SUTTER INSTRUMENT®

MP-285

Micromanipulator, Stage, Translator, or Microscope Objective Control System (MP-285, MP-x8 & MT-20xx Series, MOM, & SOM)

(Includes MP-285 and MP-285A Model Controllers)

External Control Quick Reference

Rev. 2.80 (20201005)

NOTE: Unless otherwise specified, all "MP-285" references refer equally to Model MP-285 and Model MP-285A controllers.

Controlling the MP-285 or MP-285A externally via computer is accomplished by sending commands between the computer and the equivalent connector on the rear of the controller: **SERIAL** (RS-232, 9-pin DSUB (MP-285 or MP-285A) or **USB** (MP-285A only).

The SERIAL (RS-232) Interface: RS-232-C, minimal 3-wire (Ground, Transmit, & Receive), 9-pin D-Shell connector (labeled "SERIAL" on the rear panel of the controller.

Table 1. Serial RS-232 (DB9 connector) port settings.

Property	Setting
Data ("Baud") Rate (bps (bits per second))	19200, **9600***, 4800, 2400, 1200
Data Bits	**8**
Stop Bits	**1***, 1.5, 2
Parity	**None***, Even, Odd
Flow Control	**None**

**** Default***

NOTE: The data rate can be selected via the MP-285 controller's display and keypad. The default data rate (9600 bps) is recommended for most applications. The Parity can also be configured to be "ON", although the OFF ("None") setting is recommended and is the default.

The MP-285A USB Interface: Controlling the MP-285A externally via computer is accomplished by sending commands over the USB interface between the computer and the USB connector on the rear panel of the MP-285A controller/ROE. The USB device driver for Windows is downloadable from Sutter Instrument's web site (www.sutter.com). The MP-285A requires USB CDM (Combined Driver Model) Version 2.10.00 or higher. The CDM device driver for the MP-285A 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 MP-285A. 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 and enabling of the VCP device driver.

Table 2. MP-285A USB-VCP interface serial port settings.

Property	Setting
Data ("Baud") Rate (bps (bits per second))	9600
Data Bits	8
Stop Bits	1
Parity	None
Flow Control [1]	"Hardware" or RTS/CTS

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: Most command sequences have a terminator: ASCII CR (Carriage Return; 13 decimal, 0D hexadecimal) (see the *MP-285 external-control commands* table). 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 se-

[1] While the Flow Control property for the RS-232 DB9 interface is always set to "None", it must be set to "Hardware" or RTS/CTS signaling for the virtual serial port via the USB-VCP device driver.

COPYRIGHT © 2020 SUTTER INSTRUMENT COMPANY. ALL RIGHTS RESERVED.

quence of bytes that follow the command byte. No delimiters are used between command sequence arguments, and command sequence terminators are used in most cases. 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 being transmitted to the controller must contain an unsigned binary value. Attempting to code command sequences as "strings" is not advisable. Any command data being returned from the controller must also be received and initially treated as a sequence of unsigned byte values. 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 MP-285 controller, all axis position values (number of microsteps) are stored as "long" (or "signed long") 32-bit positive or negative 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 or negative 32-bit values ("long" (or optionally, "signed long") for C/C++; "I32" for LabVIEW).

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 position 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 after receiving from, or before 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 (µsteps). To convert between microsteps and microns (micrometers (µm)), use the following conversion factors (multipliers):

Table 3. Microns/microsteps conversion.

Device	From/To Units	Conv. Factor
MP-285/M* micromanipulator	microsteps → microns	0.04
	microns → microsteps	25
MT-800 (**MT-20xx**) series translators	microsteps → microns	0.05
	microns → microsteps	20

** Applies also to 3DMS/M & MP-x8-series stages, and MOM & SOM microscope objective movers*

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 "long", "signed long", or "I32" integer. When converting to microns, type the result as "double" (64-bit double-precision floating-point values).

Ranges and Bounds:

Table 4. Ranges and bounds.

