ROBO CYLINDER

Serial Communications Protocol

IAI Corporation

1. Preface

This manual contains information pertaining to communicating serially with the Robo Cylinder controller using IAI's proprietary protocol communications format. For information beyond serial protocol communications, please refer to the "Robo Cylinder Operation Manual"

Caution!

Using commands or strings not specifically described in this manual may cause the system to behave improperly, potentially causing damage to either itself or its environment.

2. Communications Specifications

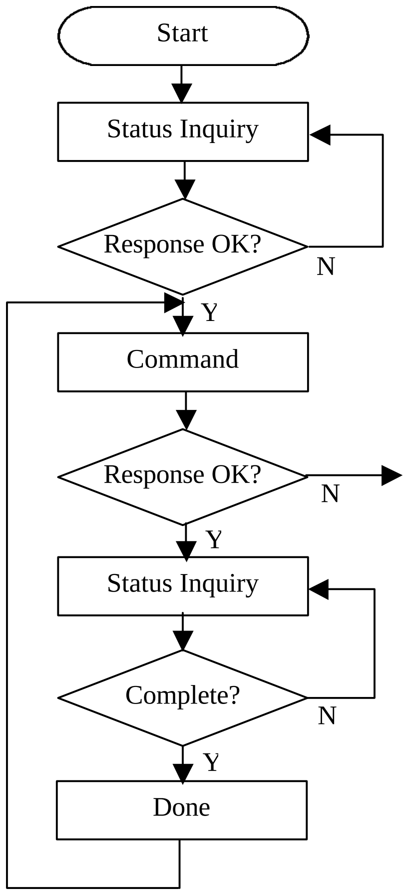
	Item	Description
1	Electrical Specs	EIA RS485
2	Synchronization Method	Asynchronous
3	Connection Method	Differential Line
4	Connector	6 pole modular
5	String Format	ASCII
6	Baud Rate	38.4 Kbps *
7	Data Bits	8
8	Parity	None
9	Stop Bit	1
10	Communication Method	Half-Duplex

Note: The Robo Cylinder PC interface software supplies the recommended serial communication cable and RS485→RS232 adapter.

* The default baud rate is 38.4 Kbps. However, communications speeds from 9.6 Kbps up to 115.2 Kbps may be obtained. Please contact Intelligent Actuator if other speeds are required.

3. Communication Procedure

General procedure for communicating with the Robo Cylinder controller.



Check to ensure good communications has been established before proceeding.

Send desired command (be sure to home axis before moving to point data)

If error occurs, be sure to check the error code list to correct the problem.

If necessary, ensure the command was completed successfully before proceeding.

4. Communication Format

4-1 General Information

1) All communication is via ASCII characters.
2) Data format is fixed at 16 characters.
3) Up to 16 axes can be linked serially. These are referred to via the axis number in the data strings in hexadecimal format (0-F).
4) Each controller can contain up to 16 positions. These are referred to via the position number in the data strings in hexadecimal format (0-F).
5) Communication integrity is checked via the Block Check Characters (BCC). The BCC is obtained by first ignoring the STX, BCC and ETX codes from the string and summing the remaining hexadecimal ASCII codes (aside from the STX, BCC and ETX characters, the code is 12 characters long). Next, take the 2's complement of the result. Finally, the least significant byte of the 2's complement is used as the BCC.

   BCC calculation example:
   [STX]0Q3010000000[BCC][ETX]
   The sum of the hexadecimal ASCII codes of the 12-character data string is as follows:
   30H+51H+33H+30H+31H+30H+30H+30H+30H+30H+30H+30H = 265H
   The 2's complement 265H is D9B of which the least significant byte is 9B. Therefore the BCC is 9B.

   * 2's complement is performed by representing the data in 16 bit form, flipping each bit (0→1, 1 → 0) and adding 1.

4-2 Data Strings

Status Inquiry	STX	Axis#	'n'	10 continuous 0's										BCC		ETX
	02		6E	30	30	30	30	30	30	30	30	30	30			03
Status Response	STX	'U'	Axis#	'n'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		6E									30			03

BCC Data

*The first row of both the send and receive is in ASCII characters, the second is in hexadecimal ASCII code. The actual strings sent are ASCII characters.
**Information regarding the contents of Status, Alarms, IN and OUT can be found in section 4.3 Description of Codes.

