Controlling the MPC-325 externally via computer is accomplished by sending commands to the MPC-200 controller over the USB interface between the computer and the USB connector on the rear of the ROE-200 that's connected to the MPC-200 controller. The USB device driver for Windows is downloadable from Sutter Instrument's web site (www.sutter.com). The MPC-325 (MPC-200) requires USB CDM (Combined Driver Model) Version 2.10.00 or higher. The CDM device driver for the MPC-325 (MPC-200) consists of two device drivers: 1) USB device driver, and 2) VCP (Virtual COM Port) device driver. Install the USB device driver first, followed by the VCP device driver. The VCP device driver provides a serial RS-232 I/O interface between a Windows application and the MPC-325 (MPC-200). Although the VCP device driver is optional, its installation is recommended even if it is not going to be used. Once installed, the VCP can be enabled or disabled.

The CDM device driver package provides two I/O methodologies over which communications with the controller over USB can be conducted: 1) USB Direct (D2XX mode), or 2) Serial RS-232 asynchronous via the VCP device driver (VCP mode). The first method requires that the VCP device driver not be installed, or if installed, that it be disabled. The second method requires that the VCP be installed and enabled.

Virtual COM Port (VCP) Serial Port Settings: The following table lists the required RS-232 serial settings for the COM port (COM3, COM5, etc.) generated by the installation or enabling of the VCP device driver.

Table 1. USB-VCP interface serial port settings.

Property	Setting
Data ("Baud") Rate (bits per second (bps))	128000
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	None

The settings shown in the above table can be set in the device driver's properties (via the Device Manager if in Windows) and/or programmatically in your application.

Protocol and Handshaking: Command sequences do not have terminators. All commands return an ASCII CR (Carriage Return; 13 decimal, 0D hexadecimal) to indicate that the task associated with the command has completed. When the controller completes the task associated with a command, it sends

ASCII CR back to the host computer indicating that it is ready to receive a new command. If a command returns data, the last byte returned is the task-completed indicator.

Command Sequence Formatting: Each command sequence consists of at least one byte, the first of which is the "command byte". Those commands that have parameters or arguments require a sequence of bytes that follow the command byte. No delimiters are used between command sequence arguments, and command sequence terminators are not used. Although most command bytes can be expressed as ASCII displayable/printable characters, the rest of a command sequence must generally be expressed as a sequence of unsigned byte values (0-255 decimal, 00 – FF hexadecimal, or 00000000 – 11111111 binary). Each byte in a command sequence transmitted to the controller must contain an unsigned binary value. Attempting to code command sequences as "strings" is not advisable. Any command data returned by the controller should be initially treated as a sequence of unsigned byte values upon reception. Groups of contiguous bytes can later be combined to form larger values, as appropriate (e.g., 2 bytes into 16-bit "word", or 4 bytes into a 32-bit "long" or "double word"). For the MPC-200, all axis position values (number of microsteps) are stored as "unsigned long" 32-bit positive-only values, and each is transmitted and received to and from the controller as four contiguous bytes.

Axis Position Command Parameters: All axis positional information is exchanged between the controller and the host computer in terms of microsteps. Conversion between microsteps and microns (micrometers) is the responsibility of the software running on the host computer (see *Microns/microsteps conversion* table for conversion factors).

Microsteps are stored as positive 32-bit values ("long" (or optionally, "signed long"), or "unsigned long" for C/C++; "I32" or "U32" for LabVIEW). "Unsigned" means the value is always positive; negative values are not allowed. The positive-only values can also be stored in signed type variables, in which case care must be taken to ensure that only positive values are exchanged with the controller.

The 32-bit value consists of four contiguous bytes, with a byte/bit-ordering format of Little Endian ("Intel") (most significant byte (MSB) in the first byte and least significant (LSB) in the last byte). If the platform on which your application is running is Little Endian, then no byte order reversal of axis posi-

2

tion values is necessary. Examples of platforms using Little Endian formatting include any system using an Intel/AMD processor (including Microsoft Windows and Apple Mac OS X).