Device	Axis	Len. (mm)	Origin	Microns (Micrometers (µm))	Microsteps (µsteps)
MP-285/M, 3DMS, MP-78, MOM, SOM	X, Y, & Z	25 mm	**COT***	**-12,500 – 12,500**	**-200,000 – 200,000**
			BOT	0 – 25,000	0 – 400,000
MT-800	X & Y	22 mm	**COT***	**-11,000 – 11,000**	**-140,800 – 140,800**
			BOT	0 – 22,000	0 – 281,600
	Z	25 mm	**COT***	**-12,500 – 12,500**	**-200,000 – 200,000**
			BOT	0 – 25,000	0 – 400,000

** Factory default.*

NOTE: Origin is a physical position of travel that defines the center of the absolute position coordinate system (i.e., absolute position 0).
Physical Positions: BOT (Beginning Of Travel), COT (Center Of Travel), & EOT (End Of Travel).
In the MP-285, the Origin can be set to any physical position (factory default is COT).

NOTE: The MP-x8-series stage and MT-800 (MT-20xx series) translator do not have a Z-axis motor. In either case, the controller's Z axis can be optionally connected to a motor of a different device (e.g., focus drive).

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

Table 5. Travel speeds.

Resolution	Speed Range (microns/sec)
Low (coarse: 0.2 µm/µstep (10 µsteps/step))	0 – 3000*
High (fine: 0.04 µm/µstep (50 µsteps/step))	0 – 1310

** CAUTION: Although the absolute maximum microns/sec. speed allowable in low (coarse) resolution is 6,550, it is essential that a speed no higher than 3,000 be used with the MP-285A model controller.*

Command Reference: The following table lists all the external-control commands for the MP-285.

Table 6. MP-285[A] controller external-control commands.

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 Current Position ('c')	**Tx**	All	2	0	99	63	0110 0011	0099		'c'	Returns the current positions of X, Y, & Z axes in µsteps.
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		13		Three 4-byte (32-bit) values (current positions in µsteps of X, Y, & Z), + 1 byte for completion indicator.						
				0 (4)							Current X–axis position in µsteps (32-bit signed integer)
				4 (4)							Current Y–axis position in µsteps (32-bit signed integer)
				8 (4)							Current Z–axis position in µsteps (32-bit signed integer)
				12	13	0D	0000 1101			<CR>	Task-completion indicator
Move to Specified Position ('m')	**Tx**	All	14	0	109	6D	0110 1100	0109		'm'	Moves to specified position (µsteps) (see *Ranges* table)
				1 (4)							Target position for X in µsteps (32-bit signed integer)
				5 (4)							Target position for Y in µsteps (32-bit signed integer)
				9 (4)							Target position for Z in µsteps (32-bit signed integer)
				13	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx			1	13	0D	0000 1101	0013		<CR>	Task-completion indicator
Set Velocity & Resolution ('V')	**Tx**	All	4	0	86	56	0101 0110	0086		'V'	Command (Note: Uppercase 'V') (see *Resolution & Velocity* note)
				1 (2)	0 – 4095 or 32, 768 - 35, 768	0000 - 051E or 8000 - 8BB8	00000000 00000000 - 00000101 00011110 or 10000000 00000000 - 10001011 10111000	One **unsigned** short (16-bit) integer (2 bytes) containing both resolution and velocity values. MSB (Bit 15) contains resolution setting; remaining bits (14 –0) contains velocity value. **Resolution (Bit 15)**: 0 = Low (coarse: 0.2 µm/µstep (10 µsteps/step)) 1 = High (fine: 0.04 µm/µstep (50 µsteps/step)) **Velocity (Bits 14-0)**: Low Res.: 0 – 6550 (MP-285) or 3000 (MP-285A) µm/sec High Res.: 0 – 1310 µm/sec			
				3	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx			1	13	0D	0000 1101	0013		<CR>	Task-completion indicator
Set Origin ('o')	**Tx**	All	2	0	111	6F	0110 1111	0111		'o'	Sets the Absolute Origin to the current position.
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx			1	13	0D	0000 1101	0013		<CR>	Task-completion indicator

