NAME

lumosctl − Manual control for Lumos SSR controller hardware

SYNOPSIS

lumosctl [**−dhkPRSvwXz**] [**−a** *addr*] [**−A** *addr*] [**−b** *speed*] [**−B** *speed*] [**−c** *file*] [**−C** *file*] [**−D** *sens*] [**−E** *sens*] [**−p** *port*] [**−P** *phase*] [**−s** *file*] [**−t** *s*[**orw**]**:***init***:***seq***:***term*] *channel-outputs*...

Where *channel-outputs* may be any combination of:

channel[**@***level*[**,**...]]
*channel***d**[**:***steps*[**:***time*]]
*channel***u**[**:***steps*[**:***time*]]
x*id*
p*time*

DESCRIPTION

This command allows you to directly manipulate the state of a supported Lumos SSR controller unit, including administration functions such as changing the unit's address, phase offset, etc.

Other software such as **lumos**(1) or—providing appropriate drivers are installed—popular third-party programs such as Vixen are more appropriate for performing ("playing") sequences of light patterns on these boards. By contrast, **lumosctl** is more suited to setting up and configuring the boards (although some basic real-time control of channel outputs is possible using **lumosctl**).

In the absence of any command-line options to the contrary, the normal operation of **lumosctl** is to make a number of channel output level changes as determined by the non-option arguments which are of the form:

channel

or

*channel***@***level*[**,**...]

or

*channel***{u|d}**[**:***steps*[**:***time*]]

In the first case, a channel number by itself means to turn on that channel to full brightness. In the second case, by specifying a level value (a number from 0 to 255, inclusive), that channel's output is dimmed to the given level. Level 255 is the same as turning on to full brightness; level 0 is the same as turning it fully off.

In the third case, the dimmer level is ramped up smoothly from its current value to full brightness ("**u**"), or down smoothly until fully off ("**d**"). Optionally you may specify the number of dimmer level increments to increase or decrease at each change (1−128, default is 1); additionally, you may specify the amount of time to wait between each step, in units of 1/120 second (1−128, default is 1). As a convenience, this may be expressed as a real number of seconds followed by the letter "**s**". Thus, the argument **13@127** sets channel 13 to half brightness. If this were followed by the argument **13u** then channel 13 would be smoothly increased in brightness from there to full brightness (which is another 128 levels to take it from 127 to 255), by incrementing it one level every 1/120th of a second, reaching a full brightness level 128/120 seconds later (1.0666 seconds). If the argument **13d:10:2** were given, then channel 13 would drop to being fully off, going in steps of 10 levels at a time, 1/60th of a second between each step. Finally, an argument **10u:5:0.25s** fades channel 10 up from its current value to full brightness by incrementing its value by 5 every quarter-second.

Bulk updating of channels is also supported. If multiple values are listed for a channel, such as:

10@0,0,255,255,127,40,30,20,10

Then the channel named (10 in this example) is assigned the first value (0), and the subsequent values are assigned to the immediately following channels (so channel 11 is set to 0, 12 is set to 255, and so forth).

Note that if a controller implements a lower resolution dimmer than 256 levels (e.g., 128 or 64 levels), the same number scale is used (0−255), with the dimmer output scaled accordingly. For example, if the hardware implements only 128 dimmer levels, then levels 0 and 1 are fully off, 2 and 3 are the next level up, and so on, with values 252 and 253 being the penultimate dimmer level, and 254 and 255 being fully on.

In addition to the channel-setting argument described above, an argument of the form **x***i* causes stored sequence *i* to be executed. Note that this is run in the "background"—any subsequent channel-setting

arguments will be acted upon *while* the sequence is running. If a sequence was already running, it is stopped first. As a special case, **x0** stops the currently-running sequence but does not start a new one.

A pause in the execution of the arguments may be effected by adding an argument of the form **p***t* which makes **lumosctl** pause for *t* seconds before continuing on to the next argument. The *t* value need not be an integer.

A number of options are provided as described below. These command the SSR controller to perform certain administrative functions or configuration changes.

When giving multiple types of commands in one invocation of this program, they will be carried out in the following order:

1. Address Change
2. Kill all channels
3. Other configuration changes
4. Disable privileges
5. Channel(s) off/on/dim/etc.
6. Shutdown

OPTIONS

Each of the following options may be specified by either a long option (like "**−−verbose**") or a shorter option letter (like "**−v**"). If an option takes a parameter, it may follow the option as "**−a12**", "**−a 12**", "**−−address 12**", or "**−−address=12**".

