Readerboard User's Guide WORKING DRAFT

For readerboard hardware version 2.0.0, and firmware version 0.0.0.

Contents

List of Figures

List of Tables

Chapter 1

Protocol Description

> I don't stand on protocol. Just call me your Excellency.
>
> —Henry Kissinger

The control protocol used to display information on the readerboard sign is very simple. Commands are expressed largely in plain ASCII characters and are executed immediately as they are received.[1]

In addition to plain, printable 7-bit ASCII characters, a few control codes are recognized as described below. String data may include any 8-bit value except as otherwise indicated.

USB vs. RS-485

The protocol used to send commands to the readerboard is different depending on whether the host is sending directly to a single readerboard over a USB cable, or to (possibly) multiple readerboards over an RS-485 bus network.

USB

A readerboard connected via USB accepts the commands just as documented below, with the addition that each such command is terminated by a `^D` byte (hex value $\texttt{04}_{16}$).

[1] Technically, they may even be executed *while* they are being received.

0	$\cdots$	$n-1$	n
$\langle command \rangle$			`^D`

If there is an error parsing or executing a command, the readerboard will ignore all subsequent input until a `^D` is received, whereupon it will expect to see the start of another command. Thus, `^D` may not appear in any transmitted data except to terminate commands.

RS-485

Commands sent over RS-485 are intended to target one or more of a set of connected readerboards over a network which may also contain other Lumos-protocol-compatible devices, so they use a protocol compatible with that use case.

Each command begins with one of the following binary headers, depending on the set of target readerboard signs which should obey the command.

Single Target

To send a command to a single sign, begin with a single byte encoded as:

7	6	5	4	3	2	1	0
1	0	0	1	$\langle ad_0 \rangle$			

where $\langle ad_0 \rangle$ is the sign's address on the bus, which must be a value in the range 0–15. This byte is followed by any command as described below.

Multiple Targets

Alternatively, a command may be targetted to multiple signs by starting the command with a multi-byte code:

7	6	5	4	3	2	1	0
1	0	1	1	$\langle ad_G \rangle$			
0				$\langle n \rangle$			
0				$\langle ad_0 \rangle$			
			$\vdots$				
0				$\langle ad_{n-1} \rangle$			

where $\langle ad_G \rangle$ is the "global" device address which signals readerboards generally (see the = command below). This will send to the $\langle n \rangle$ devices addressed as $\langle ad_0 \rangle$ through $\langle ad_{n-1} \rangle$.

All Targets

If you wish to send a command to all attached readerboards, simply send the following byte ahead of the command:

7	6 5 4	3 2 1 0
1	1 0 1	$\langle ad_G \rangle$

where $\langle ad_G \rangle$ is the “global” device address which signals readerboards generally (see the = command below).

All Off

As a special case, the single byte

7	6 5 4	3 2 1 0
1	0 0 0	$\langle ad \rangle$

will cause the readerboard addressed as $\langle ad \rangle$ to turn off all LEDs. If $\langle ad \rangle$ is the $\langle ad_G \rangle$ address, then all readerboards will turn off all LEDs.

No other command bytes need to follow; this byte is sufficient to turn off the sign(s).

Subsequent Command Bytes

All subsequent bytes which follow the above binary headers *must* have their MSB cleared to `0`.

To cover cases where a value sent as part of a command must have the MSB set, we use the following escape codes:

- A hex byte $\texttt{7E}_{16}$ causes the next byte received to have its MSB set upon receipt.
- A hex byte $\texttt{7F}_{16}$ causes the next byte to be accepted without any further interpretation.

Thus, the byte $\texttt{C4}_{16}$ is sent as the two-byte sequence `7E 44`, while a literal `7E` is sent as `7F 7E` and a literal `7F` as `7F 7F`.

If there is an error parsing or executing a command, the readerboard will ignore all subsequent input until a byte arrives with its MSB set to `1`, whereupon it will expect to see the start of another command.

Command Summary

The eight discrete LEDs are intended for a simple display of status information in a manner analogous to the Busylight project by the same author.[2] To support this usage, the `F`, `S`, `X`, `*`, and `?` commands are recognized

[2]See github.com/MadScienceZone/busylight.