Command	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (len.)	Value			Alt-key-pad	Ctrl-char	ASCII def./-char.	Description
					Dec.	Hex.	Binary				
Set Absolute Mode (‘a’)	**Tx**	All	2	0	97	61	0110 0001	0097		**‘a’**	Sets movement mode to Absolute. Each ‘m’-command axis value represents an absolute position. (*Note: No controller display update.*)
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx			1	13	0D	0000 1101	0013		<CR>	Task-completion indicator
Set Relative Mode (‘b’)	**Tx**	All	2	0	98	62	0110 0010	0098		**‘b’**	Sets movement mode to Relative. Each ‘m’-command axis value represents a position relative to the current position. (*Note: No controller display update.*)
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx			1	13	0D	0000 1101	0013		<CR>	Task-completion indicator
Interrupt Move (^C)	**Tx**	All	1	0	3	03	0000 0011	0003	^C	**<ETX>**	Interrupts an ‘m’-command initiated move in progress
	Rx		1	0	61	3D	0011 1011	0061		‘=’	Move in progress indicator
			2	0	13	0D	0000 1101	0013		<CR>	Task-completion indicator
	Rx			0	13	0D	0000 1101	0013		<CR>	Task-completion indicator (movement was not in progress)
Refresh VFD Display (‘n’)	**Tx**	All	2	0	110	6E	0110 0110	0101		**‘n’**	Refreshes the controller’s display (X, Y, & Z coordinates only)
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1	0	13	0D	0000 1101	0013	^M	<CR>	Task-completion indicator
Reset Controller (‘r’)	**Tx**	All	2	0	114	72	0111 0010	0114		**‘r’**	Resets the controller.
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1	0	13	0D	0000 1101	0013		<CR>	Task-completion indicator
Get Status (‘s’)	**Tx**	All	2	0	115	73	0111 0011	0115		**‘s’**	Returns status information
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		33	0 (32)							Status data – see *Status Data Structure* table.
				32	13	0D	0000 1101	0013		<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 and Microns:** 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 */
long   cp_x_us, cp_y_us, cp_z_us;  /* microsteps */
double cp_x_um, cp_y_um, cp_z_um;  /* microns */
/* specified (move-to) position for X, Y, & Z */
long   sp_x_us, sp_y_us, sp_z_us;  /* microsteps */
double sp_x_um, sp_y_um, sp_z_um;  /* microns */
```

   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-285/M based config. */
double us2umCF = 0.04; /* microsteps to microns */
double um2usCF = 25;   /* microns to microsteps */
/* conversion factors for MT-800 config. */
double us2umCF = 0.05; /* microsteps to microns */
double um2usCF = 20;   /* microns to microsteps */
```

   NOTE: In an MP-285A-based system configured for an MP-78 stage or MT-800-based XY translator (MT-2078), the Z axis may be configured for different conversion factors (e.g., if Z is wired to a separate device such as a focus drive). In such cases, make sure the appropriate microsteps/microns conversion factors are used for Z while using the standard factors for X and Y.

Converting between microsteps and microns in C/C++:

```
/* converting X axis current position */
cp_x_um = cp_x_us * us2umCF;  /* microsteps to microns */
cp_x_us = cp_x_um * um2usCF;  /* microns to microsteps */
```

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 include positive or negative values for positions. All positions are absolute as measured from the Origin position as set in the controller for all axes of the attached device. The factory default Origin position is the physical center position (between beginning of travel and end of travel) of the device. In application programming, it is important that positional values be checked (>= minimum and <= maximum) to ensure that a position is within the bounds of travel before it is sent to the controller.

6. **Absolute Positioning System Origin:** The Origin is set to a physical position of travel to define absolute position 0. The factory default physical Origin position is center of travel (COT). This means that all higher positions (towards end of travel (EOT)) are positive values, and all lower positions (towards beginning of travel (BOT)) are negative values. The Origin can be changed (via the controller's front panel display/keypad or via the Origin ('o') command sent from an external program.

