Busylight
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 busylight hardware version 1.0.2, and firmware version 2.0.0.

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 busylight device is very simple. Commands are expressed largely in plain ASCII characters and are executed immediately as they are received.[1]

USB vs. RS-485

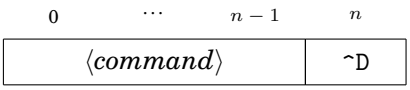
The protocol used to send commands to the busylight is different depending on whether the host is sending directly to a single device over a USB cable, or to (possibly) multiple devices over an RS-485 bus network.

This was designed for the Adruino Nano controller, which only has a single USART, so the busylight will differentiate between USB and RS-485 commands from their respective formats. On other units with multiple serial ports, one is dedidated to the RS-485 network.

USB

A busylight 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 04_{16}).

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



If there is an error parsing or executing a command, the busylight 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.

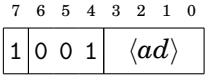
RS-485

Commands sent over RS-485 are intended to target one or more of a set of connected busylight units 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 busylight units which should obey the command.

Single Target or All Devices

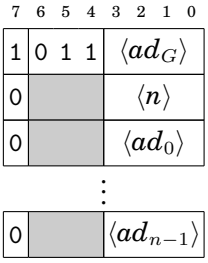
To send a command to a single unit, begin with a single byte encoded as:



where $\langle ad \rangle$ is the unit'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 busylight units which have that set as their global address will obey the command.

Multiple Targets

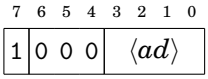
Alternatively, a command may be targetted to multiple units by starting the command with a multi-byte code:



where $\langle ad_G \rangle$ is the "global" device address which signals busylight units 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 Off

As a special case, the single byte



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

No other command bytes need to follow; this byte is sufficient to turn off the unit(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:

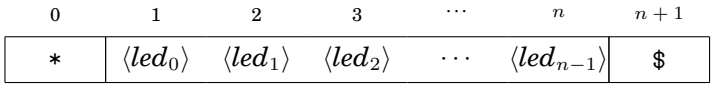
- A hex byte $\texttt{7E}_{16}$ causes the next byte received to have its MSB set upon receipt.
- A hex byte $\texttt{7F}_{16}$ causes the next byte to be accepted without any further interpretation.

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 busylight 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

*—Strobe Lights in Sequence



Each $\langle led \rangle$ value is an ASCII character corresponding to a discrete LED as shown in Table 1.4. An $\langle led \rangle$ 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).

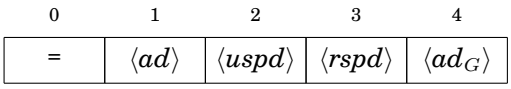
Command	Description	Notes
?	Query discrete LED status	[1] [2]
=	Set operational parameters	[2]
*	Strobe LEDs in Sequence	
F	Flash LEDs in Sequence	
L	Light one or more LEDs steady	
Q	Query device status	[1] [2]
S	Light one LED steady	
X	All LEDs off	

[1] Sends response

[2] USB only

Table 1.1: Summary of All Commands

=—Set Operational Parameters



This command sets a few operational parameters for the unit. 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 1.3. This enables the RS-485 interface and assigns this unit's address to $\langle ad \rangle$.

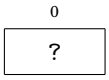
The baud rate for the USB and RS-485 interfaces is set by the $\langle uspd \rangle$ and $\langle rspd \rangle$ values respectively. Each is encoded as per Table 1.2.

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 busylight units should pay attention to the start of the command because it might target them either as part of a list of specific busylight units or because the command is intended for all busylight units at once. This is encoded in the same way as $\langle ad \rangle$. If you only have one busylight or do no wish to assign a global address, just set $\langle ad_G \rangle$ to the same value as $\langle ad \rangle$.

This command may only be sent over the USB port.

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

?—Query Discrete LED Status



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 1.2: Baud Rate Codes

This command causes the unit 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	$\langle led_0 \rangle$	$\langle led_1 \rangle$	$\langle led_2 \rangle$	$\langle led_3 \rangle$	$\langle led_4 \rangle$	$\langle led_5 \rangle$	$\cdots$	$\langle led_{n-1} \rangle$
$	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. a "?" is sent if the value set for the LED is invalid. One such value is sent for each LED installed in the unit (typically seven for busylight units), 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:

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 1.3: ASCII Encoded Integer Values (0–63)

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

In either case, $\langle run \rangle$ is the ASCII character "S" if the unit is stopped or "R" 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 1.3. The $\langle led_x \rangle$ 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.

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 $\langle led \rangle$ value is an ASCII character corresponding to a discrete LED as shown in Table 1.4. 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 $\langle led \rangle$ value of "_" means there is to be no LED illuminated at the corresponding position in the sequence.

Up to 64 $\langle led \rangle$ codes may be listed. The unit will cycle through the sequence, lighting each specified LED briefly before moving on to the next

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

*If a unit is built with different colors in these positions, the letter codes for those LEDs will match the custom color arrangement for that unit.

Table 1.4: Discrete LED Codes and Colors

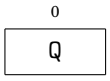
one. The sequence is repeated forever in a loop until an L, S or X command is received.

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.

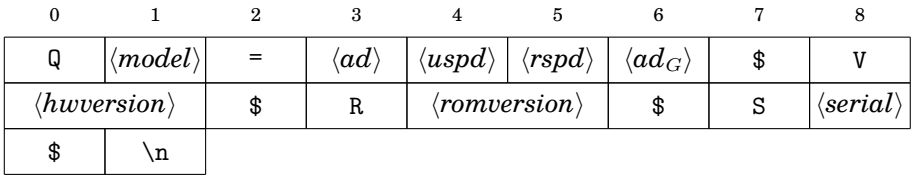
The sequence is terminated by either a dollar-sign ("$") character or the escape control character (hex byte 1B), indicated in the protocol diagram above simply as "$".

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

Q—Query Device Status



This command causes the unit 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:



The ⟨*model*⟩ field is "B" for the busylight hardware. Other codes designate other compatible devices.

⟨*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

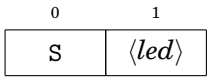
standard 2.0.0.[2] Each is terminated by a dollar-sign (`$`) character (and thus those strings may not contain dollar signs).

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 B000–B299 are reserved for the original author's use. This string is also terminated with a dollar sign.

The ⟨*ad*⟩, ⟨*uspd*⟩, and ⟨*rspd*⟩ values are as last set by the `=` command (or the factory defaults if they were never changed). If the serial device is disabled, "`_`" is sent instead of the baud rate code. Likewise, "`_`" is sent for the address of a device for which no address is set. "`*`" is sent for an address that is somehow invalid.

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

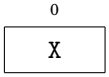
`S`—Light Single LED



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 1.4. 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`").

`X`—Turn off Discrete LEDs



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

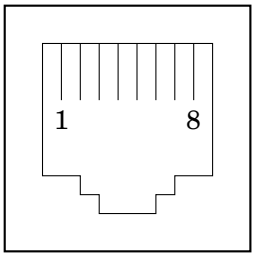
This command may be given in upper- or lower-case ("`x`" or "`X`").

[2] See semver.org.

CHAPTER 2

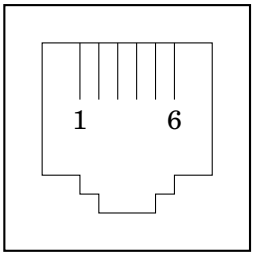
Connector Pinouts

Full-Duplex RS-485 Connection (8p8c Female Modular Jacks)



1 Return Data (+) (W/Or)
2 Return Data (–) (Or)
3 Cable Check (W/Gr)
4 Data (–) (Bl)
5 Data (+) (W/Bl)
6 Cable Check (Gr)
7 Data Ground (W/Br)
8 Return Data Ground (Br)

Half-Duplex RS-485 Connection (6p6c Female Modular Jacks)



1 Cable Check (W/Gn)
2 Data Ground (W/Br)
3 Data (–) (Bl)
4 Data (+) (W/Bl)
5 Data Ground (Br)
6 Cable Check (Gn)