Long option names may be abbreviated to any unambiguous initial substring.

−−address=*addr* (**−a** *addr*) Specifies the address of the target controller unit. The *addr* value is an integer from 0 to 15, inclusive. It defaults to 0.

−−clear−sequences (**−S**) Delete all stored sequences from the device's memory.

−−disable−sensor=*s* (**−D** *s*) Disable inputs from the sensor(s) specified as the *s* parameter (which are given as a set of one or more letters, e.g., **−−disable−sensor=ab**). The Lumos board will act as though those sensors were inactive regardless of their actual inputs. The special character "*****" appearing in *s* means to disable all sensors.

−−drop−privileged−mode (**−d**) If the Lumos device is in privileged command mode (for configuraiton of the device), this will cancel that mode. Further privileged commands will not be recognized on that device.

−−dump−configuration= *file* (**−C** *file*) Dump the device configuration into the named *file*. See below for a description of the configuration file format.

−−enable−sensor=*s* (**−E** *s*) Enable inputs from the sensor(s) specified as the *s* parameter. See **−−disable−sensor**.

−−help (**−h**) Prints a summary of these options and exits.

−−kill−all (**−k**) Turn off all output channels at once.

−−load−configuration= *file* (**−c** *file*) Load the device configuration from the named *file* and program that into the device.

−−load−sequence= *file* (**−s** *file*) Load one or more sequences from the specified source *file* (see below for sequence source code syntax) and program them into the device. If another sequence already exists with the same number, it replaces the old one; however, beware that the controller device does not optimize memory storage, so eventually stored sequences may become fragmented, resulting in running out of storage space for them. To avoid this, it is best to clear all sequences using the **−−clear−sequences** option, then load all the sequences you want on the device at once.

−−port=*port*
(**−p** *port*) Specify the serial port to use when communicating with the controller unit. This may be a simple integer value (0 for the first serial port on the system, 1 for the next one, etc.) or the actual device name on your system (such as "**COM1**" or "**/dev/ttys0**").

−−probe
(**−P**) Search for, and report on, all Lumos controllers attached to the serial network. If the **−−report** option is also specified, this provides that level of output for every attached device; otherwise, it only lists device models and addresses.

−−report
(**−R**) Report on the current device status to standard output in human-readable form.

−−sensor=*s*[**orw**]**:***init***:***seq***:***term*
(**−t** *s*[**orw**]**:***init***:***seq***:***term*) Define an action to be taken when a sensor is triggered. When the sensor is activated, the sequence *init* is run, followed by the sequence *seq* and then finally the sequence *term* when the sensor event is over. The sensor assigned this action is given as the parameter *s* and is one of the letters **A**, **B**, **C**, or **D**. This may be followed by the following option letters as needed:

- **o** Trigger once: play sequence *seq* only one time. The action will not be taken again until the sensor input transitions to inactive and then asserts itself as active again. This is the default action.
- **r** Repeat mode: play sequence *seq* indefinitely until explicitly told to stop (by an overt stop command such as an **x0** argument, or another sequence being triggered manually or by sensor action).
- **w** Trigger while active: play sequence *seq* repeatedly as long as the sensor remains active. When the sensor input transitions to inactive again, terminate the action.

If 0 is specified for any of the sequence numbers, that means no sequence is called for that part of the trigger action.

−−set−address=*addr*
(**−A** *addr*) Change the device address to *addr*. This must be an integer in the range [0, 15]; however, the address 15 is also a limited "broadcast" address, so ideally the address of a Lumos controller should be in the range [0, 14] unless you know for sure that the board can be loacated at address 15 without upsetting your purposes for it.

−−set−baud−rate=*speed*
(**−B** *rate*) Set a new baud rate for the device to start using from now on.

−−set−phase=*offset*
(**−P** *offset*) Set the phase *offset* in the device to the specified value. This must be an integer in the range [0, 511]. *This is an advanced setting which affects the ability of the AC relay boards to function properly. Do not change this setting unless you know exactly what you are doing.*

−−sleep
(**−z**) Tell the unit to go to sleep (this instructs the board to turn off a power supply which it is controlling, if any, but has no other effect).

−−shutdown
(**−X**) Command the unit to shut down completely. It will be unresponsive until power cycled or the reset button is pressed to reboot the controller.

−−speed=*rate*
(**−b** *rate*) Set the serial port to the given baud *rate*. [Default is 19200 baud.]

−−wake
(**−w**) Tell the unit to start the attached power supply from sleep mode. command is given at a future time.

−−verbose
(**−v**) Output messages to the standard output. Additional **−−verbose** options increases verbosity. High levels of verbosity include a dump of every bit sent or received on the serial network.

CONFIGURATION FILE FORMAT

The files read and written by the **--dump-configuration** and **--load-configuration** options uses a fairly standard configuration file format similar to the "ini" files used by early versions of Microsoft Windows and other systems. Full details of this format see **http://docs.python.org/library/configparser.html**, but the highlights include:

1. One data value per line (long lines may be continued by indentation ala RFC 822 headers).
2. Each line consists of the name of a data value, either an equals sign or a colon, and the value itself.
3. A syntax **%(name)s** can be used to substitute values into other values. Literal percent signs in values are simply doubled ("**%%**").

All confugration data are contained in a stanza called "**[lumos_device_settings]**". The values are:

baud=*n*

The configured serial I/O speed of the device. Supported values include 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, and 250000. Speeds slower than 9600 baud are not recommended. [Default is 19200.]

dmxchannel=*n*

If this field exists, the Lumos board is to run in DMX512 mode, with its channel #0 appearing at DMX512 slot #*n*, where *n* is an integer in the range [1, 512]. If this field is not present, the Lumos board will not be configured to recognize DMX512 packets at all.

phase=offset

The AC waveform phase offset for the unit. This should only be changed if needed due to some anomaly with the zero-crossing detector which throws off the unit's timing. This is an integer in the range [0, 511]. [Default is 2.]

resolution={**high**|**low**}

If "**high**", channel output levels in the full range [0, 255] are recognized. If "**low**", then only 128 levels are used, although the data values referenced by the user remain in the range [0, 255]. However, only 7 bits of dimmer data are transmitted to the hardware unit, saving a small amount of transmission data. In this mode, values 0 and 1 are the fully-off value, 2 and 3 are the next step up from that, 254 and 255 are fully on, and so forth.

sensors=*list*

The value is a list of single letters in the range [A, D]. Each letter appearing in this list indicates that the corresponding sensor input should be enabled in the hardware. You must ensure that the hardware is really configured that way.

Sensor Configuration

For each sensor listed in the **sensors** field, a corresponding stanza called "**[lumos_device_sensor_***x***]**" appears, where *x* is the name of the sensor ("**A**", "**B**", "**C**", or "**D**"), with the following fields:

enabled=*bool*

If "**yes**", the sensor input is set to be monitored. If "**no**", it is ignored. [Default is "**yes**".]

mode={**once**|**repeat**|**while**}

Define the operating mode of the sensor trigger: play once per trigger, repeat forever until another trigger (or explicit command to stop), or play as long as sensor remains active. [Default is **once**.]

setup=*id*

Sequence *id* number to be played initially when the sensor becomes active

sequence=*id*

Sequence *id* number to be played as the main (possibly repeated) aciton for the sensor.

terminate=*id*

Sequence *id* number to be played when the action stops. Note that the main sequence might not have played to completion.

SEQUENCE SOURCE SYNTAX

Each source file given to **−−load−sequence** contains one or more sequence definitions as described here. The formal syntax definition for the sequence language is:

<sequence> ::= SEQUENCE <id> [’(’ <arg-list> ’)’] <block>
<statement-list> ::= <statement> <newline> | <statement-list> <statement> <newline>
<statement> ::= [<symbol> ’:’] [<command>]
<command> ::= BLACKOUT
| CHANNEL <chan> <state>
| RAMP <direction> <chan> [BY <steps> [PER <time>]]
| CALL <id> [’(’ <value-list> ’)’]
| EXECUTE <id> [’(’ <value-list> ’)’]
| WAIT <time>
| FOR <symbol> ’=’ <value> TO <value> [BY <value>] <block>
| IF <condition> <block>
| UNLESS <condition> <block>
| REPEAT [<value> [AS <symbol>]] <block>
| BREAK <symbol>
| CONTINUE <symbol>
| <symbol> ’=’ <value>
| SLEEP
| WAKE
| SUSPEND [WITH UPDATE]
| RESUME [WITH FADE]
| EXIT
<block> ::= ’:’ <newline> <statement-list> END
<id> ::= 0 | 1 | ... | 127
<chan> ::= <value> (allowed range 0...63)
<dimmer-value> ::= 0 | 1 | ... | 255
<percentage> ::= 0 | 1 | ... | 100
<state> ::= ON | OFF | <dimmer-value> | <percentage> ’%’
<direction> ::= UP | DOWN
<steps> ::= 1 | 2 | ... | 128
<time> :: = <cycles> | <real> SEC[ONDS]
<cycles> ::= 1 | 2 | ... | 128
<real> ::= <digits> [’.’ [<digits>]] | [<digits>] ’.’ <digits>
<condition> ::= SUSPENDED
| SENSOR <sens-list>
| <value> <comparison> <value>
<comparison> ::= ’<’ | ’>’ | ’==’ | ’!=’ | ’<=’, ’>=’
<arg-list> ::= <symbol> | <arg-list> ’,’ <symbol>
<value-list> ::= <value> | <value-list> ’,’ <value>
<value> ::= <int>
| <symbol>
| <value> ’*’ <value>
| <value> ’+’ <value>
| <value> ’-’ <value>
| <value> ’/’ <value>
| ’(’ <value> ’)’
<int> ::= [’-’ | ’+’] <digits>
<symbol> ::= <letter> | <symbol> <digit> | <symbol> <letter>
<digits> ::= <digit> | <digits> <digit>
<digit> ::= ’0’ | ’1’ | ... | ’9’
<letter> ::= ’_’ | ’A’ | ’B’ | ... | ’Z’
<sens-list> ::= <sens> | <sens-list> OR <sens>

<sens> ::= 'A' | 'B' | 'C' | 'D'
<newline> ::= '\n'

While the language keywords are shown here in upper-case letters, in fact all symbols are interpreted irrespective of case throughout the sequence source file.

Expressions

Simple math expressions are supported, including addition, subtraction, multiplication, and division, but the values supported by the hardware runtime system are exclusively 8 bit unsigned integers. Division is not supported at all in the hardware. Math involving compile-time values is not so limited, but if the expression evaluation remaining to be done at runtime it too complex, the compiler will issue a fatal error.

Flow Control Constructs

Each of these constructs operates on a *block* of code. A block is a sequence of one or more lines of code (each terminated by a newline), ending with an **END** statement.

IF SENSOR *s1* [**OR** *s2*...]**:** ... **END**

Execute the statements in the block if any of the sensors are currently active.

IF SUSPENDED: ... **END**

Execute the block if the unit is currently in a suspended state.

IF *condition***:** ... **END**

Execute the block if the condition holds. This is a simple comparison between two values, where the comparison may be equals, not equals, less than, greater than, less than or equal, or greater than or equal as represented by the operators ==, **!**=, <, >, <=, and >=, respectively.

UNLESS ...

All of the forms of the **IF** construct may be used with the word **UNLESS** substituted for **IF**. In this case, the sense of the conditional test is reversed.

REPEAT [*n*] [**AS** *var*]**:** ... **END**

Repeat the commands in the block until forced to quit via **EXIT** or **BREAK**. If a repeat count *n* is given, then at most only repeat the commands that many times. The **AS** clause allows the current loop counter (which begins at 0 and counts up to, but never reaches, *n*) to be visible within the block as the variable *var*.

FOR *var*=*start* **TO** *end* [**BY** *step*]**:** ... **END**

Repeat the commands in the block once for each value of *var* from *start* to *end*, inclusive. If *step* is given, *var* is incremented by that amount between each iteration. The default *step* is 1. The behavior of this loop may be modified by **BREAK** or **CONTINUE** statements just like the **REPEAT** loops are.

BREAK [*var*]

Terminate the innermost loop immediately. If an outer loop is to be terminated, then specify its index *var* to identify which loop is the target of this command.

CONTINUE [*var*]

Begin the next iteration of the innermost loop immediately. If an outer loop is to be continued, then specify its index *var* to identify which loop is the target of this command.

SEQUENCE *id* [(*var***,** ...**)]:** ... **END**

Define a stored sequence consisting of the statements to the **END**. The *id* may be a number from 0 to 127, with a few caveats:

0 Sequence #0 may *never* be explicitly invoked by anything. This sequence is invoked automatically during device boot to initialize the unit.

1−63 Sequences in this range are stored in EEPROM and will survive a power failure or device reboot. Note that there probably won't be enough available memory to actually store 63 sequences.

64−127 Sequences in this range are stored in RAM and will *not* survive a power failure or device reboot. Note that there probably won't be enough available memory to actually store 64 sequences in RAM.

If input parameters are expected, they are given symbolic names inside parentheses between the sequence *id* and the colon. A maximum of four parameters are allowed.

Commands

Each command is described briefly here. For more details, see the Lumos board user manual.

BLACKOUT

Immediately turn off all channel outputs.

CALL *id* [**(***value***,** ...**)**]

Suspend the execution of the current sequence and begin executing sequence #*id*. When that sequence has completed, execution of the current sequence will resume where it left off. Up to four parameters may be passed to the called sequence by placing them inside parentheses after the sequence *id*. The *id* cannot be 0.

CHANNEL *c* **ON|OFF|***value*[**%**]

Change the output level of channel *c* to fully on, fully off, or to a specific value. If the value is followed by a percent sign ("**%**"), then the value is assumed to be a percentage in the range [0, 100]. Otherwise the value interpreted as an absolute output value in the range [0, 255]. Note that sequences are always considered to be "high resolution" so this number range is used ragardless of whether the board itself is in high or low resolution mode.

EXECUTE *id* [**(***value***,** ...**)**]

Abandon the current sequence and begin executing the specified sequence instead. When that sequence has completed, execution of the current sequence will not resume. Up to four parameters may be passed to the new sequence by placing them inside parentheses after the sequence *id*. Note that the *id* cannot be 0.

EXIT Terminate execution of the current sequence.

RAMP UP|DOWN *c* [**BY** *steps* [**PER** *time* [**SEC[ONDS]]]**

Start fading channel *c* up or down from its current output level to the maximum or minimum value. This is done by adding or subtracting the value of *steps* (which is in the range [1, 128]) each time, pausing for *time* between each value change. If the *time* value is followed by **SEC** or **SECONDS**, it is assumed to be the number of seconds between level changes (a real number in the range [0.0083, 1.0667]). Otherwise, it is assumed to be an integer in the range [1, 128] specifying the number of 1/120ths of a second for the pause.

RESUME [**WITH FADE**]

Resume from **SUSPEND** mode. If the **WITH FADE** option is given, then all channels are faded down to fully off, and then up to their last-known state is supposed to be.

SLEEP

Enter sleep state. The load power supply is commanded to shut down (if such a supply is connected that way to the controller). Note that the unit may still wake on its own as needed, possibly even immediately.

SUSPEND [**WITH UPDATE**]

Suspend input of commands from outside. From this point forward, only saved sequences will affect channel outputs. If the **WITH UPDATE** option is given, then external commands are still accepted but have no effect on the outputs until after a **RESUME** is executed. Note that a unit may still remain in suspended state after the sequence putting it in that state exits. It only ends upon **RESUME**.

WAIT *t* [**SEC**[**ONDS**]]

Pause execution of the script for the designated time. The time *t* is interpreted identically to that described above for the **RAMP** command.

WAKE Wake up the unit from **SLEEP** mode. The power supply is commanded to start (if a suitable one is appropriately configured).

AUTHOR

Software Alchemy / support@alchemy.com

COMPATIBILITY

This version of **lumosctl** is compatible with the following boards:

* Lumos 48-channel controller version 3.1 or 3.2 *providing it has been upgraded or installed with ROM firmware version 2.0 or later* (boards with ID markings beginning with “48CTL-3-1” or “LUMOS-48CTL-3.2”). (Whether this controller is driving AC or DC boards is irrelevant.)
* Lumos 24-channel DC controller version 1.0 (boards with ID markings beginning with “LUMOS-24SSR-DC-1.0”).

HISTORY

This program first appeared under then name **48ssrctl** and was used only for the Lumos 48-channel AC controller boards, employing the older firmware (ROM version 1.*x*).

This document describes version 2.0 of this utility, which is the first to carry this name and to include the expanded features for firmware version 2.0.

LIMITATIONS

This program does not send DMX512 commands to the device(s), only Lumos native commands.

BUGS

The sequence language is constrained by the limits of the hardware (such as 8-bit unsigned integer values and limited arithmetic expression evaluation), and can be compiled to fit in a very small memory space. As such, the optimization toward certain use cases and against others may seem odd at first, but it serves that purpose.

Submit any found to support@alchemy.com