   CAUTION: When changing the Origin from its factory default, it is not possible to obtain the new Origin's physical position via an external control command. If changing the Origin's physical position via the external control 'o' command, it is recommended that the external application keep careful track of all Origin changes, and automatically adjust its view of the absolute position coordinate system according to the current Origin's physical position.

7. **Absolute vs. Relative Positioning:** Current position (via the 'c' command) report absolute positions of each axis. Moving to a new position (via the 'm' command) is specified with absolute position values when in Absolute mode ('a' command) or with relative values (relative to the current position) when in Relative mode ('b' command).

   CAUTION: In an external control program, care should be taken to ensure that the Absolute/Relative mode state be updated upon a mode change and kept track of, as it not possible to obtain the current mode from the controller. In addition, any computational relative positioning made in an external program while in Absolute mode must ensure that relative positions are accurately translated to correct absolute positions before initiating 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 can be either positive or negative, so the type must be "signed". 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 a signed integer ("long" or "signed long" in C/C++; "I32" 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. **Z-Axis Usage in 2-Axis Systems:** On an MT-800, MP-78, or MP-88 system, the Z axis can be used as a focus drive, with a conversion factor that may be custom according to the make and model of the microscope being used.

12. **Setting Resolution & Velocity**: The Set Resolution & Velocity ('V') command unsigned 16-bit value can be easily composed mathematically using the following formula:

    *unsigned short ResSpeed = (Res * 0x8000) + Speed*

    where "ResSpeed" is the final unsigned 16-bit value (Little Endian bit order), "Res" is the resolution (0 for Low; 1 for High), 0x8000 (32,768 decimal) as a multiplier positions the resolution (0 or 1) to Bit 15 (the high order bit), and then the "Speed" value is simply added to occupy Bits 14 through 0. The "unsigned short" is a C/C++ data type definition that ensures that "ResSpeed" is a 16-bit variable that holds only positive values.

13. **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.

Table 7. Status data structure (as returned by Get Status ('s') command.

Offset	Length	Name	Description							
0	8 bits	**FLAGS**	**Bit**	**Name**	**Description**	**Values**				
			0-3	**SETUP #**	Currently loaded setup number coded in BCD (decimal digit 0-9)	**Binary-Coded Decimal (BCD)**				**Dec. Digit**
						3	**2**	**1**	**0**	
						0	0	0	0	0
						0	0	0	1	1
						0	0	1	0	2
						0	0	1	1	3
						0	1	0	0	4
						0	1	0	1	5
						0	1	1	0	6
						0	1	1	1	7
						1	0	0	0	8
						1	0	0	1	9
						1 (Set)		**0 (Clear)**		
			4	**ROE_DIR**	Last ROE direction	Negative		Positive		
			5	**REL_ABS_F**	Display origin	Absolute		Relative		
			6	**MODE_F**	Manual mode flag	Continuous		Pulse		
			7	**STORE_F**	Setup condition	Stored		Erased		
1	Byte	**UDIRX**	User-defined values for motor axis directions. Valid values: 0-5							
2	Byte	**UDIRY**								
3	Byte	**UDIRZ**								
4	Word	**ROE_VARI**	Microsteps per ROE click							
6	Word	**UOFFSET**	User-defined period start value							
8	Word	**URANGE**	User-defined period range							
10	Word	**PULSE**	Number of microsteps per pulse							
12	Word	**USPEED**	Adjusted pulse speed microsteps per sec.							
14	Byte	**INDEVICE**	Input device type							
15	8 bits	**FLAGS_2**	**Bit**	**Name**	**Description**	**1 (Set)**		**0 (Clear)**		
			0	**LOOP_MODE**	Program loops	Do loops		Execute once		
			1	**LEARN_MODE**	Learn mode status	Learning now		Not learning		
			2	**STEP_MODE**	Resolution (microsteps/step)	50		10		
			3	**SW2_MODE**	Joystick side button	Enabled		Disabled		
			4	**SW1_MODE**	Enable FSR/Joystick	Enabled		Cont/Pulse (keypad)		
			5	**SW3_MODE**	ROE switch	Enabled		Disabled		
			6	**SW4_MODE**	Switches 4 & 5	Enabled		Disabled		
			7	**REVERSE_IT**	Program sequence	Reverse		Normal		
16	Word	**JUMPSPD**	"Jump to max at" speed							
18	Word	**HIGHSPD**	"Jumped to" speed							
20	Word	**DEAD**	Dead zone, not saved							
22	Word	**WATCH_DOG**	Programmer's function (analog input for overload protection)							
24	Word	**STEP_DIV**	Microns ←→ Microsteps conversion factor. See Note 2.							
26	Word	**STEP_MUL**	Microns ←→ Microsteps conversion factor. See Note 2.							
28	Word	**XSPEED**	Velocity (microns/sec., Bits 14 - 0) & resolution (0 or 1, Bit 15). See Note 3.							
30	Word	**VERSION**	Firmware version. See Note 4.							
32	Byte		End of received data terminator (ASCII CR (13 decimal or 0D hexadecimal))							