Command	Description	Notes
`*`	Strobe LEDs in Sequence	[1]
`?`	Query discrete LED status	[1] [2] [4]
`F`	Flash LEDs in Sequence	[1]
`L`	Light one or more LEDs steady	[3]
`S`	Light one LED steady	[1]
`X`	All LEDs off	[1]
`^D`	Abort/terminate command	
`<`	Scroll text across display	
`=`	Set operational parameters	[4]
`@`	Move current column cursor	
`A`	Select character font	
`C`	Clear matrix display	
`H`	Add histogram/bargraph data point	
`I`	Draw bitmap graphic image	
`Q`	Query matrix display status	[2] [4]
`T`	Display text on display	

[1] Busylight compatibile command
[2] Sends response (USB only)
[3] Busylight extension (not in original Busylight)
[4] USB only

Table 1.1: Summary of All Commands

in a manner compatible with how Busylight uses those same commands. These are categorized as "Busylight compatibility commands." Unlike all other commands listed here, these are recognized regardless of case. Since the readerboard has a power supply capable of illuminating all of the status LEDs at once,[3] a new command `L` is added which allows any arbitrary pattern of steady LEDs to be turned on.

The remaining commands are used for management of the matrix display. All commands are summarized in Table 1.1.

Although the rev 2 hardware supports the ability to enable the RS-485 transmitter and send data back onto the network, this is not currently implemented by the firmware, and the intent is to have all devices listen passively to RS-485 traffic at all times.

[3] The Busylight cannot, since it is powered from the host computer's USB port.

Code	Hex	Description
`^C`$\langle pos\rangle$	`03`$\langle pos\rangle$	Move current column cursor to $\langle pos\rangle$
`^D`	`04`	Never allowed in strings (command terminator)
`^F`$\langle digit\rangle$	`06`$\langle digit\rangle$	Switch current font
`^H`$\langle pos\rangle$	`08`$\langle pos\rangle$	Move cursor left $\langle pos\rangle$ columns
`^L`$\langle pos\rangle$	`0C`$\langle pos\rangle$	Move cursor right $\langle pos\rangle$ columns
`^[`	`1B`	Never allowed in strings (string terminator)

Table 1.2: Control Codes in String Values

*—Strobe Lights in Sequence

0	1	2	3	$\cdots$	n	$n+1$
*	$\langle led_0\rangle$	$\langle led_1\rangle$	$\langle led_2\rangle$	$\cdots$	$\langle led_{n-1}\rangle$	$

Each $\langle led\rangle$ value is an ASCII digit character corresponding to a discrete LED as shown in Table 1.8.

This command functions identically to the `F` command (see below), except that the lights are "strobed" (flashed very briefly with a pause between each light in the sequence).

<—Scroll Text Across Display

0	1	2	$\cdots$	n+1	n+2
<	$\langle loop\rangle$		$\langle string\rangle$		ESC

Displays the text $\langle string\rangle$ by scrolling it across the display from right to left. If $\langle loop\rangle$ is "`.`", the text is only scrolled once; if it is "`L`" then it repeatedly scrolls across the screen in an endless loop.

The text is rendered in the current font and may contain any 8-bit bytes except as otherwise noted (but avoiding ASCII control codes is wise to be safe from conflict with future control codes which may be added to the protocol). The string is terminated by an escape character (hex byte `1B`), indicated in the protocol description as `ESC`.

The string may include control codes as listed in Table 1.2.

=—Set Operational Parameters

0	1	2	3	4
=	$\langle ad\rangle$	$\langle uspd\rangle$	$\langle rspd\rangle$	$\langle ad_G\rangle$

Code	Speed	
0	300	
1	600	
2	1,200	
3	2,400	
4	4,800	
5	9,600	(default)
6	14,400	
7	19,200	
8	28,800	
9	31,250	
A	38,400	
B	57,600	
C	115,200	

Table 1.3: Baud Rate Codes

This command sets a few operational parameters for the sign. Once set, these will be persistent across power cycles and reboots.

If the $\langle ad \rangle$ parameter is "*" then the RS-485 interface is disabled entirely. Otherwise it is a value from 0–15 encoded as described in Table 1.4. This enables the RS-485 interface and assigns this sign's address to $\langle ad \rangle$.

The baud rate for the USB and RS-485 interfaces is set by the $\langle uspd \rangle$ and $\langle rspd \rangle$ values respectively. Each is encoded as per Table 1.3.

The $\langle ad_G \rangle$ value is an address in the range 0–15 which is not assigned to any other device on the RS-485 network. This is used to signal that all readerboards should pay attention to the start of the command because it might target them either as part of a list of specific readerboards or because the command is intended for all readerboards at once. This is encoded in the same way as $\langle ad \rangle$.

This command may only be sent over the USB port.

By default, an unconfigured readerboard is set to 9,600 baud with the RS-485 port disabled.

?—Query Discrete LED Status

0
?

This command causes the sign to send a status report back to the host to indicate what the discrete LEDs are currently showing. This response has the form:

0	1	2	3	4	5	6	7	8
L	$\langle led_0 \rangle$	$\langle led_1 \rangle$	$\langle led_2 \rangle$	$\langle led_3 \rangle$	$\langle led_4 \rangle$	$\langle led_5 \rangle$	$\langle led_6 \rangle$	$\langle led_7 \rangle$
F	flasher status (see below)							
S	strober status (see below)							
\n								

The flasher and strober status values are variable-width fields which indicate the state of the flasher (see `F` command) and strober (see `*` command) functions. In each case, if there is no defined sequence, the status field will be:

0	1
$\langle run \rangle$	X

Otherwise, the state of the flasher or strober unit is indicated by:

0	1	2	3	4	$\cdots$	$n+3$
$\langle run \rangle$	$\langle pos \rangle$	@	$\langle led_0 \rangle$	$\langle led_1 \rangle$	$\cdots$	$\langle led_{n-1} \rangle$

In either case, $\langle run \rangle$ is the ASCII character "`0`" if the unit is stopped or "`1`" if it is currently running. If there is a defined sequence, $\langle pos \rangle$ indicates the 0-origin position within the sequence of the light currently being flashed or strobed, encoded as described in Table 1.4. The $\langle led \rangle$ values are as given to the `F` or `*` command that set the sequence.

Note that the $\langle pos \rangle$ value may be the character "`X`" to indicate the position value of 40, so it is important to distinguish between the field value "$\langle run \rangle$`X`" vs. "$\langle run \rangle$`X@`$\langle sequence \rangle$".

The status message sent to the host is terminated by a newline character (hex byte `0A`), indicated in the protocol description above as "`\n`".

This command may only be sent on the USB port.

@—Set Column Cursor Position

0	1
@	$\langle pos \rangle$

Sets the column cursor position to the value indicated by $\langle pos \rangle$. See Table 1.4.

Value	Code	Value	Code
0–9	`0–9`	17–42	`A–Z`
10	`:`	43	`[`
11	`;`	44	`\`
12	`<`	45	`]`
13	`=`	46	`^`
14	`>`	47	`_`
15	`?`	48	`` ` ``
16	`@`	49–63	`a–o`

(Each code is the numeric value plus 48.)

Table 1.4: ASCII Encoded Integer Values (0–63)

Code	Font Description
`0`	Fixed-width 5×7 matrix plus descenders in 8th row
`1`	Variable-width version of font 0
`2`	Bold alphanumerics and special symbols

Table 1.5: Font Codes

`A`—Select Font

0	1
`A`	⟨*digit*⟩

Sets the font to use for rendering text with the `<` and `T` commands. The font codes for ⟨*digit*⟩ are listed in Table 1.5. The full text fonts support the printable ASCII characters plus a majority of the Unicode glyphs with codepoints less than 256. See Tables 1.6–1.7 for a complete font glyph listing with codepoint assignments.

`C`—Clear Matrix Display

0
`C`

Clears the matrix display so that no LEDs are illuminated. Does not affect the discrete LEDs.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	
00x				move	end		font		0*x*
01x	back				forw				
02x									1*x*
03x				esc					
04x		!	"	#	$	%	&	’	2*x*
05x	(	)	*	+	,	-	.	/	
06x	0	1	2	3	4	5	6	7	3*x*
07x	8	9	:	;	<	=	>	?	
10x	@	A	B	C	D	E	F	G	4*x*
11x	H	I	J	K	L	M	N	O	
12x	P	Q	R	S	T	U	V	W	5*x*
13x	X	Y	Z	[	\	]	^	_	
14x	‘	a	b	c	d	e	f	g	6*x*
15x	h	i	j	k	l	m	n	o	
16x	p	q	r	s	t	u	v	w	7*x*
17x	x	y	z	{	|	}	~	///	
20x	“	”	‘	’	†	‡	. . .	′	8*x*
21x	″	!!	¯	←	→	↑	↓	≠	
22x	≤	≥	≈	Γ	Δ	Ξ	Π	Σ	9*x*
23x	Ω	π	ρ	σ	—	‰			
24x		¡	¢	£	¤	¥	¦	§	A*x*
25x		©	ª	«	¬	-	®		
26x	°	±	²	³		µ	¶	•	B*x*
27x		¹	º	»	1/4	1/2	3/4	¿	
30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	C*x*
31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	D*x*
33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
34x	à	á	â	ã	ä	å	æ	ç	E*x*
35x	è	é	ê	ë	ì	í	î	ï	
36x	ð	ñ	ò	ó	ô	õ	ö	÷	F*x*
37x	ø	ù	ú	û	ü	ý	þ	ÿ	
	8	9	A	B	C	D	E	F	

Table 1.6: Font Table for Fonts #0 and #1

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	
00x				move	end		font		0*x*
01x	back				forw				
02x									1*x*
03x				esc					
04x		**!**							2*x*
05x					,		.		
06x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	3*x*
07x	**8**	**9**							
10x	©	**A**	**B**	**C**	**D**	**E**	**F**	**G**	4*x*
11x	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	
12x	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	5*x*
13x	**X**	**Y**	**Z**						
14x		🡐	🡒	🡑	🡓	↖	↗	↘	6*x*
15x	↙	←	→	↑	↓	▚	■		
16x	✓	√	∞	Œ	œ	€	∴	∵	7*x*
17x	✖	◀	▶	▲	▼	↕	♦	◊	
20x	λ	Θ	Φ	Ψ					8*x*
21x									
22x									9*x*
23x									
24x		TS1	TS2	TS3	TS4	TS5	TS6	TS7	A*x*
25x	TS8								
	8	9	A	B	C	D	E	F	

TS⟨*n*⟩ = Thin space of ⟨*n*⟩ pixels

Table 1.7: Font Table for Font #2

F—Flash Lights in Sequence

0	1	2	3	⋯	n	$n+1$
F	$\langle led_0 \rangle$	$\langle led_1 \rangle$	$\langle led_2 \rangle$	⋯	$\langle led_{n-1} \rangle$	$

Each ⟨*led*⟩ value is an ASCII digit character corresponding to a discrete LED as shown in Table 1.8. Note that the assignment of colors to these LEDs is dependent on your particular hardware being assembled that way. As an open source project, of course, you (or whomever assembled the unit) may choose any color scheme you like when building the board.

Up to 64 ⟨*led*⟩ codes may be listed. The sign will cycle through the sequence, lighting each specified LED briefly before moving on to the next one. The sequence is repeated forever in a loop until an L, S or X command is received.

Code	Light	Color
0	L_0	white
1	L_1	blue
2	L_2	blue
3	L_3	red
4	L_4	red
5	L_5	yellow
6	L_6	yellow
7	L_7	green

Table 1.8: Discrete LED Codes and Colors

If only one $\langle led \rangle$ is specified, that light will be flashed on and off. Setting an empty sequence (no codes at all) stops the flasher's operation.

The sequence is terminated by either a dollar-sign ("`$`") character or the escape control character (hex byte `1B`), indicated in the protocol diagram above simply as "`$`".

This command may be given in upper- or lower-case ("`f`" or "`F`").

H—Draw Bar Graph Data Point

0	1
`H`	$\langle n \rangle$

This command is used to draw a bar-graph element. Repeating this command causes a scrolling data display which shows a set of data samples over some period of time. The value $\langle n \rangle$ is an ASCII digit character in the range "`0`"–"`8`", and is drawn in the far-right column of the matrix display, as a column of $\langle n \rangle$ lights stacked up from the bottom row (a value of 0 results in no lights, up to 8 which is a full column of eight lights; a value of 9 is treated as if it were 8). All existing matrix data are scrolled left one column.

I—Draw Bitmap Image

0	1	2	3	4–5	6–7	
`I`	$\langle merge \rangle$	$\langle pos \rangle$	$\langle trans \rangle$	$\langle coldata_0 \rangle$	$\langle coldata_1 \rangle$	$\cdots$
$\langle coldata_{n-1} \rangle$		`$`				

Draws an arbitrary bitmap image onto the matrix display starting at column $\langle pos \rangle$, encoded as per Table 1.4. A $\langle pos \rangle$ value of "`~`" represents the current column cursor position.

Code	Transition
`.`	No transition
`>`	Scroll in from left
`<`	Scroll in from right
`^`	Scroll up from bottom
`v`	Scroll down from top
`L`	wipe left
`R`	wipe right
`U`	wipe up
`D`	wipe down
`|`	wipe left and right from middle column
`-`	wipe up and down from middle row
`?`	choose a random transition

Table 1.9: Transition Effect Codes

Each column data, from left to right, are given by ⟨*coldata*⟩ values, each of which is a two-digit ASCII hexadecimal value with the least-significant bit representing the top row of the matrix.

The column data values are terminated by either a dollar-sign (“`$`”) character or the escape control character (hex byte `1B`), indicated in the protocol diagram above simply as “`$`”.

The column cursor is moved to be after the end of the image.

If ⟨*merge*⟩ is the character “`.`” then each column’s contents is cleared before setting the pixels as per the ⟨*coldata*⟩ value. If ⟨*merge*⟩ is “`M`” the bits set in ⟨*coldata*⟩ are added to the lit pixels already in the column.

The ⟨*trans*⟩ value indicates the transition effect to use when adding the image to the display. See Table 1.9.

`L`—Light Multiple LEDs

0	1	2	3	···	n	$n+1$
`L`	$\langle led_0 \rangle$	$\langle led_1 \rangle$	$\langle led_2 \rangle$	···	$\langle led_{n-1} \rangle$	`$`

This command is identical to the `S` command (see below), except that multiple discrete LEDs can be specified, all of which are illuminated simultaneously. See Table 1.8. Note that if a strobe sequence is running (via a previous `*` command), it remains running.

The list of ⟨*led*⟩ values is terminated by either a dollar sign (`$`) character or an escape byte (hex value `1B`), represented in the protocol diagram as “`$`”.

`Q`—Query Readerboard Status

0
`Q`

This command causes the sign to send a status report back to the host to indicate the general status of the device except for the discrete LED display which may be queried using the `?` command. The response has the form:

0	1	2	3	4	5	6	7	8
`Q`	$\langle model\rangle$	`=`	$\langle ad\rangle$	$\langle uspd\rangle$	$\langle rspd\rangle$	$\langle ad_G\rangle$	`$`	`V`
$\langle hwversion\rangle$		`$`	`R`	$\langle romversion\rangle$		`$`	`S`	$\langle serial\rangle$
`$`	`M`	$\langle coldata_0\rangle$		$\langle coldata_1\rangle$		$\cdots$	$\langle coldata_{63}\rangle$	
`\n`								

The $\langle model\rangle$ field may be "`L`" for the legacy hardware the author still has lying around (but this shouldn't be something anyone else would need to see), or "`M`" for the current 64×8 matrix display hardware.

$\langle hwversion\rangle$ and $\langle romversion\rangle$ indicate the versions, respectively, of the hardware the firmware was compiled to drive, and of the firmware itself. Each of these fields are variable-width and conform to the semantic version standard 2.0.0.[4] Each is terminated by a dollar-sign (`$`) character (and thus those strings may not contain dollar signs).

The $\langle serial\rangle$ field is a variable-width alphanumeric string which was set when the firmware was compiled. It should be a unique serial number for the device (although that depends on some effort on the part of the person compiling the firmware to insert that serial number each time). Serial numbers 0–299 are reserved for the original author's use. This string is also terminated with a dollar sign.

The $\langle coldata\rangle$ bytes are sent just as with the `I` command, as hexadecimal values indicating the LEDs lit on the matrix display, with $\langle coldata_0\rangle$ being the leftmost column of the display and $\langle coldata_{63}\rangle$ being the rightmost. In each column, the least significant bit indicates the LED on the top row.

The $\langle ad\rangle$, $\langle uspd\rangle$, and $\langle rspd\rangle$ values are as last set by the `=` command (or the factory defaults if they were never changed).

The status message sent to the host is terminated by a newline character (hex byte `0A`), indicated in the protocol description above as "`\n`".

[4]See semver.org.

`S`—Light Single LED

0	1
`S`	⟨*led*⟩

Stops the flasher (cancelling any previous `F` command) and turns off all discrete LEDs. The single LED indicated by ⟨*led*⟩ is turned on. See Table 1.8. Note that if a strobe sequence is running (via a previous `*` command), it remains running.

This command may be given in upper- or lower-case (“`s`” or “`S`”).

`T`—Display Text

0	1	2	3	4	···	n+4
`T`	⟨*merge*⟩	⟨*align*⟩	⟨*trans*⟩	⟨*string*⟩		`ESC`

Displays the text ⟨*string*⟩ at the current cursor position. The cursor position is then moved past the text to be ready for the next string to be printed.

The ⟨*align*⟩ code indicates how the text is aligned on the display (TBD).

The ⟨*trans*⟩ code specifies the transition effect to be used to add this string to the display as shown in Table 1.9.

The text is rendered in the current font and may contain any 8-bit bytes except as otherwise noted (but avoiding ASCII control codes is wise to be safe from conflict with future control codes which may be added to the protocol). The string is terminated by an escape character (hex byte `1B`), indicated in the protocol description as `ESC`. (Since a dollar sign may appear in ⟨*string*⟩, the terminator must be an escape character for this command.)

The string may include control codes as listed in Table 1.2.

`X`—Turn off Discrete LEDs

0
`X`

Turns off the flasher, strober, and all discrete LEDs.

This command may be given in upper- or lower-case (“`x`” or “`X`”).