Position Move	STX	Axis#	'Q3'		'01'		Pos #		5 continuous 0's					BCC		ETX
	02		51	33	30	31	30		30	30	30	30	30			03
Pos Move Response	STX	'U'	Axis#	'Q'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		51									30			03

*The Pos# is in hexadecimal format representing position 0-15 as 00-0F.

Home Command	STX	Axis#	'o'	Direction		8 continuous 0's								BCC		ETX
	02		6F	30	37	30	30	30	30	30	30	30	30			03
Home Response	STX	'U'	Axis#	'o'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		6F									30			03

*Direction determines whether the axis homes toward the motor or away from the motor. Typical home direction (motor end) is 07, non-motor end is 08.

Absolute Move	STX	Axis#	'a'	Position Data (HEX)								'0'		BCC		ETX
	02		61									30	30			03
Abs Move Response	STX	'U'	Axis#	'a'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		61									30			03

*Position data is a hexadecimal representation of encoder pulses. Please see section 4.3 Description of codes for full explanation.

VEL/ACC Command	STX	Axis#	'v'	'2'	Velocity				Acceleration				'0'	BCC		ETX
	02		76	32									30			03
VEL/ACC Response	STX	'U'	Axis#	'v'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		76									30			03

*For detailed information concerning the Velocity and Acceleration field data, please refer to section 4.3 Description of Codes

Servo ON/OFF	STX	Axis#	'q'	0/1	9 continuous 0's									BCC		ETX
	02		71		30	30	30	30	30	30	30	30	30			03
Servo Response	STX	'U'	Axis#	'q'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		71									30			03

*0/1 signifies servo OFF/ON where 1=ON, 0=OFF.

Increment Move	STX	Axis#	'm'	Position Data (HEX)								'0'		BCC		ETX
	02		6D									30	30			03
Inc Move Response	STX	'U'	Axis#	'm'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		6D									30			03

*Position data is a hexadecimal representation of encoder pulses from current location (not from home). Please see section 4.3 Description of codes for full explanation.

Stop Motion	STX	Axis#	'd'	10 continuous 0's										BCC		ETX
	02		64	30	30	30	30	30	30	30	30	30	30			03
Stop Response	STX	'U'	Axis#	'd'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		64									30			03

*Motion stops regardless of position or previous command. This command may be used in conjunction with an absolute or incremental move to jog the system.

Position Inquiry	STX	Axis#	'R4'		'000074000'									BCC		ETX
	02		52	34	30	30	30	30	37	34	30	30	30			03
Position Response	STX	'U'	Axis#	'R4'		Position Data (HEX)								BCC		ETX
	02	55		52	34											03

*Position data is a hexadecimal representation of encoder pulses. Please see section 4.3 Description of codes for full explanation.

The following commands work in conjunction to write data to the point table within the controller. All 4 commands must be used in succession in order to write 1 piece of information to an existing point in the point table (i.e. positional data for point number 1). The address in the T4 command determines the type of data written (see section 4.3 Description of Codes for more details). When writing a new point to the point table, all 6 fields must be entered which means that a total of 14 commands must be sent to the controller for each new point in the point table.

A→B Transfer	STX	Axis#	'Q1'		'01'		Pos #		5 continuous 0's					BCC		ETX
	02		51	31	30	31	30		30	30	30	30	30			03
A→B Response	STX	'U'	Axis#	'Q'	Status		Alarms		IN		OUT		'0'	BCC		ETX
	02	55		51									30			03

*The Pos# is in hexadecimal format representing position 0-15 as 00-0F.

Address Allocation	STX	Axis#	'T4'		Address (HEX)								'0'	BCC		ETX
	02		54	34									30			03
Address Response	STX	'U'	Axis#	'T4'		Address (HEX)								BCC		ETX
	02	55		54	34											03

*See section 4.3 Description of Codes for more details about the Address contents.

Data Write	STX	Axis#	'W4'		Data								'0'	BCC		ETX
	02		57	34									30			03
Data Response	STX	'U'	Axis#	'W4'		Address (HEX) + 1								BCC		ETX
	02	55		57	34											03

*See section 4.3 Description of Codes for more details about the Data and Address contents.

B→A Transfer	STX	Axis#	'V5'		'01'		Pos #		5 continuous 0's					BCC		ETX
	02		56	35	30	31	30		30	30	30	30	30			03
B→A Response	STX	'U'	Axis#	'V5'		Accumulated Number of Writes								BCC		ETX
	02	55		56	35											03

*The Pos# is in hexadecimal format representing position 0-15 as 00-0F. The Accumulated Number of Writes counts the number of times a field has been written.

4-3 Definition of Codes

1) Status Command

The status command is as follows:

[STX]+[Axis #]+[‘n’]+[‘0000000000’]+[BCC]+[ETX]

Character	Description
STX	Start Text character (Hex ASCII code: 02)
Axis #	Axis # as specified on controller dipswitches (0-F)
BCC (2 characters)	Sum Check characters (2’s complement of the sum of the 12 data characters)
ETX	End Text character (Hex ASCII code: 03)

2) Status Response:

The status response is as follows:

[STX]+[‘U’]+[Axis #]+[‘n’]+[STATUS]+[ALARM]+[IN]+[OUT]+[BCC]+[ETX]

STATUS
The status portion consists of 2 characters (1 byte) in hexadecimal format representing the following table of bits (bit# 0 is least significant, bit# 7 most significant):

Bit #	Description
7	Command refusal (0=OK, 1=refused) If 1, see alarm code
6	Not used
5	Not used
4	Not used
3	Home Status (0=Home not complete, 1=Home complete)
2	Run Status (0=not ready to move, 1=servo on and ready to move)
1	Servo Status (0=Servo Off, 1=Servo On)
0	Power Status (0=Power Off, 1=Power On)

ALARM

The alarm portion consists of 2 characters (1 byte) in hexadecimal format. The alarm meaning is shown in the table below:

Alarm Code	Description	Level
00	No Alarm	WARNING
5A	Receive Buffer Overflow	
5B	Receive Buffer Framing Error	
5D	Header Abnormal Character	
5E	Delimiter Abnormal Character	
5F	BCC Error	
61	Received Bad Character	
62-64	Incorrect Operand	
70	Tried to move while run status was off	
74	Tried to move during motor commutation	
75	Tried to move while homing	
B1	Position data error	ALARM
B8-B9	Motor commutation error	
BB-BE	Bad encoder feedback while homing	
C0-C1	Excess speed / servo error	
C8	Excess current	
D0-D1	Excess main power voltage / over-regeneration	
D8	Deviation error	
E0	Overload	
E8-EC	Encoder disconnect	
ED-EE	Encoder error	
F8	Corrupt memory	

INPUTS/OUTPUTS

The IN and OUT portions of the response are both 2 characters (1 byte) in hexadecimal format representing the PIO input and output status shown in the following table of bits (bit# 0 is least significant, bit# 7 most significant).

IN

Input #	Description
7	Hold
6	Not Used
5	Not Used
4	Start
3	Pos # 8
2	Pos # 4
1	Pos # 2
0	Pos # 1

OUT

Output #	Description
7	Alarm
6	Zone
5	Home Complete
4	Move Complete
3	Pos # 8
2	Pos # 4
1	Pos # 2
0	Pos # 1

3) Position data

Position data is used in several commands including absolute move, position data write, and position inquiry. The position data is in hexadecimal format representing encoder pulses. For a system that homes to the motor end, the equations necessary to convert from millimeters from home to encoder pulses is as follows (please note that rounding errors may occur):

Pos data (pulses) = FFFFFFFF – (pos data (mm) X 800 (pulses/rev)/lead (mm))

Therefore, for 0mm, the position data in the string is actually FFFFFFFF.

