

Working with the Input Buffer for Serial and Ethernet Raw Socket Ports

*How to manipulate ASCII information from input buffer with
2600/2700/5X00 controllers
April, 2005*

3



Control Technology Corporation, Hopkinton, MA • 800.282.5008 • www.ctc-control.com

This document shows you how to manipulate ASCII information from the input buffer with 2600/2700/5X00 controllers. You can manipulate the serial buffers on all 2600/2700/5X00 controllers.

The 5X00 controller also allows you to manipulate a RAW Socket port. You do this by assigning a port (3-7) as a TCP/IP Raw Socket as described in the 5100 Communications Protocol Guide. You can then treat the port just like a serial port to receive and transmit ASCII messages.

Register Info

Registers needed to work with incoming port data and a brief description of each.

Serial Communications Registers

12000	Select Controller Communications Port: W access, 1 = COM1, 2 = COM2, etc. <i>Note: 3 - 7 = TCP raw virtual socket connections are the Raw Socket ports available on the Blue Fusion Controllers only</i>
12000	Message Transmission Status for Controllers: R access, 0 = not busy, 1 = busy.
12001	Transmit Message from Data Table: W only, Store row number to transmit.
12001-12255	Controller Receive Buffer Access, R only, 1 character per location.
12300	Protocol Variation: R/W, Controls RS-232 terminal protocol modes. 0 = computer, 1 = terminal (default)
12301	Serial Baud Rate Selection: R/W only, 2 = 1200, 3=2400, 4 = 4800, 5 = 9600, 6 = 19.2K (default), 7 = 38.4K.
12302	Serial Input Buffer Counter – (R) number of characters available. (W) any value to clear buffer and zero count.
12303	Disable Automatic Parsing, R/W, 0 = inhibits response, 1 = resumes normal response to incoming messages.
12304	Extract Number from RS-232 Receive Buffer: R only, Automatically assembles ASCII strings into a numeric value. The result is a signed 32-bit number. Automatically assembles strings of ASCII characters containing numeric information into a numeric value. Number multiplied by 10,000, allowing decimal points to 4 places.
12305	Communications Priority: R/W, When running multiple tasks. 0 = normal, 1 = priority.
12308	Serial Parity: R/W, 0=None (default), 1=Odd, 2= Even
12309	Serial Stop Bits: R/W, 1 (default) or 2
12310	Serial Data Bits: R/W, 7 or 8 (default)
12316	Message String Transfer Register: R/W, write record number of message.ini file to send out serial port selected in 12000 register, read returns status with 0 = success.

Initial Port Setup

If you're using serial communications, you need to make sure your CTC Serial Port Settings match the device sending the data. For info on how to set up a TCP Raw Socket see the TCP/IP Raw Sockets section 2.0 in the 5100 Communications Protocol Guide.

Register 12000 sets the current port number. Any times you want to change a port setting, read the buffer from a port, or transmit a message you need to make sure you are working with the correct port. The following is example code that sets up Comm. Port 2 to 19.2K and turns off Automatic Parsing.

```
[1] Initialize_Port
;;; This step sets serial comm. port 2 to 19.2K baud and disables
;;; automatic parsing.
-----
<NO CHANGE IN DIGITAL OUTPUTS>
-----
store 2 to Serial_Comm_Port_Reg_12000
store 6 to Serial_Comm_Baud_Rate_Reg_12301
store 0 to Automatic_Parsing_Reg_12303
goto Next
```

The Input Buffer

The Input buffer is where all incoming communication data is temporarily stored. Serial and Raw Socket buffer data can be accessed in registers 12001-12255. Each register represents one ASCII Character. By changing the Serial Port register 12000 you set which communication port's buffer can be accessed at registers 12001-12255 and register 12304.

Register 12304 allows you to read numeric values without having to convert each ASCII character in the buffer to number. All numbers sent before a decimal place start at the 5th digit. All numbers sent after a decimal place start at the 4th digit and move toward the right. For example: if an ASCII string of 123.4567 is sent to the buffer, register 12304 would read 1234567, a value of 123.45 would read as 1234500 in register 12304, and a value of 123 would read 1230000.

Automatic Parsing vs. Manual Parsing

Changing parsing changes what happens when a carriage return is received. Automatic Parsing means the once the port sees a carriage return in the buffer the buffer count (Register 12302) goes to zero and the buffer is considered cleared. All new input data will be put in sequential registers starting at 12001 again. When Automatic Parsing is disabled the buffer count register is set to 255 when a carriage return is received. You must manually store a zero to register 12302 to allow more information into the buffer.

The first thing you need to do when you want to see incoming data on the serial or raw socket port is set the port you intend to read. You do this by setting your port number to the appropriate port number. You can then either look at register 12304 for numeric strings, or look at register 12001 – 12255 to see each individual ASCII character value if you need to find specific strings.

The following is an example of a program that accepts an ASCII character from an ASCII serial device. The program checks to see which ASCII character was sent in order to determine what mode the program should be in. In this example an ASCII A would put the program in Auto mode. An ASCII M would put the program in Manual mode and an ASCII J would put the program in Jog Mode.

[1] Initialize

```
;;; Initialize comm port 2, set the baud rate to 19200 and disable automatic
;;; parsing.
```

```
-----
<NO CHANGE IN DIGITAL OUTPUTS>
```

```
-----
store 2 to Comm_Port_Reg12000
store 6 to Baud_Rate_Reg12301
store 0 to Automatic_Parsing_Reg12303
goto Next
```

[2] ReadInfo

```
;;; Wait until a carriage return is seen. Since automatic parsing is disabled
;;; we know the character count register will be set to 255 when this occurs.
```

```
-----
<NO CHANGE IN DIGITAL OUTPUTS>
```

```
-----
if Character_Count_Reg12302=255 goto GetCharacter
```

[3] GetCharacter

```
;;; Now we can look at the character or characters that were sent.
;;; For this example we are just looking at the first character.
;;; Note that if we do not see an A, M or J we clear the buffer and
;;; go back to ReadInfo to wait for another entry.
```

```
-----
<NO CHANGE IN DIGITAL OUTPUTS>
```

```
-----
if Serial_Port_Buffer_Reg_12001=77 goto Manual
if Serial_Port_Buffer_Reg_12001=65 goto AutoMode
if Serial_Port_Buffer_Reg_12001=74 goto JogMode
store 0 to Character_Count_Reg12302
goto ReadInfo
```

[4] Manual

```
;;; Here you would put your code for Manual Mode. Don't forget to
;;; clear the buffer.
```

```
-----
<NO CHANGE IN DIGITAL OUTPUTS>
```

store 0 to Character_Count_Reg12302

[5] AutoMode

;;; Here you would put your code for Auto Mode. Don't forget to
;;; clear the buffer.

<NO CHANGE IN DIGITAL OUTPUTS>

store 0 to Character_Count_Reg12302

[6] JogMode

;;; Here you would put your code for Jog Mode. Don't forget to
;;; clear the buffer.

<NO CHANGE IN DIGITAL OUTPUTS>

store 0 to Character_Count_Reg12302

For info on transmitting ASCII messages see TechNote No.30 ASCII Message Transmitting
with CTC Controllers

<http://www.ctc-control.com/customer/techinfo/technotes/ASCIIMessage.pdf>

Representation of Data for ASCII Message Transmitting

ASCII Codes

Non-printing Codes		Printable Characters					
0	Nul (ends message)	32	Space	64	@	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL (rings device bell)	39	`	71	G	103	g
8	BS (backspace)	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF (line feed)	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF (form feed)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1 (XON)	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3 (XOFF)	51	3	83	S	115	s
20	DC4	52	4	84	T	116	T
21	NAK	53	5	85	U	117	U
22	SYN	54	6	86	V	118	V
23	ETB	55	7	87	W	119	W
24	CAN	56	8	88	X	120	W
25	EM	57	9	89	Y	121	Y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	delete