NOTES:

1. All values are stored in Little-Endian bit order. All byte values are ordered Bit 7 through Bit 0. All "word" (16-bit) values are ordered Bit 15 through Bit 0. To reverse the bit order of word values to Big Endian, swap positions of both bytes (least significant byte becomes most significant and most significant becomes least significant).
2. **STEP_DIV and STEP_MUL:** Both contain 16-bit values used as factors for converting positional values between microns and microsteps. See the *Microns/microsteps conversion factors* table for what the values need to be for conversions. Position values in microns are typically stored in "double" (for double-precision floating-point) data type variables, while positions in microsteps are stored as 32-bit signed integer variables data-typed as "signed long". "double" and "signed long" (or just "long") are C/C++ data types. Both conversion factors as copied or derived from STEP_DIV and STEP_MUL should be stored in "double" data type variables so they can be used as multipliers to facilitate accurate conversions between double-precision microns and 32-bit integer microsteps.

In the C/C++ examples below "status_data_block" is the address of the data returned by the 's' command, and "double" is the data type for double-precision floating-point variables.

```
/* define both conversion factors as double-precision floating
point variables */
double um2usCF, us2umCF;
```

MP-285: STEP_DIV contains the microsteps/micron conversion factor. STEP_MUL contains the microns/microstep conversion factor (the reciprocal of STEP_DIV) * 100.

```
/* Get microsteps/microns conversion factor */
us2umCF = (double)((unsigned short)status_data_block[24]);
/* Get microns/microstep conversion factor */
um2usCF = (double)((unsigned short)status_data_block[26]) / 100;
```

For example, if the controller is configured for an MP-285/M micromanipulator or derived device (3DMS-285 or MP-78 stage, or MOM or SOM objective mover), then STEP_DIV contains 25 and STEP_MUL contains 4 (0.04 (the actual reciprocal of STEP_DIV * 100).

If the controller is configured for an MT-800 XY Translator, then STEP_DIV contains 20 and STEP_MUL contains 5 (0.05 * 100).

MP-285A: Both STEP_DIV and STEP_MUL contain the distance travelled by ten microsteps, expressed in nanometers (where 1 nanometer = 0.001 micron). To get the length of one microstep in nanometers, divide the field's value by 10 and then again by 1000. To get the number of microsteps required to move one micron, take the reciprocal of the length of one microstep (in microns).

```
/* microns/microstep conversion factor: Divide by 10 for
nanometers, then by 1000 for microns */
um2usCF = (double)((unsigned short)status_data_block[26]) / 10000;
/* microsteps/micron: Reciprocal of microns/microstep */
us2umCF = 1 / um2usCF;
```

For example, if the controller is configured for an MP-285/M micromanipulator or derived device (3DMS-285 or MP-78 stage, or MOM or SOM objective mover), then both fields contain 400, meaning 10 microsteps = 400 nanometers. The length of one microstep is therefore 400/10 = 40 nanometers, or 400/10000 = 0.04 microns. The number of microsteps needed to move one micron is 1/0.04 = 25 microsteps. Thus, the conversion factors for the MP-285/M are 0.04 microns/microstep and 25 microsteps/micron.