If the platform on which your application is running is Big Endian (e.g., Motorola PowerPC CPU), then these 32-bit position values must have their bytes reverse-ordered <u>after</u> receiving from, or <u>before</u> sending to, the controller. Examples of Big-Endian platforms include many non-Intel-based systems, LabVIEW (regardless of operating system & CPU), and Java (programming language/environment). MATLAB and Python (script programming language) are examples of environments that adapt to the system on which each is running, so Little-Endian enforcement may be needed if running on a Big-Endian system. Some processors (e.g., ARM) can be configured for specific endianess.

Microsteps and Microns (Micrometers): All coordinates sent to and received from the controller are in microsteps. To convert between microsteps and microns (micrometers), use the following conversion factors (multipliers):

Table 2. Microns/microsteps conversion factors (multipliers).

Device	From/To Units	Conversion Factor (multiplier)
MP-225/M* micromanipulator	μsteps $\rightarrow$ μm	0.0625
	μm $\rightarrow$ μsteps	16

* Same applies to MP-285/M & MP-265/M micromanipulator, 3DMS & MPC-78 stages, and MOM & SOM microscope objective movers. Other devices:

- *MP-x45[S]/M & MP-865/M μmanipulators and MPC-x8 series stages: 1 μstep = 0.046875 μm; 1 μm = 21.333333333 μsteps.*
- *MT-800-based translators:*
    1 μstep = 0.078125 μm; 1 μm = 12.8 μsteps.

For accuracy in your application, type these conversion factors as "double" (avoid using the "float" type as it lacks precision with large values). When converting to microsteps, type the result as a 32-bit "unsigned long" (C/C++), "uint32" (MATLAB), or "U32" (LabVIEW) integer (positive only) value.

When converting to microns, type the result as a "double" (C/C++, MATLAB) or "DBL" (LabVIEW) 64-bit double-precision floating-point value.

Table 3. Travel distances and bounds.

Device	Axis	Len. (mm)	Microns (Micrometers (μm))	Microsteps (μsteps)
MP-225/M *	X	25mm	0 – 25,000	0 – 266,667
	Y	25mm	0 – 25,000	0 – 266,667
	Z	25mm	0 – 25,000	0 – 266,667

* Same applies to MP-285/M μmanipulators, 3DMS & MPC-78-series stages, & SOM objective mover. Other devices:

- *TRIO MP-x45/M series micromanipulator and MPC-x8 series stages: 25mm (533,333 microsteps) for X, Y, & Z. (Requires firmware v3.19+.)*
- *TRIO MP-865/M micromanipulator: 50mm for X, 12.5mm for Y, and 25mm for Z. (Requires firmware v3.21+.)*
- *MP-265/M micromanipulator: 25mm for X & Z, 12.5mm for Y. (Discontinued product – replaced by TRIO MP-865/M.)*
- *MT-8x0 (MT-22xx) series translator: 22mm in all three axes. Only X & Y are connected (Z can be optionally connected to another device (e.g., a focus drive)).*
- *MOM objective mover (firmware v3.13 or 3.16, and device Port A only): 21.5mm in all three axes.*

Travel Speed: The following table shows the travel speeds for single-, double-, and triple-axis movements for supported devices using orthogonal move commands.

Table 4. Travel speeds.

Device	mm/sec or μm/ms		
	Single Axis	Dual Axis (x 1.4)	Triple Axis (x 1.7)
MP-225/M *	3	4.2	5.1

* Applies also to the MP-245[S]/M, MP-845[S]/M, MP-865/M. & MP-265/M μmanipulators; and MPC-x8 series stages.

NOTE: The MP-285/M has a single-axis speed of 5 mm/sec., 7 for double axis, and 8.5 for triple-axis movement. The same applies also to the 3DMS & MPC-78 stages, MT-800-based translators, and SOM & MOM objective movers.

Command Reference: The following table lists all the external-control commands for the MPC-325 (MPC-200).

Table 5. MPC-200 controller external-control commands.

Command	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (Len.)	Value Dec.	Hex.	Binary	Alt-keypad #	Ctrl-char	ASCII def./-char.	Description
Get Connected-Devices Status ('A') (FW <3)	Tx	<3	1	0	65	41	0100 0001	0065		A	Returns the number of devices connected (0-4). *NOTE: This command is replaced by ('U') command in Firmware Ver. 3 and above (described next).*
	Rx	<3	0	Zero bytes = no devices connected. ROE displays "NO MANIPULATOR CONNECTED".							
	Rx	<3	2	0	1 – 4	01 – 04	0000 0000 – 0000 0100		^A – ^D	<SOH> <EOT>	Number of devices connected
				1	13	0D	0000 1101		^M	<CR>	Completion indicator

Command	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (Len.)	Value Dec.	Value Hex.	Value Binary	Alt-key-pad #	Ctrl-char	ASCII def./-char.	Description
Get Connected-Devices Status ('U') (FW 3+)	Tx	3+	1	0	85	55	1000 0101	0085		U	Returns the number of devices connected (0-4), and the connected status of Ports 1 - 2 (1st controller) and 3 - 4 (2nd controller daisy chained to the first). *NOTE: This command replaces the previous command ('A') in Firmware Ver. 3 and above.*
	Rx	3+	0								Zero bytes = no devices connected. ROE displays "NO MANIPULATOR CONNECTED".
	Rx	3+	6	0	1 – 4	01 – 04	0000 0000 – 0000 0100		^A – ^D	<SOH> – <EOT>	Number of devices connected
											Device # Connected Status
				1	0	0	0000 0000		^@	<NUL>	1: No
					1	1	0000 0001		^A	<SOH>	1: Yes
				2	0	0	0000 0000		^@	<NUL>	2: No
					1	1	0000 0001		^A	<SOH>	2: Yes
				3	0	0	0000 0000		^@	<NUL>	3: No
					1	1	0000 0001		^A	<SOH>	3: Yes
				4	0	0	0000 0000		^@	<NUL>	4: No
					1	1	0000 0001		^A	<SOH>	4: Yes
				5	13	0D	0000 1101		^M	<CR>	Completion indicator
Get Active Device & Firmware Version ('K')	Tx	All	1	0	75	4B	0100 1011	0075		K	Command
	Rx	<3	2	0	1 – 4	01 – 04	0000 0000 – 0000 0100		^A – ^D	<SOH> – <EOT>	Currently-active device (1 – 4).
				1	13	0D	0000 1101		^M	<CR>	Task-completion indicator
	Rx	3+	4	0	1 – 4	01 – 04	0000 0000 – 0000 0100		^A – ^D	<SOH> – <EOT>	Currently-active device (1 – 4).
				1							Minor version number coded in BCD (e.g., if ver. = 3.15, then byte = 0x15 (upper nibble = 1; lower nibble = 5))
				2							Major version number coded in BCD (e.g., if ver. = 3.15, then byte = 0x03 (upper nibble = 0; lower nibble = 3))
				3	13	0D	0000 1101		^M	<CR>	Completion indicator
Get Current Position ('C')	Tx	All	1	0	67	43	0100 0011	0067		C	Command
	Rx		14	0	1–4	01 – 04	0000 0000 – 0000 0100		^A – ^D	<SOH> – <EOT>	Drive number (1 – 4) to which the current position applies
											Three 4-byte (32-bit) values (current positions in μsteps of X, Y, & Z) + 1 byte for completion indicator. See *Ranges* table for minimum and maximum values.
				1–4 (4)							X pos. in μsteps
				5–8 (4)							Y pos. in μsteps
				9–12 (4)							Z pos. in μsteps
				13	13	0D	0000 1101		^M	<CR>	Completion indicator

4

Command	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (Len.)	Value Dec.	Value Hex.	Value Binary	Alt-key-pad #	Ctrl-char	ASCII def./-char.	Description
Change Active Device ('I')	Tx	All	2	0	73	49	0100 1001	0073		I	Command
				1	1–4	01 – 04	0000 0001 – 0000 0100	0001 – 0004	^A – ^D	<SOH> – <EOT>	Device number (by value) to change (1 through 4)
	Rx	1 - 1.05	1	0	13	0D	0000 1101		^M	<CR>	Task-completion indicator
	Rx	1.06 +	2	0	1–4 or 69	01 – 04 or 45	0000 0001 – 0000 0100 or 0100 0101		^A – ^D	<1> – <4> or 'E'	If device specified exists, then device number (1-4) is returned. Otherwise, 'E' (error) is returned.
				1	13	0D	0000 1101		^M	<CR>	Completion indicator
Move to Home Position[1] ('H')	Tx	All	1	0	72	48	0100 1000	0072		H	Command
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Move to Work Position [2] ('Y')	Tx	All	1	0	89	59	0101 1001	0089		Y	Command
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Move to Center Position ('N') (FW <=1.03)	Tx	<= 1.03	1	0	78	4E	0100 1110	0078		N	Command: Moves active device to the center of travel position.
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Calibrate ('N') (FW >1.03)	Tx	>1.03	1	0	78	4E	0100 1110	0078		N	Command: Calibrates the active device.
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Move to Specified Position Orthogonally at Full Speed ('M')	Tx	All	13	0	77	4D	0100 1101	0077		M	Moves X, Y, and Z to specified position (stereotypic at fastest speed)
					X, Y, & Z target absolute positions, in microsteps, each consisting of 4 contiguous bytes representing a single 32-bit positive integer value (see *Ranges* table).						
				1–4 (4)							X µsteps
				5–8 (4)							Y µsteps
				9–12 (4)							Z µsteps
	=> 30ms										Time of travel (see Notes)
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Completion indicator

---

[1] The "Home Position" is defined manually on the ROE-200.

[2] The "Work Position" is defined manually on the ROE-200.

Command	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (Len.)	Dec.	Hex.	Binary	Alt-key-pad #	Ctrl-char	ASCII def./-char.	Description
Move to Specified Position in a Straight Line at Specified Speed ('S')	Tx	3+	14	0	83	53	0101 0011	0083		S	Command
				1	0 – 15	00 – 0F	0000 0000 – 0000 FFFF	0000 – 0015	^@ – ^O	<NUL> – <SI>	Velocity (0 = slowest, 15 = fastest) (See Notes)
	30ms										⚠ Required delay between Velocity byte and remaining bytes
	Tx										X, Y, & Z target absolute positions, in microsteps, each consisting of 4 contiguous bytes representing a single 32-bit positive integer value (see *Ranges* table).
				2–5 (4)							X μsteps
				6–9 (4)							Y μsteps
				10–13 (4)							Z μsteps
	=> 30ms										Time of travel (see Notes)
	Rx		1	0	13	0D	0000 1101		^M	<CR>	Completion indicator (streaming positional data is OFF – see 'F' command) once movement completes, or streaming current position data (see 'O' command) and then completion indicator once movement completes. (See *'S' Command's Streaming Return Data for Current Position* note.)
Interrupt Move (^C)	Tx	All	1	0	3	03	0000 0111	0003	^C	<ETX>	Interrupts a move in progress that was previously initiated by any move command.
	Rx		1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Turn OFF S-Move command streaming data ('F')	Tx	3+	1	0	70	46	0100 0110			F	Command (see *'S' Command's Streaming Return Data for Current Position* note)
	Rx		1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Turn ON S-Move command streaming data ('O')	Tx	3+	1	0	79	4F	0100 1111			O	Command (see *'S' Command's Streaming Return Data for Current Position* note)
	Rx		1	0	13	0D	0000 1101		^M	<CR>	Completion indicator
Set ROE MODE ('L')	Tx	All	2	0	76	4C	0100 1100	0076		L	Command
				1	0–9	0–9	0000 0000 – 0000 1001	0000 – 0009			Mode 0 - 9 (coarsest/fastest to finest/slowest)
	Rx	All	1	0	13	0D	0000 1101		^M	<CR>	Task-completion indicator

NOTES:

1. **Task-Complete Indicator:** All commands will send back to the computer the "Task-Complete Indicator" to signal the command and its associated function in controller is complete. The indicator consists of one (1) byte containing a value of 13 decimal (0D hexadecimal), and which represents an ASCII CR (Carriage Return).

2. **Intercommand Delay:** A short delay (usually around 2 ms) is recommended between commands (after sending a command sequence and before sending the next command).

3. **Clearing the I/O Send & Receive Buffers:** Clearing (purging) the transmit and receive buffers of the I/O port immediately before sending any command is recommended. Note that this clearing of the buffers affects only the computer-side I/O; it does not (necessarily) clear the buffers on the controller side, requiring, when necessary, to reset/power-cycle the controller. Following the rules described will generally avoid problems with getting garbage data in the I/O buffers of both the computer and controller (i.e., using exact number of bytes for both command sequences and return data (as per the *Commands* table), never sending a command before the previous command is finished with its task, etc.).

4. **Positions in Microsteps:** All positions sent to and received from the controller are in microsteps (μsteps). See *Microns/microsteps conversion* table) for conversion between μsteps and microns (micrometers (μm)).

Declaring position variables in C/C++:
```
/* current position for X, Y, & Z */
unsigned long  cp_x_us, cp_y_us, cp_z_us;  /* mi-
crosteps */
double        cp_x_um, cp_y_um, cp_z_um;  /* mi-
crons */
/* specified (move-to) position for X, Y, & Z */
unsigned long  sp_x_us, sp_y_us, sp_z_us;  /* mi-
crosteps */
double        sp_x_um, sp_y_um, sp_z_um;  /* mi-
crons */
```
Use the same convention for other position variables the application might need.

Declaring the microsteps/microns conversion factors in C/C++:
```
/* conversion factors for MP-225/M, MP-285/M, or
MP-265/M based config. */
double us2umCF = 0.0625; /* microsteps to microns
*/
double um2usCF = 16;     /* microns to microsteps
*/
/* conversion factors for MP-245[S]/M, MP-
845[S]/M, or MP-865/M based config. */
double us2umCF = 0.046875;    /* microsteps to
microns */
double um2usCF = 21.333333333; /* microns to mi-
crosteps */
/* conversion factors for MT-800 config. */
double us2umCF = 0.078125; /* microsteps to mi-
crons */
double um2usCF = 12.8; /* microns to microsteps */
```
Converting between microsteps and microns in C/C++:
```
/* converting X axis current position */
cp_x_um = cp_x_us * us2umCF; /* microsteps to mi-
crons */
cp_x_us = cp_x_um * um2usCF; /* microns to mi-
crosteps */
```
Do the same for Y and Z, and for any other position sets used in the application.

5. **Ranges and Bounds:** See Ranges and Bounds table for exact minimum and maximum values for each axis of each compatible device that can be connected. All move commands must include positive values only for positions – negative positions must never be specified. All positions are absolute as measured from the physical beginning of travel of a device's axis. In application programming, it is important that positional values be checked (>= 0 and <= max.) to ensure that a negative absolute position is never sent to the controller and that end of travel is not exceeded. All computational relative positioning must always resolve to accurate absolute positions.

Declaring minimum and maximum absolute position variables in C/C++:
```
/* minimum and maximum positions for X, Y, & Z */
double min_x_um, min_y_um, min_z_um;  /* minimum
microns */
double max_x_um, max_y_um, max_z_um;  /* maximum
microns */
```
Set minimum and maximum absolute positions for each axis – see Ranges & Bounds table.
```
/* initialize all minimum positions in microns*/
min_x_um = 0;
min_y_um = 0;
min_z_um = 0;
/* initialize all maximum positions in microns*/
/* MP-225/M, MP-285/M, MP-845[S]/M, MP-245[S]/M,
etc. */
max_x_um = 25000;
max_y_um = 25000;
max_z_um = 25000;
/* MP-865/M */
max_x_um = 50000;
max_y_um = 12500;
max_z_um = 25000;
```
```
/* MP-265/M */
max_x_um = 25000;
max_y_um = 12500;
max_z_um = 25000;
```

6. **Absolute Positioning System Origin:** The Origin is set to a physical position of travel to define absolute position 0. The physical Origin position is fixed at beginning of travel (BOT). This means that all higher positions (towards <u>end</u> of travel (EOT)) are positive values; there are no lower positions and therefore no negative values are allowed.

7. **Absolute vs. Relative Positioning:** Current position ('c') and move commands always use absolute positions. All positions can be considered "relative" to the Origin (Position 0), but all are in fact absolute positions. Any position that is considered to be "relative" to the current position, whatever that might be, can be handled synthetically by external programming. However, care should be taken to ensure that all relative position calculations always result in correct positive absolute positions before initiating a move command.

Declaring relative position variables in C/C++:
```
/* relative positions for X, Y, & Z */
double rp_x_um, rp_y_um, rp_z_um;  /* microns */
/* initialize all relative positions to 0 after
declaring them */
rp_x_um = rp_y_um = rp_z_um = 0;
```

Enter any positive or negative value for each relative position (e.g., rp_x_um = 1000; rp_y_um = 500; rp_z_um = -200 ... etc.

For each axis, check to make sure that the new resultant absolute position (to which to move) is within bounds. Reset the relative position to 0 if not. If relative value is negative, its positivized value must not be greater than the current position. Otherwise, if positive, adding current position with relative position must not exceed the maximum position allowed. If out of bounds, resetting relative position to 0 allow the remaining conversions and movement to resolve without error.
```
/* check to make sure that relative X is within
bounds */
if ( ( rp_x_um < 0 && abs(rp_x_um) > cp_x_um ) ||
    (cp_x_um + rp_x_um > max_x_um) )  /* out of
bounds? */
    rp_x_um = 0; /* yes, so reset relative pos. to
0 */
```
Repeat the above bounds check for each of the remaining axes.

For each axis, calculate new absolute position in microns and then convert to microsteps before issuing a move command.
```
/* convert X relative position to absolute posi-
tion */
sp_x_um = cp_x_um + rp_x_um; /* add relative pos.
to current pos. */
/* convert new absolute X position in microns to
microsteps */
sp_x_us = sp_x_um * um2usCF;
```
Repeat for each of the remaining axes as required before issuing a move command.

8. **Position Value Typing:** All positions sent and received to and from the controller are in microsteps and consist of 32-bit integer values (four contiguous bytes). Position values in microsteps are always positive, so data type must be an "unsigned" integer that can hold 32 bits of data. Although each positional value is transmitted to, or received from, the controller as a sequence of four (4) contiguous bytes, for computer application computational and storage purposes each should be typed as an unsigned 32-bit integer ("unsigned long" in C/C++; "uint32" in MATLAB, "U32" in LabVIEW, etc.).

Position values in microns (micrometers or μm) should be data typed as double-precision floating point variables ("double" in C/C++ and MATLAB, "DBL" in LabVIEW, etc.).

Note that in Python, incorporating the optional NumPy package brings robust data typing like that used in C/C++ to your program, simplifying coding and adding positioning accuracy to the application.

9. **Position Value Bit Ordering:** All 32-bit position values transmitted to, and received from, the controller must be bit/byte-ordered in "Little Endian" format. This means that the least significant bit/byte is last (last to send and last to receive). Byte-order reversal may be required on some platforms. Microsoft Windows, Intel-based Apple Macintosh systems running Mac OS X, and most Intel/AMD processor-based Linux distributions handle byte storage in Little-Endian byte order so byte reordering is not necessary before converting to/from 32-bit "long" values. LabVIEW always handles "byte strings" in "Big Endian" byte order irrespective of operating system and CPU, requiring that the four bytes containing a microsteps value be reverse ordered before/after conversion to/from a multibyte type value (I32, U32, etc.). MATLAB automatically adjusts the endianess of multibyte storage entities to that of the system on which it is running, so explicit byte reordering is generally unnecessary unless the underlying platform is Big Endian. If your development platform does not have built-in Little/Big Endian conversion functions, bit reordering can be accomplished by first swapping positions of the two bytes in each 16-bit half of the 32-bit value, and then swap positions of the two halves. This method efficiently and quickly changes the bit ordering of any multibyte value between the two Endian formats (if Big Endian, it becomes Little Endian, and if Little Endian, it becomes then Big Endian).

10. **Travel Lengths and Durations:** "Move" commands might have short to long distances of travel. If not polling for return data, an appropriate delay should be inserted between the sending of the command sequence and reception of return data so that the next command is sent only after the move is complete. This delay can be auto calculated by determining the distance of travel (difference between current and target positions) and rate of travel. This delay is not needed if polling for return data. In either case, however, an appropriate timeout must be set for the reception of data so that the I/O does not time out before the move is made and/or the delay expires.

11. **Orthogonal Move Speed:** Full speed for the "Orthogonal Move 'M'" command is 5000 microns/sec. (5 mm/sec. or microns/millisecond) for single-axis movements (3000 μm/sec. (3 mm/sec. or μm/ms) for MP-225/M).

12. **Straight-Line Move Speeds:** Actual speed for the "Straight-Line Move 'S'" command can be determined with the following formula: (1300 / 16) * (sp +1), where 1300 is the maximum speed in microns/second and "sp" is the speed level 0 (slowest) through 15 (fastest). For mm/second or microns/millisecond, multiply result by 0.001.

Table 6.Straight-line move 'S' command speeds.

Speed Setting	mm/sec or μm/ms	μm/sec or nm/ms	nm/sec	in/sec or mil/ms	% of Max.
15	1.30000	1300.00	1300000	0.051181102	100.00%
14	1.21875	1218.75	1218750	0.047982283	93.75%
13	1.13750	1137.50	1137500	0.044783465	87.50%
12	1.05625	1056.25	1056250	0.041584646	81.25%
11	0.97500	975.00	975000	0.038385827	75.00%
10	0.89375	893.75	893750	0.035187008	68.75%
9	0.81250	812.50	812500	0.031988189	62.50%
8	0.73125	731.25	731250	0.028789370	56.25%
7	0.65000	650.00	650000	0.025590551	50.00%
6	0.56875	568.75	568750	0.022391732	43.75%
5	0.48750	487.50	487500	0.019192913	37.50%
4	0.40625	406.25	406250	0.015994094	31.25%
3	0.32500	325.00	325000	0.012795276	25.00%

Speed Setting	mm/sec or μm/ms	μm/sec or nm/ms	nm/sec	in/sec or mil/ms	% of Max.
2	0.24375	243.75	243750	0.009596457	18.75%
1	0.16250	162.50	162500	0.006397638	12.50%
0	0.08125	81.25	81250	0.003198819	6.25%

13. **Multi Axis Movement Speed Increase:** Specified travel speeds are for single-axis movements. When travel traverses a 45° diagonal within a dual-axis square, speed is increased by 40% (x 1.4), and by 70% (x 1.7) within a triple-axis cube.

14. **Move Interruption:** A command should be sent to the controller only after the task of any previous command is complete (i.e., the task-completion terminator (CR) is returned). One exception is the "Interrupt Move" (^C) command, which can be issued while a command-initiated move is still in progress.

15. **Extracting the MPC-200 Firmware Version Number:** The firmware version number returned by the 'K' command is encoded in BCD (Binary Coded Decimal) in two bytes, with minor version byte first, followed by major version byte, each of which contains two-digit pairs , the first of which is in the upper nibble and the next in the lower nibble. For example, if the complete version is 3.15, then the bytes at offsets 1 and 2 will show (in hexadecimal) as 0x15 0x03 (ret[1] and ret[2] as shown in the following code snippets). The following code shows how to extract and convert the 4 BCD digits into usable forms for later comparison without altering the original command return data (written in C/C++ and is easily portable to Python, Java, C#, MATLAB script, etc.).

```
/* "ret" is the array of bytes containing
the 'K' command's return data */
/* define variables */
unsigned char verbyte;  /* temp work byte */
int minver, majver, majminver;
float version;
```

Minor version number as an integer (e.g., 15):
```
verbyte = ret[1]; /* get minor ver. digits */
/* get 1's digit & then get & add 10's digit */
minver = (verbyte & 0x0F) +
    ((verbyte >>4 & 0x0F) * 10);
```

Major version number as an integer (e.g., 3):
```
verbyte = ret[2];  /* get major ver. digits */
majver = (verbyte & 0x0F) +
    ((verbyte >>4 & 0x0F) * 10);
```

Complete (thousands) version as an integer (e.g., 315):
```
majminver = majver * 100 + minver;
```

Complete version as a floating-point number (e.g., 3.15):
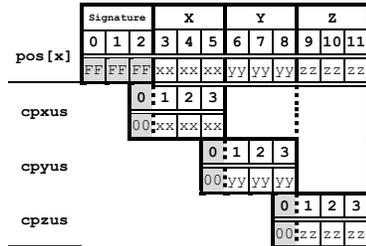```
version = majminver * .01;
```

16. **'S' Command's Streaming Return Data for Current Position:** The Straight-Line Move ('S') command has two modes of operation:

a. a CR is returned when the target position has been reached ('F' (Off) command before the 'S' command sequence), or

b. streaming positional data is returned while movement is occurring, and then a CR once movement is complete ('O' (On) command before the 'S' command sequence).

Positional data is streamed at every 1 micron of movement, and the rate (data per second) depends on the 'S' command speed level used. Each positional data block streamed consists of 12 bytes:
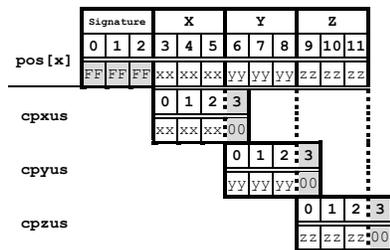
1. The 1st three bytes each contains FF hexadecimal (255 decimal) as a data block signature,

2. the next 3 contains positional data for the X axis,

3. the penultimate is for Y, and

4. last for Z.

All positional data are in microsteps. Each 3-byte position needs to be converted into 4-byte blocks by prepending a byte containing 0, so that the resulting data (now 4 bytes) can be treated programmatically as an unsigned 32-bit "long" (C/C++) or "U32" (LabVIEW) data type. All positional data streamed is in Little-Endian bit/byte order (Wintel), so conversion to 32-bit longs will require bit-order reversal (byte swapping) for Big-Endian platforms (e.g., LabVIEW). The appropriate microstep-to-microns conversion factor is needed according to the device type being moved (see *Microns/microsteps conversion factors (multipliers)* table).

Little-Endian bit/byte order environment:



Big-Endian bit/byte order environment ("pos" is in Little-Endian format):



The following C/C++ code snippets can be used to process the streaming data.

*Array for a streamed 12-byte block of data containing current position*

```
unsigned char pos[12];
```

32-bit variables for current position in microsteps, all initialized to 0 to ensure MSB (Most Significant Byte) allows only positive values

```
long cpxus, cpyus, cpzus;
cpxus = cpyus = cpzus = 0;
```

Copy 24-bit (3-byte) position for each axis to 32-bit (4-byte) equivalents. Use the byte position offsets shown in the diagram above. ("le" means Little Endian; "be" means Big Endian bit/byte order.)

If in Little-Endian environment (e.g., Windows, Intel-MacOSX), copy all 3 U24 bytes for each axis to the respective U32 variables.

```
memcpy(&cpxus[1], &pos[3], 3); // X
memcpy(&cpyus[1], &pos[6], 3); // Y
memcpy(&cpzus[1], &pos[9], 3); // Z
```

If in Big-Endian environment (e.g., legacy MacOS, LabVIEW), copy U24 to U32 byte at a time (1st to 3rd, 2nd to 2nd, & 3rd to 1st). Note that "pos" is always in Little-Endian bit/byte order.

```
memcpy(&cpxus[2], &pos[3], 1); /* X */
memcpy(&cpxus[1], &pos[4], 1);
memcpy(&cpxus[0], &pos[5], 1);
memcpy(&cpyus[2], &pos[6], 1); /* Y */
memcpy(&cpyus[1], &pos[7], 1);
memcpy(&cpyus[0], &pos[8], 1);
memcpy(&cpzus[2], &pos[9], 1); /* Z */
memcpy(&cpzus[1], &pos[10], 1);
memcpy(&cpzus[0], &pos[11], 1);
```

Ready to update UI with current position in microsteps using 32-bit integer values. Double-precision variables for current position in microns; initialize each to 0.

```
double cpxum, cpyum, cpzum;
cpxum = cpyum = cpzum = 0;
```

Microsteps-to-microns conversion factor (see "Microns / microsteps conversion" table for appropriate factor)

```
double us2umCF = 0.0625;
```

Get microns from microsteps for each axis

```
cpxum = cpxus * us2umCF;
cpyum = cpyus * us2umCF;
cpzum = cpzus * us2umCF;
```

Ready to update UI with current position in microns using double-precision values. Loop for next data block as desired until streaming ends.

For LabVIEW, a 3-byte positional value for an axis can be transferred into a byte array, and then into a U32 data type via a byte-swap function to ensure 24-bit to 32-bit conversion while making sure that no high-order value is misinterpreted as a sign bit (there should never be a negative positional value in the MPC-200). LabVIEW data types (e.g., U16, U32, I32) are always in Big-Endian bit/byte order, while MPC-200 multibyte values are always transcieved in Little-Endian bit/byte order.

A single completion indicator byte (ASCII CR) is returned when streaming ends and target position has been reached.

NOTES: