Readerboard User's Guide
WORKING DRAFT

The information in this document, and the hardware and software it describes, are hobbyist works created as an educational exercise and as a matter of personal interest for recreational purposes.

It is not to be considered an industrial-grade or retail-worthy product. It is assumed that the user has the necessary understanding and skill to use it appropriately. The author makes NO representation as to suitability or fitness for any purpose whatsoever, and disclaims any and all liability or warranty to the full extent permitted by applicable law. It is explicitly not designed for use where the safety of persons, animals, property, or anything of real value depends on the correct operation of the software.

For readerboard hardware version 3.2.1, and firmware version 0.0.0.

Contents

List of Figures

List of Tables

Chapter 1

Hardware

> People who are really serious about software should make their own hardware.
>
> —Alan Kay

As a hobby project, the design of the hardware and software for the readerboard project "grew in the telling" as new ideas sprang to mind. As of this writing, there are three different models of hardware prototypes which were designed and created.

Version 1.0.1

The initial prototype was a fairly large board, approximately $20\frac{5}{8} \times 6\frac{5}{8}''$ with an LED pitch of $\frac{1}{3}''$.

It requires an Arduino Mega 2560 or Arduino Due to drive it. A custom-made shield board attaches to the Arduino, which provides connectors for power and ribbon cables to drive the display (one to control the matrix, the other for the 8 status LEDs which appear in a horizontal row along the bottom right of the matrix).

Version 2.0.0

This version is based on the 1.0.1 design, but shrunk to a board size of about $18\frac{3}{16} \times 5\frac{3}{16}''$ with an LED pitch of $\frac{1}{4}''$. It also replaces the obsolete TPIC6B595 with the newer STPIC6D595 chip. It also relocates the ribbon cable which

connects the Arduino controller to the LED matrix, moves the eight status LEDs to a vertical column to the right of the matrix, and switches to resistor arrays for the matrix current limiting resistors, instead of the individual discrete resistors used on the 1.0.1 board.

Version 2.1.0

Version 2.1.0 is a significant change to the 2.0.0 board in terms of how it interfaces with the Arduino, although it has the same size and physical layout as the 2.0.0 board.

This one eliminates the need for the ribbon cables and shield board. Instead, the Arduino controller mounts directly to the back of the display board. This means that the lower portion of the left edge of the enclosure has all of the external connections in one place, including power, RS-485 in and out, and the USB connector(s) on the Arduino itself.

Version 3.2.1

The current version of the hardware expands on the version 2.1.0 design by replacing the single-color LEDs with RGB LEDs arranged into 24 logical rows—each of the eight physical rows is logically a row of red, green, and blue LED elements.

Firmware Implications

The boards for versions 1.0.1 and 2.0.0 were compatible and used the same Arduino shield. Thus, they use the same firmware images. The 2.1.0 board, however, which integrates the Arduino directly instead of using a separate shield with ribbon cables, assigns signals to different I/O pins, and as such needs a different firmware image.

Both images are generated from the same source files, with compile-time switches to make one or the other.

Version 3.2.1 expands the number of logical rows due to the RGB implementation, which requires a matching firmware implementation and changes to the protocol to support the new color data.

At this point in development it is unclear whether we'll continue making the different versions available using compile-time switches or if we'll just support this version going forward.

CHAPTER 2

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.

JSON

The organization of the rest of this chapter is oriented toward a description of the protocol used to control the hardware directly over a USB or RS-485 connection. However, you can set up a central server which is connected to all the readerboard signs and busylight indicators in a small area (via USB or RS-485). This allows clients to send commands to the server over the network for control of the readerboards.

The protocol between clients and servers uses messages which are simple JSON representations of the underlying hardware protocol, which is more convenient for network clients and servers to use. Since this mirrors the hardware protocol, those JSON objects will be listed below as well.

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

As a general rule, these messages are each terminated with a newline, and may not contain newlines otherwise.

Clients and servers MUST accept UTF-8 but SHOULD emit 7-bit ASCII with standard JSON escape codes for codepoints greater than 127.

Message Type

Every message has a field called `"m"` which contains the message type as a string. Each message contains additional fields as needed for its message type. Any expected fields which are missing are assumed to contain the "zero" value for their type (`false`, `0`, `""`, etc.) and unexpected fields which are presented are silently ignored.

Message Target Addresses

Every message also has a field called `"a"` which is a list of target readerboards to which the message applies. This is always a list even if it has a single target.

So, for example, the message containing `"a":[2]` would cause the server to transmit the corresponding command to the readerboard with address 2 directly connected to it via USB, or would transmit to the RS-485 network the command starting with the hex byte `92`.

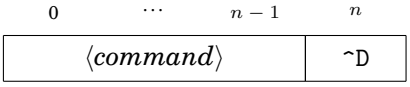
If multiple devices are targeted, say with a message containing the JSON field `"a":[2,5,37]` with address 15 (e.g., $\texttt{F}_{16}$) configured as the global address ad_G, it will transmit to the RS-485 network a command starting with the hex byte sequence `BF 03 02 05 25`.

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 $\texttt{04}_{16}$).



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.

Commands which contain arbitrary-size data fields, such as text strings or lists of LEDs to light in sequence, end such fields with a terminator byte. In most cases this may be either a dollar-sign ("$") character or the escape control character (hex byte $1B_{16}$), indicated in the protocol diagram above simply as "$". In cases where a dollar-sign could be part of the data, then only an escape character may be used to terminate the field, in which case the protocol description will show the terminator as "ESC".

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 adhere to a protocol that is also compatible with those devices.

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 or All Readerboards

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	$\langle ad \rangle$			

where $\langle ad \rangle$ 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. If the global address ad_G is given as the $\langle ad \rangle$ value, then all readerboards which have that set as their global address will obey the command.

Multiple Targets

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

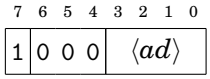
7	6	5	4	3	2	1	0
1	0	1	1	$\langle ad_G \rangle$			
0	0	$\langle n \rangle$					
0	0	$\langle ad_0 \rangle$					
⋮							
0	0	$\langle ad_{n-1} \rangle$					

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 $\langle ad_{n-1} \rangle$.

Note that device addresses are constrained to the range 0–15 if they are to be addressed in the command start byte. However, using the multiple target header, device addresses in the range 0–63 may be used.

All Off

As a special case, the single byte



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

The corresponding JSON message is `{"m":"off","a":[`$\langle ad \rangle$`]}`

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 $7E_{16}$ causes the next byte received to have its MSB set upon receipt.
- A hex byte $7F_{16}$ causes the next byte to be accepted without any further interpretation.

Thus, the byte $C4_{16}$ 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.

A few illustrative examples are shown in Table 2.1.

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 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

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

Input Sequence	Resulting Byte
00	00
7D	7D
7F 7E	7E
7F 7F	7F
7E 00	80
7E 01	81
7E 7D	FD
7E 7E	FE
7E 7F	FF

Table 2.1: Examples of RS-485 Escape Bytes.

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 2.2.

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.

*—Strobe Lights in Sequence

0	1	2	3	⋯	n	$n+1$
*	⟨led_0⟩	⟨led_1⟩	⟨led_2⟩	⋯	⟨led_{n-1}⟩	$

{"m":"strobe","a":[⟨ad_0⟩,...],"l":[⟨led_0⟩,⟨led_1⟩,...,⟨led_{n-1}⟩]}

Each ⟨*led*⟩ value is an ASCII character corresponding to a discrete LED as shown in Table 2.9. An ⟨*led*⟩ value of "_" means there is no LED illuminated at that point in the sequence.

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).

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

Message	Cmd	Description	Notes
strobe	*	Strobe LEDs in Sequence	[1]
query	?	Query discrete LED status	[1] [2] [4] [5]
flash	F	Flash LEDs in Sequence	[1]
light	L	Light one or more LEDs steady	[3]
light	S	Light one LED steady	[1]
off	X	All LEDs off	[1] [5]
	^D	Abort/terminate command	
scroll	<	Scroll text across display	
	=	Set operational parameters	[4]
move	@	Move current column cursor	
font	A	Select character font	
clear	C	Clear matrix display	
graph	H	Add histogram/bargraph data point	
bitmap	I	Draw bitmap graphic image	
color	K	Set current color	
query	Q	Query matrix display status	[2] [4] [6]
text	T	Display text on display	

[1] Busylight compatibile command

[2] Sends response (USB only)

[3] Busylight extension (not in original Busylight)

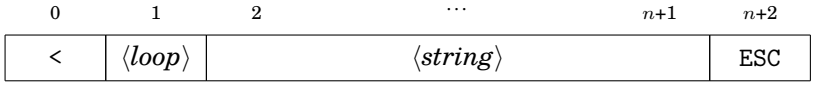
[4] USB only

[5] Only if field "status" is true

[6] Only if field "status" is omitted or false

Table 2.2: Summary of All Commands

<—Scroll Text Across Display



{"m":"scroll","a":[⟨ad_0⟩,...],"loop":⟨*bool*⟩,"t":⟨*string*⟩}

Displays the text ⟨*string*⟩ by scrolling it across the display from right to left. If ⟨*loop*⟩ 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 2.3.

Code	Hex	Description
^C⟨*pos*⟩	03⟨*pos*⟩	Move current column cursor to ⟨*pos*⟩
^D	04	Never allowed in strings (command terminator)
^F⟨*digit*⟩	06⟨*digit*⟩	Switch current font
^H⟨*pos*⟩	08⟨*pos*⟩	Move cursor left ⟨*pos*⟩ columns
^K⟨*rgb*⟩	0B⟨*rgb*⟩	Change color to ⟨*rgb*⟩
^L⟨*pos*⟩	0C⟨*pos*⟩	Move cursor right ⟨*pos*⟩ columns
^[	1B	Never allowed in strings (string terminator)

Table 2.3: Control Codes in String Values

=—Set Operational Parameters

0	1	2	3	4
=	⟨*ad*⟩	⟨*uspd*⟩	⟨*rspd*⟩	⟨ad_G⟩

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

If the ⟨*ad*⟩ parameter is "_" then the RS-485 interface is disabled entirely. Otherwise it is a value from 0–15 encoded as described in Table 2.5. This enables the RS-485 interface and assigns this sign's address to ⟨*ad*⟩.

The baud rate for the USB and RS-485 interfaces is set by the ⟨*uspd*⟩ and ⟨*rspd*⟩ values respectively. Each is encoded as per Table 2.4.

The ⟨ad_G⟩ 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 ⟨*ad*⟩. If you only have one readerboard or do no wish to assign a global address, just set ⟨ad_G⟩ to the same value as ⟨*ad*⟩.

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
?

{"m":"query","a":[⟨ad_0⟩,...],"status":true}

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:

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 2.4: Baud Rate Codes

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

Each $\langle led_x \rangle$ value is a single character which is "_" if the corresponding LED is off, or the LED's color code or position number if it is on. One such value is sent for each LED installed in the sign (typically eight for reader-boards), followed by a "$" to mark the end of the list.

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
$\langle run \rangle$	_

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

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

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 2.5: ASCII Encoded Integer Values (0–63)

In either case, ⟨*run*⟩ is the ASCII character "S" if the unit is stopped or "R" if it is currently running. If there is a defined sequence, ⟨*pos*⟩ indicates the 0-origin position within the sequence of the light currently being flashed or strobed, encoded as described in Table 2.5. The ⟨led_x⟩ values are as allowed for the F or * command that set the sequence. (Regardless of the actual F or * command parameters, the report will show symbolic color codes where possible, or numeric position codes otherwise.)

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
@	⟨*pos*⟩

{"m":"move","a":[⟨ad_0⟩,...],"pos":⟨*col*⟩}

Sets the column cursor position to the value indicated by ⟨*pos*⟩. See Table 2.5. (The JSON message uses integer values for ⟨*col*⟩ without the encoding used for ⟨*pos*⟩.)

A—Select Font

0	1
A	⟨*digit*⟩

{"m":"font","a":[⟨ad_0⟩,...],"idx":⟨*digit*⟩}

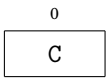
Sets the font to use for rendering text with the < and T commands. The font codes for ⟨*digit*⟩ are listed in Table 2.6. The full text fonts support the

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 2.6: Font Codes

printable ASCII characters plus a majority of the Unicode glyphs with codepoints less than 256. See Tables 2.7–2.8 for a complete font glyph listing with codepoint assignments.

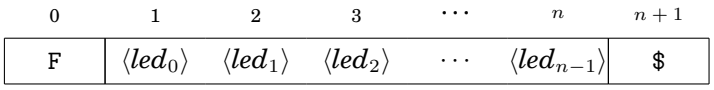
C—Clear Matrix Display



`{"m":"clear","a":[`⟨ad_0⟩`,...]}`

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

F—Flash Lights in Sequence



`{"m":"flash","a":[`⟨ad_0⟩`,...],"l":[`⟨led_0⟩`,`⟨led_1⟩`,`⟨led_2⟩`,...,`⟨led_{n-1}⟩`]}`

Each ⟨*led*⟩ value is an ASCII character corresponding to a discrete LED as shown in Table 2.9. 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.

An ⟨*led*⟩ value of "_" means there is to be no LED illuminated at the corresponding position in the sequence.

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.

If only one ⟨*led*⟩ 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`").

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	
00x				move	end		font		0*x*
01x	back			color	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	Đ	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 2.7: Font Table for Fonts #0 and #1

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	
00x				move	end		font		0*x*
01x	back			color	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	AM	PM	°F	°C					9*x*
23x	NM	WC	1Q	WG	FM	WG	3Q	WC	
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 2.8: Font Table for Font #2

Code*	Light	Color
W	L_0	white
B	L_1	blue
b	L_2	blue
R	L_3	red
r	L_4	red
Y	L_5	yellow
y	L_6	yellow
G	L_7	green
_	—	(no LED/off)
0–9	L_0–L_9	LED installed at physical position 0–9

*If a sign is built with different colors in these positions, the letter codes for those LEDs will match the custom color arrangement for that sign. (Custom firmware modification required.)

Table 2.9: Discrete LED Codes and Colors

H—Draw Bar Graph Data Point

0	1
H	⟨*n*⟩

{"m":"graph","a":[⟨*ad*$_0$⟩,...],"v":⟨*n*⟩}

0	1	2	3	4	5	6	7
H	K	⟨*rgb*$_0$⟩	⟨*rgb*$_1$⟩	⟨*rgb*$_2$⟩	⟨*rgb*$_3$⟩	⟨*rgb*$_4$⟩	⟨*rgb*$_5$⟩
⟨*rgb*$_6$⟩	⟨*rgb*$_7$⟩						

{"m":"graph","a":[⟨*ad*$_0$⟩,...],"v":⟨*n*⟩,"colors":[⟨*rgb*$_0$⟩,⟨*rgb*$_1$⟩,
⟨*rgb*$_2$⟩,⟨*rgb*$_3$⟩,⟨*rgb*$_4$⟩,⟨*rgb*$_5$⟩,⟨*rgb*$_6$⟩,⟨*rgb*$_7$⟩]}

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.

In the simplest form, it takes a single byte parameter. The value ⟨*n*⟩ 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 ⟨*n*⟩ 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.

In the second form, it takes eight ⟨*rgb*⟩ values (q.v.) which give the color to light up each row in the bar-graph column.

In JSON, the values of ⟨*n*⟩ and ⟨*rgb*$_n$⟩ are sent as integer values.

I—Draw Bitmap Image

0	1	2	3	4	5	6	7	
I	⟨*merge*⟩	⟨*pos*⟩	⟨*trans*⟩	⟨*R coldata*$_0$⟩		⟨*R coldata*$_1$⟩		···
⟨*R coldata*$_{n-1}$⟩	$	⟨*G coldata*⟩...		$	⟨*B coldata*⟩...		$	

{"m":"bitmap","a":[⟨*ad*$_0$⟩,...],"merge":⟨*bool*⟩,"pos":⟨*col*⟩,
"trans":⟨*trans*⟩,"image":[[⟨*R coldata*$_0$⟩,⟨*R coldata*$_1$⟩,...,
⟨*R coldata*$_{n-1}$⟩],[⟨*G coldata*$_0$⟩,⟨*G coldata*$_1$⟩,...,
⟨*G coldata*$_{n-1}$⟩],[⟨*B coldata*$_0$⟩,⟨*B coldata*$_1$⟩,...,⟨*B coldata*$_{n-1}$⟩]]}

Draws an arbitrary bitmap image onto the matrix display starting at column ⟨*pos*⟩, encoded as per Table 2.5. A ⟨*pos*⟩ value of "~" represents the current column cursor position. (In JSON, since ⟨*col*⟩ is an integer value, this is represented by a value of −1 for ⟨*col*⟩.)

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. (In JSON, these are simply plain decimal integer values.)

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 2.10: Transition Effect Codes

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

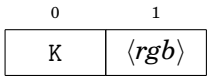
The first set of column data provide the bits for the red color plane. After the terminating byte, two more sets of column data are sent, each with identical format to the first, to provide the bits for the green and blue color planes, respectively.

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 2.10.

K—Set Current Color



{"m":"color","a":[⟨ad_0⟩,...],"color":⟨*rgb*⟩}

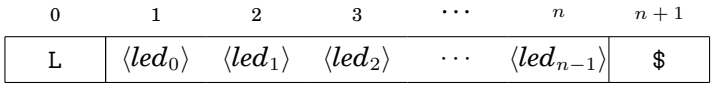
This command sets the color which all future commands will use by default (although some of them allow for the specification of colors directly on an *ad hoc* basis, which does not affect the current default color).

The ⟨*rgb*⟩ value is a digit in the range 0–7, with the meanings defined in Table 2.11. Note that anything drawn in black simply turns off the corresponding pixels.

Code	Color
0	black
1	red
2	green
3	yellow
4	blue
5	magenta
6	cyan
7	white

Table 2.11: Color codes for ⟨*rgb*⟩ parameters

L—Light Multiple LEDs

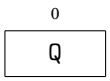


{"m":"light","a":[⟨ad_0⟩,...],"l":[⟨led_0⟩,⟨led_1⟩,⟨led_2⟩,...,⟨led_{n-1}⟩]}

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 2.9. 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



{"m":"query","a":[⟨ad_0⟩,...],"status":false}

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	⟨*model*⟩	=	⟨*ad*⟩	⟨*uspd*⟩	⟨*rspd*⟩	⟨ad_G⟩	$	V
⟨*hwversion*⟩		$	R	⟨*romversion*⟩		$	S	⟨*serial*⟩
$	M	⟨R *coldata*$_0$⟩	⟨R *coldata*$_1$⟩	⋯		⟨R *coldata*$_{63}$⟩		
$	⟨G *coldata*$_0$⟩	⟨G *coldata*$_1$⟩	⋯	⟨G *coldata*$_{63}$⟩				$
⟨B *coldata*$_0$⟩	⟨B *coldata*$_1$⟩	⋯	⟨B *coldata*$_{63}$⟩				\n	

The ⟨*model*⟩ 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), "M" for the current 64×8 matrix display hardware, or "C" for 64×8 RGB color display boards.

⟨*hwversion*⟩ and ⟨*romversion*⟩ 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 ⟨*serial*⟩ 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 ⟨*coldata*⟩ bytes are sent just as with the I command, as hexadecimal values indicating the LEDs lit on the matrix display, with ⟨$coldata_0$⟩ being the leftmost column of the display and ⟨$coldata_{63}$⟩ being the rightmost. In each column, the least significant bit indicates the LED on the top row.

The ⟨*ad*⟩, ⟨*uspd*⟩, and ⟨*rspd*⟩ 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".

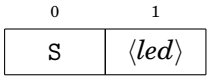
Busylight Device Query Response

If a dedicated (non-readerboard) busylight device is sent the Q command, it responds with the following information.

0	1	2	3	4	5	6	7	8
Q	B	=	⟨*ad*⟩	⟨*uspd*⟩	⟨*rspd*⟩	⟨ad_G⟩	$	V
⟨*hwversion*⟩		$	R	⟨*romversion*⟩		$	S	⟨*serial*⟩
$	\n							

[4] See semver.org.

S—Light Single LED

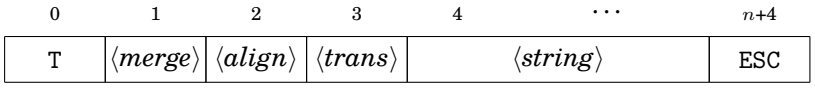


{"m":"light","a":[⟨ad_0⟩,...],"l":[⟨*led*⟩]}

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 2.9. 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



{"m":"text","a":[⟨ad_0⟩,...],"merge":⟨*bool*⟩,
"align":⟨*align*⟩,"trans":⟨*trans*⟩,"t":⟨*string*⟩}

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.

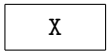
The ⟨*align*⟩ code indicates how the text is aligned on the display (TBD).

The ⟨*trans*⟩ code specifies the transition effect to be used to add this string to the display as shown in Table 2.10.

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 2.3.

X—Turn off Discrete LEDs



{"m":"off","status":true}

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

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

CHAPTER 3

PINOUTS

No cord or cable can draw so forcibly, or bind so fast, as love can do with a single thread.

—Robert Burton

THIS CHAPTER DESCRIBES each connector used by the readerboard devices and what signals are present on which pins.

Connectors for Version 3.1.0 Boards

Arduino Bus Connectors [3.1.0 J0]

Two parallel rows of stacking pins, for a total of 50 pins, extend out of the back of the readerboard PCB. An Arduino Mega 2560 or Due microcontroller board is connected to these pins.

The USB connector on the Arduino board may be used to directly connect the readerboard to a PC for initial configuration and/or to use it as a directly-connected singleton device. (If there are multiple devices in use, it may be preferable to switch to the RS-485 network after initial configuration of the device.)

Power / RS-485 (8-pin terminal) [3.1.0 J1]

The single external connector on version 2.1.0 boards is an 8-pin screw terminal block. This accepts a +9 V DC power input and ground on pins 3 and 4 respectively, which powers the entire board and the attached Arduino

controller. If RS-485 communications will be used, the incoming signal is received on pins 1, 2, and 5 while the outgoing signal is on pins 6, 7, and 8. (In actuality, the "input" and "output" sense is arbitrary and either set of A and B signals may be used as input or output.)

If this device is the last in the RS-485 network chain, insert a 120Ω resistor between the A and B terminals that would have been used as the output if there had been another device connected there. This properly terminates the RS-485 network at that point.

1	A (Data in +)	5	GND
2	B (Data in –)	6	GND
3	+9 V DC in	7	A (Data out +)
4	GND	8	B (Data out –)

Connectors for Legacy Boards

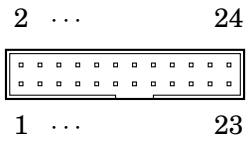
Connectors for Version 2.1.0 Boards

Power / RS-485 (8-pin terminal) [2.1.0 J0]

This is wired the same as J1 on the 3.1.0 boards.

Earlier Boards

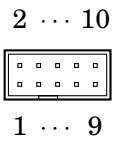
Matrix Control (24-pin ribbon cable) [1.0.1 J1, 2.0.0 J1]



For readerboards before version 2.1.0, this 24-position ribbon cable carries signals to directly drive the 64×8 LED matrix. The other end of this cable mates with J0 on the shield board. The pinout of its IDC header is:

1	D6	7	RCLK	13	+5 V DC	19	Gnd
2	D5	8	D2	14	+5 V DC	20	Gnd
3	D7	9	$\overline{\text{G}}$	15	+5 V DC	21	R2
4	D4	10	D1	16	+5 V DC	22	REN
5	SRCLK	11	$\overline{\text{SRCLR}}$	17	Gnd	23	R1
6	D3	12	D0	18	Gnd	24	R0

Discrete LEDs (10-pin ribbon cable) [1.0.1 J2, 2.0.0 J2]



For readerboards before version 2.1.0, this 10-position ribbon cable carries signals to directly drive the 64×8 LED matrix. The other end of the cable mates with J1 on the shield board. The pinout of its IDC header is:

1	GND	6	L6
2	GND	7	L2
3	L0	8	L5
4	L7	9	L3
5	L1	10	L4

Board Power (3-pin screw terminal) [1.0.1 J0, 2.0.0 J0]

This 3-position screw terminal block provides power to the readerboard. Note that the +5 V supply is also connected to pins 13–16, and Ground to pins 17–20 of J1, the only power required here is the +9 V input that drives the LEDs themselves.

1	+5 V DC in
2	GND
3	+9 V DC in

Shield Power (5-pin screw terminal) [Shield J2]

This 5-position screw terminal block accepts incoming +9 V DC power and ground on pins 4 and 3 respectively. It then provides +5 V DC, +9 V DC, and ground outputs on pins 1, 2, and 5 respectively to supply power to the main display board.

1	+5 V DC out
2	GND
3	GND
4	+9 V DC in
5	+9 V DC out

Shield RS-485 (6-pin screw terminal) [Shield J4]

This 6-position screw terminal block accepts incoming RS-485 signals A, B, and ground on pins 2, 1, and 3 respectively, and outputs the network signals A, B, and ground on pins 6, 5, and 4 respectively, to go on to the next device in the chain. If this is the last device, then nothing should be

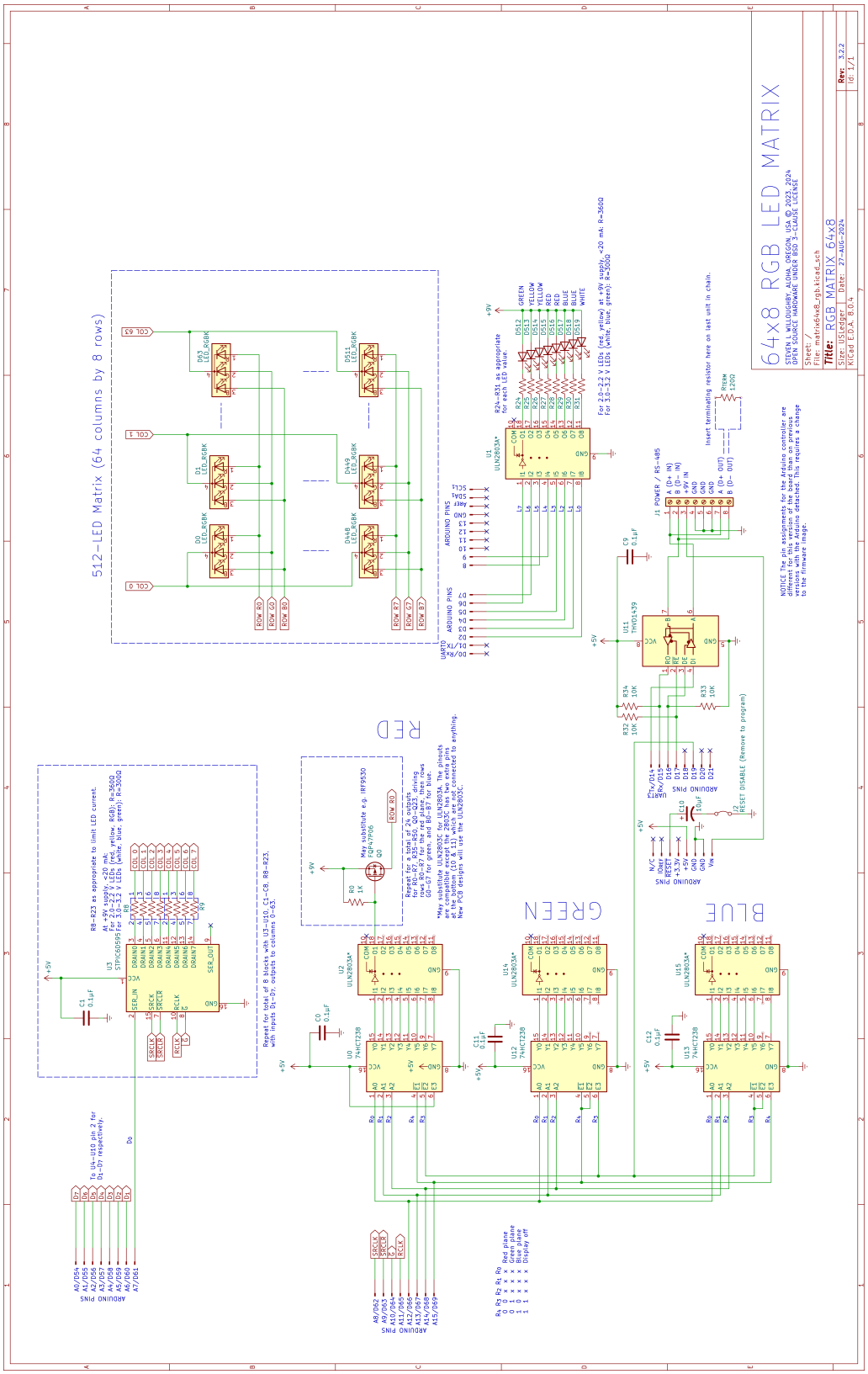
connected to pins 5 and 6. Instead, install jumper J5 which connects a 120Ω resistor across those terminals to terminate the network at that point.

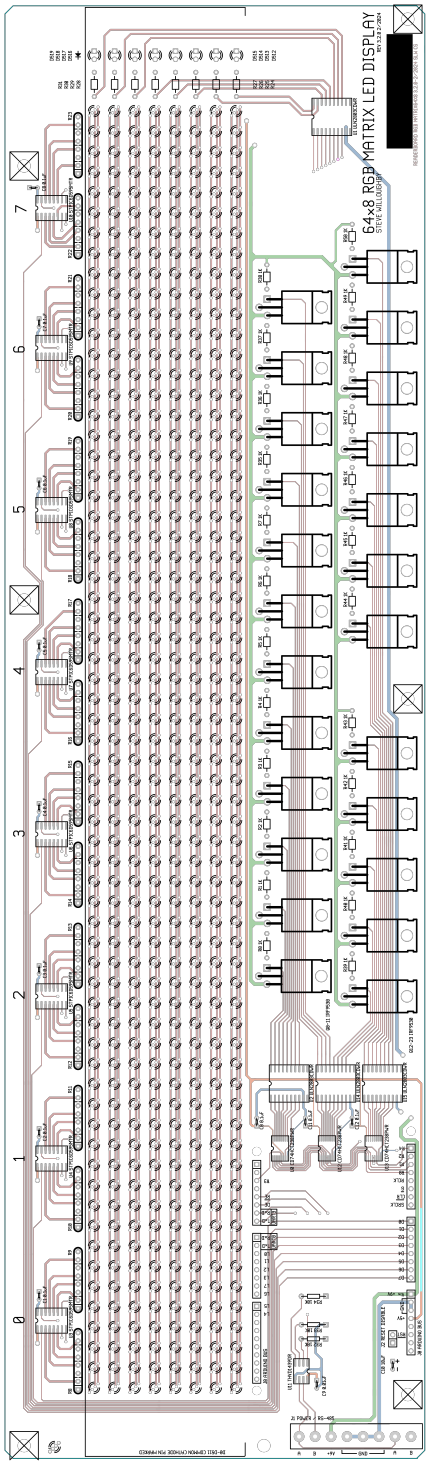
1	B (Data In –)
2	A (Data In +)
3	GND
4	GND
5	B (Data Out –)
6	A (Data Out +)

Chapter 4

Schematics and Diagrams

The following page includes the schematics for the current 3.2.1 version of the readerboard PCB and the board assembly diagram, showing where the components are placed.


64x8 RGB LED MATRIX
STEVEN L. WILLOUGHBY, ALOHA, OREGON, USA © 2023, 2024
OPEN SOURCE HARDWARE UNDER BSD 3-CLAUSE LICENSE
Title: RGB MATRIX 64x8
Rev: 3.2.2
512-LED Matrix (64 columns by 8 rows)
RED
GREEN
BLUE
NOTICE: The pin assignments for the Arduino controller are different for the Nano and Uno. There are separate firmware versions with the Arduino detached. This requires a change to the firmware image.
Repeat for total of 8 blocks with U3-U10, C1-C8, R8-R23.
Repeat for a total of 24 outputs
Repeat for total of 24 ULN2803A driving
rows R0-R7 for the red plane, then rows
R8-R15 for green, and R16-R23 for blue.
*May substitute e.g. IRF9530
Insert terminating resistor here on last unit in chain.
ARDUINO PINS
R24-R31 as appropriate for each LED value.
For 2.0-2.2 V LEDs (red, yellow) at +5V supply, <20 mA: R=360Ω
For 3.0-3.2 V LEDs (white, blue, green): R=300Ω
R8-R23 as appropriate to limit LED current.
At +5V supply, <20 mA:
For 2.0-2.2 V LEDs (red, yellow, RGB): R=360Ω
For 3.0-3.2 V LEDs (white, blue, green): R=300Ω
To U4-U10 pin 2 for D4-D7 respectively.
Rx Ry Rz Ro
0 0 0 x x x x x Red plane
0 1 0 x x x x x Green plane
1 0 0 x x x x x Blue plane
1 1 1 x x x x x Display off


64x8 RGB MATRIX LED DISPLAY