3. **XSPEED**: Contains an unsigned 16-bit value ("unsigned short" or "WORD") with both the Resolution (Low or High) and the speed (microns/sec) encoded within it. The Resolution is stored in the high-order bit (Bit 15), and the speed is stored in the remaining bits (Bits 14 through 0). Extracting both values can be done in the following way (C/C++):

```
/* "status_data_block" is the name of the address of the data
returned by the 's' command. "unsigned" is a data type prefix that
indicates positive numbers only */
unsigned short XSPEED, Speed, B15; /* 16-bit variables */
unsigned char Res;                 /* 8-bit variable */
/* read 16 bits from "status_data_block" at Index (offset) 28 */
XSPEED = (unsigned short)status_data_block[28];
Res    = 0;      /* assume Low Res */
Speed  = XSPEED; /* assume Low Res speed */
B15    = 0x8000; /* Bit 15 position value (32758 dec.) */
if (XSPEED >= B15)  /* if High Res . . . */
{
    Res   = 1;            /* set Res to High */
    Speed = (XSPEED – B15) /* extract High Res speed */
}
```

VERSION: Contains the version of the controller's firmware * 100. To extract the version, divide by 100 (e.g., 302 / 100 = 3.02 (3 is the major version number and 02 is the minor version)).

```
/* Get the version as a 16-bit positive integer value */
unsigned short VERSION =
             (unsigned short)status_data_block[30];
/* major version integer */
unsigned short ver_major = VERSION / 100;
/* minor version integer */
unsigned short ver_minor = VERSION % 100;
/* full version floating-point value */
float Ver = ((float)VERSION) / 100;
```

Table 8. Error codes.

ASCII Char.	Value			Error	Description
	Dec.	Hex.	Binary		
0	**48**	**30**	**00110000**	**SP Overrun**	The previous character was not unloaded before the latest was received
1	**49**	**31**	**00110001**	**Frame Error**	A valid stop bit was not received during the appropriate time period
2	**50**	**32**	**00110010**	**Buffer Overrun**	The input buffer is filled, and CR has not been received
4	**51**	**34**	**00110011**	**Bad Command**	Input cannot be interpreted – command byte not valid
8	**56**	**38**	**00111000**	**Move Interrupted**	A requested move was interrupted by input on the serial port. This code is ORed with any other error code. The value normally returned is "<", i.e., '8' (38h) ORed with '4' (34h) = '<' (3Ch). '4' is reported on the vacuum fluorescent display. '8' | '0' = '8' (38h | 30h = 38h) '8' | '1' = '9' (38h | 31h = 39h) '8' | '2' = ':' (38h | 32h = 3Ah) '8' | '3' = ';' (38h | 33h = 3Bh) '8' | '4' = '<' (38h | 34h = 3Ch)

Table 9. MP-285[A] Programmed robotic move external commands.