For non-motor end homing, the equation becomes as follows:

Pos data (pulses) = Pos data (mm) X 800 (pulses/rev)/lead (mm)

For incremental moves, positive direction is represented by the equations shown above. However, negative moves are represented by the equations for the opposite homing direction. For example, a positive move of 0.5 mm for a 2.5mm lead system that homes to the motor end would be 'FFFFFF5F'. However, a negative 0.5mm move with the same system would be '000000A0'.

4) Address

The address field occurs in strings such as Address Allocation and Data Response. The address field identifies the point table field that is accepting the data being written. The following list of addresses identifies those fields that are currently available:

Address Location	Field
00000400	Position data
00000403	Position band
00000404	Velocity
00000405	Acceleration/Deceleration
00000406	Push %
00000407	Push recognition time
00000409	Max Acc flag

More address locations will be added as they become available.

Position data and position band are in millimeters and follow the format shown above in section 4. Velocity and acceleration data is formatted as shown below in section 5. Push percentage is a hexadecimal representation of the push percentage multiplied by the screw lead for that system. In other words,

Push % data = Push percentage * screw lead (mm)

Push recognition time is a hexadecimal representation of the time (in msec) that the push force must be exceeded before the system records that the push is complete. However, this value should not exceed 000000FF (or 255 msec).

The Max Acc flag value when the push % is zero is either 0 or 1. When the push % is non-zero, the value is either 6 or 7 (where 6 means Max Acc=0, and 7 means Max Acc=1).

5) Velocity and Acceleration

Velocity and acceleration values must be calculated according to the following equations:

VEL(0.2rpm) = VEL(mm/s) X 300/lead(mm)
ACC(0.1rpm/msec) = ACC(G) X 5883.99/lead(mm)

Convert these values to hexadecimal before entering them into the string.

6) ASCII Chart (Hex)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	NL	SH	SX	EX	ET	EQ	AK	BL	BS	HT	LF	HM	CL	CR	SO	SI
1	DE	D1	D2	D3	D4	NK	SN	EB	CN	EM	SB	EC	→	←	↑	↓
2	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	P	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DL

7) Actuator Screw Lead Chart

RC SCREW LEADS		ACTUATOR TYPE							
		S5	S6	SS	SM	SSR	SMR	RS	RM
SPEED	L	3	3	3	5	3	5	2.5	4
	M	6	6	6	10	6	10	5	8
	H	12	12	12	20	12	20	10	16

5. Examples

The following examples contain strings that might be sent to the controller. In each case, the string will use the following format:

Chr$(02) + string + Chr$(03)

Where Chr$(02) is the STX character and Chr$(03) is the ETX character. These characters are necessary for the string, but they are not characters that are printable. Therefore, these representations will be used (coincidentally the same format as used in many programming languages).

5-1 Block Check (BCC) Computation Examples

The following examples show sample strings and their associated BCC characters:

1) Status inquiry – 0n0000000000

First, add the hex ASCII codes of the characters:

30+6E+30+30+30+30+30+30+30+30+30+30 = 27E

27E(Hex)=1001111110(Bin)

Next take 2's complement:

1001111110 => 0110000010 (Bin) = 182 (Hex)

The BCC is then the last 2 characters of the result:

BCC = 82

The resultant string becomes:

Chr$(02) + 0n000000000082 + Chr$(03)

2) Position move – 1Q3010600000

BCC = 94

The resultant string becomes:

Chr$(02) + 1Q301060000094 + Chr$(03)

5-2 General Command Examples

1) Home

The following string is for homing a system addressed as axis #3 toward the motor end:

Chr$(02) + 3o070000000077 + Chr$(03)

2) Velocity/Acceleration

For a system with a screw lead of 2.5mm that is addressed as #2, a velocity of 100mm/s and acceleration of 0.2G can be set by the following string:

Chr$(02) + 2v22EE001D602F + Chr$(03)

Where:
Vel = 100mm/s X 300 / 2.5mm = 12000 = 2EE0 (Hex)
Acc = 0.2G X 5883.99 / 2.5mm = 470.7192 = 1D6 (Hex)
BCC = 2F

3) Position Move

The following string will move a system addressed 0 to position number 11:

Chr$(02) + 0Q3010B0000089 + Chr$(03)

4) Absolute Move

For a system with a screw lead of 6mm that is addressed as #12 and is homed to the motor end, the following string accomplishes moving to 56.80mm:

Chr$(02) + CaFFFFE26A00F6 + Chr$(03)

Position = 56.8mm X 800 / 6mm = 7573.33 = 1D95 (Hex)

However, since the system homes to the motor end, the position data that is sent is:

FFFFFFFF – 1D95 = FFFFE26A

5) Incremental Move

To move the same system shown in the example above +100mm from its current position, send the following string:

Chr$(02) + CmFFFFE26A00F6 + Chr$(03)

Moving that same system –100mm from its current position requires the following string:

Chr$(02) + Cm00001D95004D + Chr$(03)

6) Servo ON/OFF

To turn on the servo for a system addressed as #1, send the following string:

Chr$(02) + 1q10000000007D + Chr$(03)

To turn the same servo off, send the following string:

Chr$(02) + 1q00000000007E + Chr$(03)

7) Position Inquiry

To poll the current position of a system addressed as #0 that has been homed to the motor end and has a 12mm lead, send the following string:

Chr$(02) + 0R40000740008F + Chr$(03)

In this case, the following response is of particular interest:

Chr$(02) + U0R4FFFF167AFE + Chr$(03)

Position = FFFF167A, therefore the actual position is:

(FFFFFFFF(Hex) – FFFF167A(Hex)) X 12mm / 800 = 896.72mm

5-3 Point Table Write Examples

The following examples show how to write data to the point table.

1) Change Position Data in Existing Point in Point Table

See section 4-2 for more information on these strings.

In order to set the point data for point# 14 to 32.45mm in a system addressed as #5 with a 8mm lead screw that homes toward the motor end, enter the following strings in succession:

Chr$(02) + 5Q1010E0000083 + Chr$(03)

Chr$(02) + 5T40000040008F + Chr$(03)

Chr$(02) + 5W4FFFFF352018 + Chr$(03)

Chr$(02) + 5V5010E000007A + Chr$(03)

Where the W4 command dictates the position data as follows:

FFFFFFFF – (32.45 X 800 / 8)(Hex) = FFFFF352 (Hex)

2) Enter New Point in Point Table

Using the same system as in the previous example and setting the point data the same, set the velocity to 100mm/s, acceleration to 0.2G, push force to 0, position band to 0.1mm and max acc flag to 0 with the following strings:

Chr$(02) + 5Q1010E0000083 + Chr$(03)

Chr$(02) + 5T40000040008F + Chr$(03)

Chr$(02) + 5W4FFFFF352018 + Chr$(03)

Chr$(02) + 5T40000040408B + Chr$(03)

Chr$(02) + 5W400000EA6064 + Chr$(03)

Chr$(02) + 5T40000040508A + Chr$(03)

Chr$(02) + 5W400000093084 + Chr$(03)

Chr$(02) + 5T40000040108E + Chr$(03)

Chr$(02) + 5W4000000C007D + Chr$(03)

Chr$(02) + 5T40000040308C + Chr$(03)

Chr$(02) + 5W40000000A07F + Chr$(03)

Chr$(02) + 5T400000409086 + Chr$(03)

Chr$(02) + 5W400000000090 + Chr$(03)

Chr$(02) + 5V5010E000007A + Chr$(03)

Please refer to the address descriptions in item 4 of section 4-3 for further information for each string.

5-4 Jog Examples

It is possible to jog the system using serial communications. Simply send the system to one end of the stroke to start the jog, and then send a stop command to end the jog.

1) Jog Forward

These strings will start jogging a system forward and then stop that jogging with the stop command:

Chr$(02) + 0aFFFF65430025 + Chr$(03)

Chr$(02) + 0d00000000008C + Chr$(03)

2) Jog Backward

These strings will start jogging a system backward and then stop that jogging with the stop command:

Chr$(02) + 0aFFFFFFFF00DF + Chr$(03)

Chr$(02) + 0d00000000008C + Chr$(03)