control value for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

The `half_steps` value must be false if full steps are required or true if half steps are required.

The half-steps control value for a motor may not be set while the motor is running. The half-steps control value is applied to the motor immediately when this API function is called.

4.8) Set motor holding current control

```
int mcl_set_motor_holding_current_control(int controller_id
                                          int motor_id,
                                          bool holding_current)
```

This API function sets the holding current control value for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

If `holding_current` is true, then a holding current is applied when the motor is idle. If `holding_current` is false, then no holding current is applied to the motor when it is idle.

The holding current control value is applied to the motor immediately when this API function is called.

4.9) Set motor limits

```
int mcl_set_motor_limits(int controller_id,
                         int motor_id,
                         int direction_0_limit_bit,
                         bool direction_0_limit_level,
                         int direction_1_limit_bit,
                         bool direction_1_limit_level)
```

This API function sets the limit switch behavior for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

There are six limit switch inputs available as described in section 3.5. The limit switch bits are numbered from 0 to 5. Since each motor has two directions of motion (0 and 1), separate limit switches must be definable for each direction.

To set the limit switch parameters for direction 0, determine which bit is to be used based on the connector pinout of section 3.5. and set `direction_0_limit_bit` to that value. If the input will be pulled to +5V when the limit is reached, set the `direction_0_limit_level` to true, otherwise set it to false if the input will be pulled to 0V.

If there is no limit switch for direction 0, then `direction_0_limit_bit` must be set to –1.

Use the same process for determining how to set the limit bit and limit level for motor direction 1.

Once set, the specified motor will stop each time the specified limit bit reaches the specified limit level for the direction the motor is traveling in. Depending on the motion command used, a homing algorithm may be triggered when a limit switch is reached.

4.10) Set limit switch debounce

```
int mcl_set_limit_switch_debounce(int controller_id,
                                  int debounce_count)
```

This API function sets the debounce count for the home (limit) switches. The limit switches are sampled at 1KHz and any change of state to a limit switch input must be stable for the number of debounce counts specified before the MC3B will react to it. The debounce count must be in the range 1 to 100.

4.11) Set motor current limits

```
int mcl_set_motor_current_limits(int controller_id,
                                 int motor_id,
                                 int holding_current_limit,
                                 int running_current_limit)
```

This API function sets the current limits for the specified motor (in the range 0 to 2 for the MC3B motor controller board) and specified motor controller board.

The current limits are integer values in units of milliamperes and must be in the range 0 to 2000.

Typically, the user will set the holding current limit substantially lower than the running current limit in order to prevent the motor from overheating.

4.12) Run motor

```
int mcl_run_motor(int controller_id,
                  int motor_id,
                  int direction)
```

This API function “runs” the specified motor (in the range 0 to 2 for the MC3B motor controller board) of the specified motor controller board.

The motor will accelerate to the maximum velocity in the specified direction and then stay at the maximum velocity until the limit for the specified direction is reached (if any) or the motor is commanded to stop. In this “free running” mode, the user may command changes in velocity, direction, acceleration, and deceleration. The MC3B motor controller will adjust its velocity/direction accordingly as the user commands changes.

4.13) Move motor

```
int mcl_move_motor(int controller_id,
                   int motor_id,
                   int direction,
                   int steps)
```

This API function “moves” the specified motor (in the range 0 to 2 for the MC3B motor controller board) of the specified motor controller board.

The motor will accelerate to a velocity no greater than the maximum velocity specified for the motor (see section 4.6) and then decelerate to a stop. Once the motion is complete, the motor will have moved the number of steps specified in the direction specified.

The motion can be halted prematurely by a command to stop the motor, or if the limit for the specified direction is reached (if any).

4.14) Home motor

```
int mcl_home_motor(int controller_id,
                   int motor_id,
                   int direction)
```

This API function “homes” the specified motor (in the range 0 to 2 for the MC3B motor controller board) of the specified motor controller board.

The motor will accelerate to the maximum velocity and hold at that velocity until the limit for the specified direction is reached. Once the limit is reached, the motor will reverse direction and move at a rate of 10 steps/second until the limit switch returns to its “off home” level. The motor then changes direction again and moves forward at 10 steps/second until the limit switch is again at the “home” level.

The motion can be halted prematurely by a command to stop the motor.

4.15) Stop motor

```
int mcl_stop_motor(int controller_id,
                   int motor_id)
```

This API function stops the specified motor (in the range 0 to 2 for the MC3B motor controller board) of the specified motor controller board.

All motion will immediately cease for the specified motor. A holding current will be applied if previously set (see sections 4.8 and 4.11).

4.16) Synchronous move

```
int mcl_synchronous_move_motor(int controller_id,
                               int motor_0_steps,
                               int motor_0_direction,
                               int motor_1_steps,
                               int motor_1_direction,
                               int motor_2_steps,
                               int motor_2_direction,
                               bool execute = true)
```

This API function will execute a synchronous motion on the specified motor controller board. All three motors must be idle in order for this command to be executed.

When executed, all three motors are moved synchronously at velocities, accelerations, and decelerations that are calculated to allow all three motors to take the same amount of time to complete their specified steps (in the specified directions). This is done without exceeding the maximum velocity, acceleration, or deceleration for any of the three motors.

Thus, if the motors were driving a linear X-Y-Z stage, the stage would traverse a straight line.

The optional execute parameter determines whether the synchronous move is to be executed immediately (execute is true) or whether the move is to be buffered in the MC3B motor controller and executed later (execute is false) as part of a chained move.

4.17) Chained synchronous move

```
int mcl_chained_synchronous_move_motor(int controller_id,
                                       int number_of_moves,
                                       int *motor_0_steps,
                                       int *motor_0_direction,
                                       int *motor_1_steps,
                                       int *motor_1_direction,
                                       int *motor_2_steps,
                                       int *motor_2_direction);
```

This API command will execute a chained synchronous motion. A chained move consists of an array of synchronized linear motions that get executed one-after-another without the pauses between motions that would otherwise occur during during the command/response interactions between the MC3B motor controller board and the host PC.

The `number_of_moves` parameter must be in the range 1 to 100. The synchronous move steps and directions are passed in arrays for each of the three motors (refer to section 4.16 for details of each parameter).

The motor control library sends the MC3B a series of commands to fill its chained move buffer and then commands the MC3B to execute the chained motion.

4.18) Get motor status

```
int mcl_get_motor_status(int controller_id,
                         int motor_id,
                         int *motor_status)
```

This API function gets the motor status for the specified motor controller board and the specified motor (in the range 0 to 2 for the MC3B motor controller board).

The motor status is returned in the integer variable at the address supplied by the calling function in the `*motor_status` pointer.

The motor status is defined by the individual bits in the integer as follows:

bit 0:	0 if the motor is idle, 1 if the motor is in motion.
bit 1:	0 if the motor is not at the direction 0 limit, 1 if the motor is at the direction 0 limit.
bit 2:	0 if the motor is not at the direction 1 limit, 1 if the motor is at the direction 1 limit.
bits 3-31:	Reserved

This function may be used to poll the motor controller board for status to see when a motion has been completed.

4.19) Get limit switches

```
int mcl_get_limit_switches(int controller_id,
                           int *limits)
```

This API function gets the debounced limit switch bits for the specified motor controller board. The limit switch bits are held in the least significant six bits of the integer at the address passed by the calling function in the `*limits` pointer.

A value of 0 for any particular bit will indicate the corresponding input pin is being pulled to 0V. A value of 1 for any particular bit will indicated to corresponding input pin is being pull to 5V.

These bits may be used to debug limit switch configurations or for general-purpose TTL inputs.

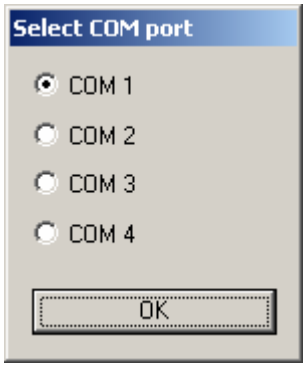
5) Sample Application

A sample application is provided that runs under Microsoft Windows 98/NT/2000/XP. The application provides convenient access to all of the motor controller's functions and can be used to manually control motors as an aid to setting up a system for automated control of the motors.

The application may be compiled under Microsoft Visual C++ 6.0. The source code is provided as an example of how the motor control library may be used.

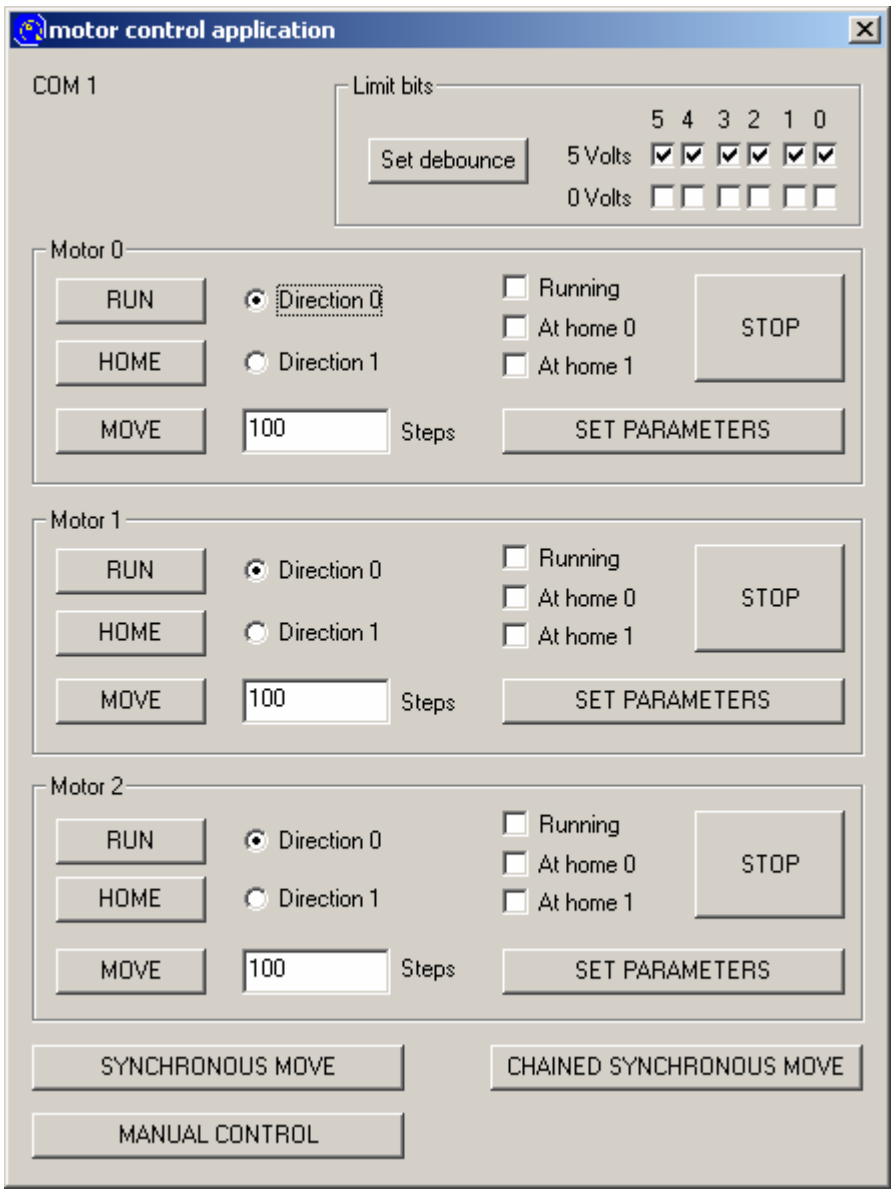
5.1) Communications port selection dialog

When the application is first started the user is prompted to select a communications port using the following dialog. The user must select the communication port corresponding to the one the MC3B motor controller board is connected to, and then press “OK”.



5.2) Control dialog

After the communications port is selected by the user, the application will search for and initialize the MC3B motor controller board found on the port. The following control panel dialog is then displayed and remains until the user closes the application.



The top portion of the dialog will display the communications port currently in use, the instantaneous state of the limit switch inputs (updated approximately once per second), and a button to set the debounce time for the limit switches.

The controls for controlling the motors are contained in separate identical group boxes (one per motor). The instantaneous motor status (Running, At home 0, At home 1) is updated continuously at a rate of approximately once per second. The user may change the motor direction and number of steps to move prior to executing a motion.

The “run”, “home”, and “move” buttons are used to start a motor. The “stop” button is used to stop a motor that is already in motion.

See section 5.3 for a description of the dialog that appears when the “set parameters” button is pressed.

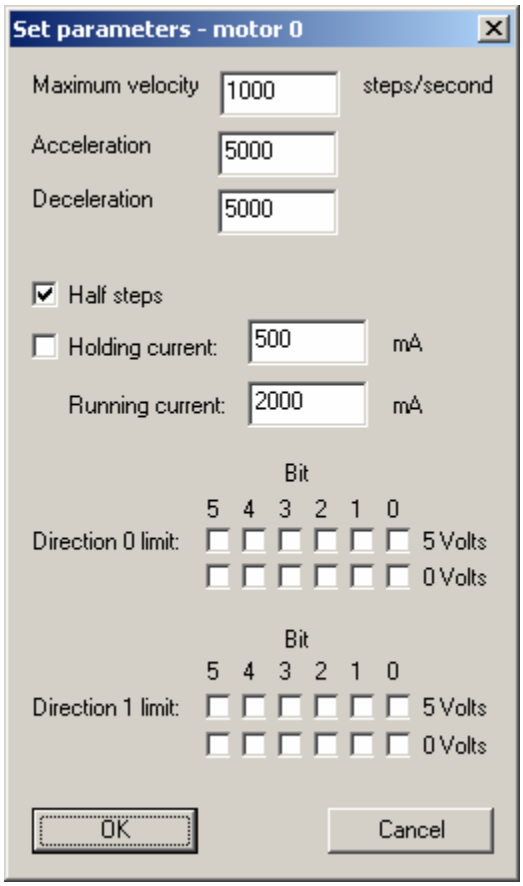
The “synchronous move” button will perform a synchronous move (see section 4.16) based on the parameters currently entered in the group box for each motor.

See section 5.4 for a description of the dialog that appears when the “chained synchronous move” button is pressed.

Finally, see section 5.5 for a description of the dialog that appears when the “manual control” button is pressed

5.3) Set parameters dialog

The following dialog appears when the “set parameters” button for a motor is pressed. Hence the controls in this dialog apply only to the motor corresponding to the “set parameter” button pressed.



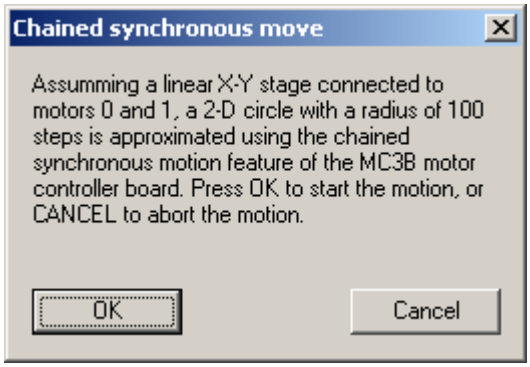
The user may set the maximum velocity, acceleration, and deceleration. The user may also select full or half stepping. If the holding current button is checked, the corresponding current limit is applied to the motor when it is idle. The running current limit may also be specified.

Only one of the twelve direction 0 limit boxes may be checked. By selecting one of the checkboxes, the input limit bit and the input limit level are specified (see section 4.9). If no check boxes are selected, then there is no home 0 position for the motor. The same applies for the direction 1 limits.

Pressing the “ok” button will cause the parameters to be applied immediately. Pressing the “cancel” button will discard any changes made to the parameters.

5.4) Chained synchronous move dialog

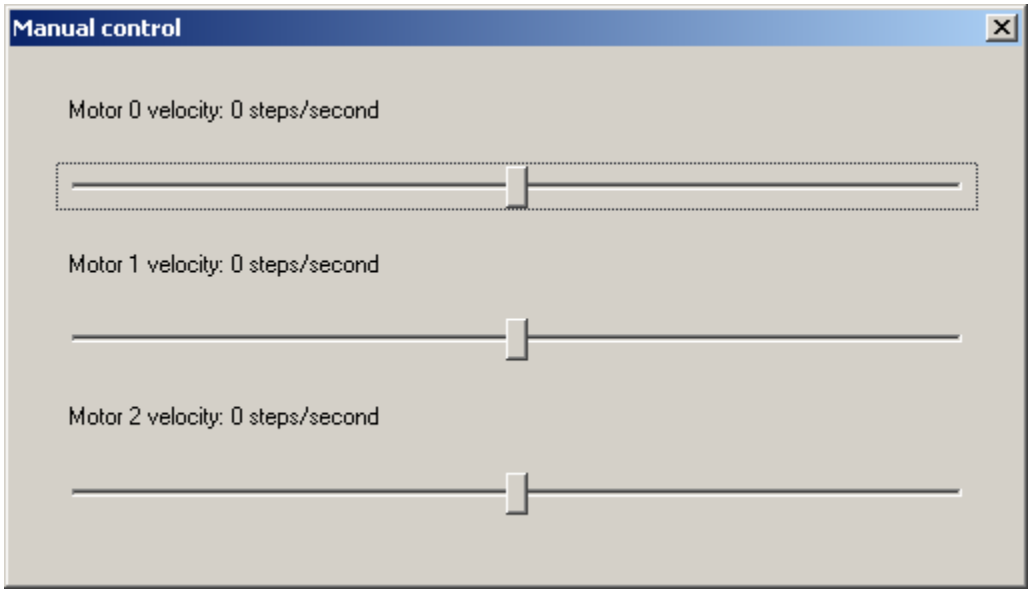
The following dialog appears when the “chained synchronous move” button is pressed.



As the dialog notes, once the OK button is pressed a 2-D circular motion is performed (assuming a linear X-Y stage). This feature illustrates the use of the chained synchronous moves in the MC3B motor controller board.

5.5) Manual control dialog

The following dialog appears when the “manual control” button is pressed.



The user may control the speed and direction of the motors (within the limits set in the main application dialog) by moving the slider controls for the individual motors. This features illustrates the ability of the MC3B to perform on-the-fly changes in velocity/direction when in free-running mode.

6) Serial Command Set

For those users wishing to control the MC3B directly from their own software without the use of the motor control library, the serial command set supported by the MC3B is listed below.

All commands are ASCII text (human readable). Thus, a communications program such as "Hyperterminal" included with most Microsoft Windows operating systems, may be used to directly control the MC3B.

Below is a screen shot of the Hyperterminal properties page. Shown are the settings required to allow communications between the host PC and the MC3B motor controller.

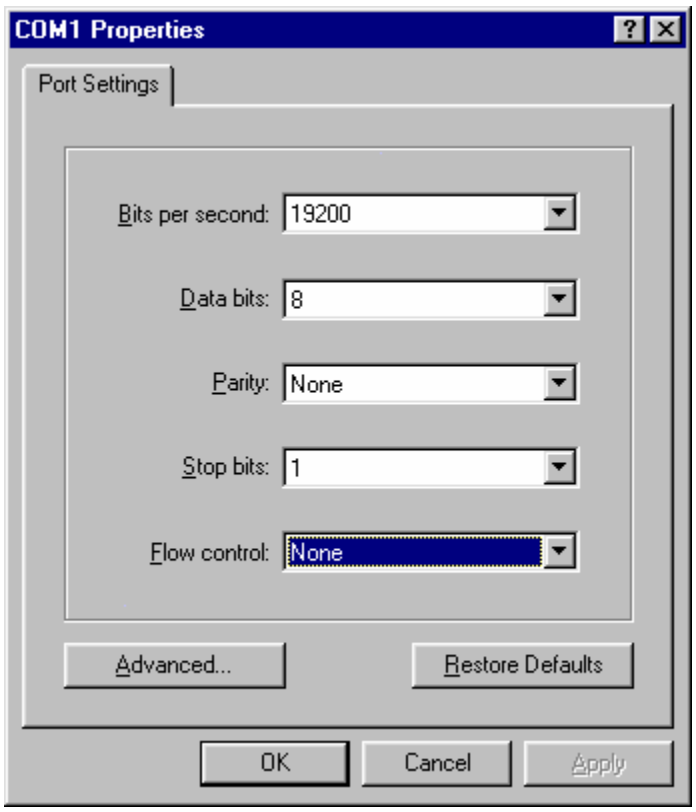
The communications protocol is a simple command/response protocol. The host PC (master) sends a command to the MC3B and will then receive a response from the MC3B. The MC3B will not initiate communications with the host PC.

The command syntax is of the form:

<controller id><command character>[<parameter 1>]
[,<parameter 2>]…[,<parameters n>]<CR>

All commands must start with a controller ID. Unless changed by the ‘I’ command (see section 6.1), the controller ID will be 0. All commands are terminated with a carriage return. Line feeds are ignored. Illegal characters will cause the command to be ignored and/or an error to be returned.

Sample Hyperterminal properties page



6.1) Initialize

<id>I

Initializes the controller and sets its controller ID to that specified in the command. The controller then acknowledges the command with the ID number it originally received. Thus, the host computer should send 0I to the motor controller and it will receive an acknowledgement of the form 0I. The motors are turned off during and after initialization.

Default values (applies to all motors):

Maximum velocity:	1000
Acceleration:	5000
Deceleration:	5000
Running current limit:	2000 mA
Holding current limit:	500 mA
Use holding current:	No
Half stepping:	Yes
Limit switches:	Disabled
Limit switch debounce count:	1

6.2) Set acceleration

<id>A<m>,<a>

Sets the acceleration <a> for motor <m>. The controller acknowledges with <id>A. The acceleration must be an integer in steps/second^2 and must be in the range of 100 to 100000.

6.3) Set deceleration

<id>D<m>,<d>

Sets the deceleration <d> for motor <m>. The controller acknowledges with <id>D. The deceleration must be an integer in steps/second^2 and must be in the range 100 to 100000.

6.4) Set maximum velocity

<id>V<m>,<v>[,<direction modifier>]

Sets the maximum velocity <v> for motor <m>. The host controller acknowledges with <id>V. The velocity is an integer in steps/seconds and must be in the range 1 to 5000.

Except for free-running motions (see section 6.7), if the command is received while the motor is in motion, an error string will be returned. In the case of free-running motions, the motor will accelerate or decelerate as needed until the newly specified velocity is reached.

The <direction modifier> parameter is optional. If it is not specified in the command it is assumed to be 0. The direction modifier is used to allow on-the-fly changes in direction for free-running motions. By specifying a value of 0 for the direction modifier, the motor will continue to turn in the direction originally specified. If the value of the direction modifier is 1, then the motor will decelerate to a stop, and then accelerate to the maximum velocity in the opposite directions. An example is shown below:

Command motor to perform a free-running motion in direction 1:

0R0,1

Command motor to change velocity. Because the direction modifier was not included in the command, the direction of the motor is unchanged:

0V0,1000

Command the motor to change velocity again. Because the direction modifier is set to 1, the motor will decelerate to a stop and then accelerate in direction 0:

0V0,1000,1

Command the motor to change velocity again. Because the direction modifier has not changed since the previous velocity command, the motor direction will not change.

0V0,1000,1

Command the motor to change velocity again. Because the direction modifier is set to 0, the motor will again decelerate to a stop and then accelerate in direction 1:

0V0,1000,0

6.5) Set limit switches

<id>L<m>,<h0a>,<h0x>,<h1a>,<h1x>

h0a: home AND mask for direction 0
h0x: home XOR mask for direction 0
h1a: home AND mask for direction 1
h1x: home XOR mask for direction 1

There are 6 home (limit) switches to share amongst the 3 motors <m> per controller <id>. A motor may utilize up to two home switches. The motor is considered to be at a home position for a specified direction (0 or 1) when:

((limit bits) AND (home_and_mask)) XOR (home_xor_mask)

is non-zero. No motion will be allowed to proceed beyond a home. By setting the home AND mask and the home XOR mask to zero for a given direction, the home switches will be ignored. This is the default power up state. The controller acknowledges the command with <id>L.

Example:

For motor controller 0, to set the motor 2 direction 0 limit to trigger on a high level in limit bit 3 and the direction 1 limit to trigger on a low level in bit 5, use the following command:

0L2,8,0,32,32

6.6) Set full/half stepping

<id>F<m>,<f>

Sets the full-step (<f> = 0) or half step (<f> = 1) setting for the specified motor <m>. The default power up state is for half steps. The controller acknowledges the command with <id>F.

6.7) Run motor

<id>R<m>,<d>

Runs the specified motor in the specified direction. The motor is accelerated to its maximum velocity and then maintains that velocity until either the home is reached, a stop command is received, or an initialization command is received. If a motion is already in progress, an error string is returned. The controller responds to the command with <id>R.

6.8) Move motor

<id>M<m>,<s>,<d>

Moves the specified motor, <m>, the specified number of steps, <s>, in the specified direction, <d>. The direction must be 0 or 1. The steps parameter is defined as a 32-bit unsigned long integer. If a motion is already in progress, an error string is returned. The controller responds to the command with <id>M.

6.9) Home motor

<id>H<m>,<d>

Homes the motor, <m>, by moving in the specified direction, <d>, until the limit switch (see section 6.5) is activated. The direction must be 0 or 1. The motion continues indefinitely until either the home is reached, a stop command is received, or an initialization command is received. When the motor reaches the home position, it will perform a low speed search to more precisely locate the home. If the motor is already in motion when this command is received, an error string is returned, otherwise the controller acknowledges this command with <id>H.

6.10) Rate-based motor motion

<id>G<m>,<s>,<d>,<rate>

This command causes the motor <m> to be stepped the specified number of steps, <s>, in direction <d> at the specified rate. No acceleration is applied. The <rate> parameter is an unsigned 32-bit integer and must be greater than or equal to 50. The <rate> parameter specifies the number of 10 microsecond intervals that must elapse between each step. For example:

0G1,100,0,75

would cause motor 1 to be moved 100 steps in direction 0 at a rate of 750 microseconds per step.

6.11) Stop motor

<id>S<m>

Immediately stops any motion for the specified motor. The controller acknowledges the command with a <id>S.

6.12) Synchronous move

<id>Y<s0>,<d0>,<s1>,<d1>,<s2>,<d2>[,<execute>]

This is the synchronous move command. The number of steps <s0>, <s1>, and <s2> for motors 0, 1, and 2 respectively, must be specified, even if 0. The directions <d0>, <d1>, and <d2> for motors 0, 1, and 2 respectively must also be specified. The motors are all moved synchronously so that a linear motion is achieved. The maximum velocity is computed in a manner that allows the motion to be completed as quickly as possible without exceeding the maximum velocity of any of the three motors.

If the option <execute> parameter is omitted, then the synchronous move specified is executed immediately and any moves stored in the chained-move buffer are discarded without being executed.

The optional <execute> parameter is used to create “chained” moves. By specifying a value of 0 for <execute>, the motion is stored in the chained-move buffer. When a value of 1 is specified for <execute>, all synchronous moves stored in the chained-move buffer are executed in the order they were stored, after which the chained-move buffer is cleared. In the example below, if used on a linear X-Y stage connected to motors 0 and 1, a square (45 degrees to the X-Y axes) synchronous motion would be executed:

0Y1000,0,1000,0,0,0,0
0Y1000,0,1000,1,0,0,0
0Y1000,1,1000,1,0,0,0
0Y1000,1,1000,0,0,0,1

Note: the motion will not start until the last command is received.

6.13) Query motor status

<id>Q<m>

Queries the state of motor <m>. The controller responds to this command with <id>Q<r>,<home0>,<home1>. The <r> parameter will be zero if the motor is not in motion and non-zero otherwise. The home0 status will be 0 if the motor is not at the home position for direction 0 and 1 if it is. The home1 status will be 0 if the motor is not at the home position for direction 1 and 1 if it is.

6.14) Query limit inputs

<id>B

Queries the state of the limit input byte. The debounced limit input byte is returned in the form <id>B<h> where h is the limit input byte value. Bit 0 corresponds to limit input 0, and so forth.

6.15) Set holding current usage

<id>C<m>,<c>

Controls the motor holding current for the specified motor <m>. If <c> is 0, no holding current is applied after each motion. If <c> is 1, the holding current is applied after each motion. A holding current allows the user to apply a series of consecutive motions without the possibility of unintended slippage due to relaxation between motions. The controller responds to this command with <id>C.

6.16) Set current limits

<id>K<m>,<h>,<r>

This command is used to set the holding current limit <h>, and the running current limit <r> for motor <m>. The current limits are expressed in milliamperes and must be in the range 0 to 2000. Setting the current limits too low may adversely affect the performance of the motors.

6.17) Set limit switch debounce count

<id>N<count>

The limit switches are sampled at a 1 KHz rate. In order for any change of state to be registered, the new state must be “debounced” by reading it a number of consecutive times without seeing a change. The <count> parameter must be in the range 1 to 100 and specifies the number of consecutive times that a limit switch bit must be at a new value before that new value is registered. If <count> is set to 1 (the default), debouncing is effectively disabled.

Note: setting the debounce count too high in relation to the motor velocity may allow the motor to take multiple steps beyond home position. It is the user’s responsibility to ensure that the system cannot be damaged by such an event.

6.18) Query product ID

<id>P

This command is used to obtain the product ID and the firmware version of the controller. The controller responds will a message in the following format:

<id>P<product>,<version>

For the MC3B motor controller, the <product> parameter will be 1. The firmware version will be an integer value.

6.19) Errors

Errors in sending serial commands to the MC-3B motor controller board will cause the controller to respond with an error string of the form <id>E<e>, where <e> is the error number. The error numbers are enumerated below:

0 Invalid command – command character not recognized

1 Motor running – command must be sent when motor is idle

2 Wrong number of parameters – the exact number of expected parameters must be supplied with a command.

3 Parameter out of range – range checking is applied to most parameters to ensure that they are valid.