Name	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (len.)	Value			Alt-key-pad	Ctrl-char	ASCII def./-char.	Details
					Dec.	Hex.	Binary				
Download Program ('d') (to the controller)	**Tx**	All	3+ (*n**12) +1	0	100	64	0110 0100	0100		**'d'**	Downloads a sequence of vectors to the controller to be stored in a specified program number.
				1	1 - 10	01 - 0A	0000 0001 - 0110 1010	0001 - 0010			Program number (1-byte unsigned integer): 1 – 10.
				2	1 - 99 (*n*)	01 - 63	0000 0001 - 0110 0011	0001 - 0099			Number of vectors (*n*) in the program (1-byte unsigned integer): 1 – 99.
				o=2	A vector consists of a 32-bit signed "long" integer value (Little-Endian) in 4 bytes for the vector type descriptor and for each axis (X, Y, & Z), for a total of 16 bytes. *o*=current offset; *v*=1. If *n* > 0, enter loop and send next 16 bytes for Vector *v*.						
				o+1 (4)	3,490, 119, 680	D007 0000	1010 0000 0000 0111 0000 0000 0000 0000				Vector type: Vector
					0	0000 0000	0000 0000 0000 0000 0000 0000 0000 0000				Vector type: Pause
				o+5 (4)							Vector *v* X–axis distance in μsteps
				o+9 (4)							Vector *v* Y–axis distance in μsteps
				o+13 (4)							Vector *v* Z–axis distance in μsteps
				o+17	*o*=offset of last byte in vector. If *n* > 0, decrement *n*, increment *v*, and loop for next vector. Else, exit loop						
				o+1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1	1	13	0D	0000 1101	0013	^M	<CR>	Task-completion indicator
Execute Stored Program ('k')	**Tx**	All	3	0	107	6B	0110 1011	0107		**'k'**	Executes (runs) a specified program stored in the controller.
				1	1-10	01-0A	0000 0001 - 0110 1010	0001 - 0010			Program number (1-byte unsigned integer): 1 – 10.
				2	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1	1	13	0D	0000 1101	0013	^M	<CR>	Task-completion indicator

Name	Tx/-Delay/-Rx	Ver.	Total Bytes	Byte Offset (len.)	Value			Alt-key-pad	Ctrl-char	ASCII def./-char.	Details
					Dec.	Hex.	Binary				
Upload Program ('u') (to the computer)	**Tx**	All	3	0	100	64	0110 0100	0100		**'u'**	Uploads to the computer a sequence of vectors stored in controller's specified program number.
				1	1-10	01-0A	0000 0001 - 0110 1010	0001 - 0010			Program number (1-byte unsigned integer): 1 – 10.
				2	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1+ (*n**12) +1	0	1 - 99 (n)	01 - 63	0000 0001 - 0110 0011	0001 - 0099			Number of vectors in the program (1-byte unsigned integer), referenced in this table as "*n*".
				o=0	A vector consists of a 32-bit signed "long" integer value (Little-Endian) in 4 bytes for the vector type descriptor and for each axis (X, Y, & Z), for a total of 16 bytes. *o*=current offset; *v*=1. If *n* > 0, next 16 bytes are returned for Vector *v*, so enter loop.						
				o+1 (4)	3,490, 119, 680	D007 0000	1010 0000 0000 0111 0000 0000 0000 0000				Vector type: Vector
					0	0000 0000	0000 0000 0000 0000 0000 0000 0000 0000				Vector type: Pause
				o+5 (4)							Vector v X–axis μsteps if vector or 0 for pause
				o+9 (4)							Vector v Y–axis μsteps if vector or 0 for pause
				o+13 (4)							Vector v Z–axis μsteps if vector or 0 for pause
				o+17	*o*=last byte in vector. If *n* > 0, decrement *n*, increment *v*, and loop for next vector. Else, exit loop						
				o+1	13	0D	0000 1101	0013	^M	<CR>	Task-completion indicator
Continue After Pause ('e')	**Tx**	All	2	0	101	65	0110 0101	0101		**'e'**	Command
				1	13	0D	0000 1101	0013	^M	<CR>	Terminator
	Rx		1	0	13	0D	0000 1101	0013	^M	<CR>	Task-completion indicator

NOTES:

1. "Download" means sending a program to the controller (computer --> controller). "Upload" means receiving a program from the controller (controller --> computer).
2. Each vector is 36 bytes (three sets of 12 bytes, each consisting of three contiguous 32-bit signed values (4 bytes each) for X, Y, and Z, in that order).
3. The following commands/functions can be inserted before any vector:
   Absolute Mode ('a')
   Relative Mode ('b')
   Set Velocity & Resolution ('V')
   Pause & duration

NOTES:

NOTES: