NAME

busylight – display busy/free status to passers-by

SYNOPSIS

busylight *state*

busylightd

busylight-standalone *color*

upcoming

OPTIONS

Each command that accepts command-line options is described below. Note that option names may be preceded by one or two hyphens (e.g., either **−−mute** or **−mute**), but options may not be abbreviated or combined.

busylight

The **busylight** CLI tool signals the daemon to manually assume a state change according to the option provided. Multiple *state* options may be given in a single invocation, but note that the signal will only report the most immediately-relevant state at any given time. It is safe to include **−−reload** with other states, but if two conflicting states are given (e.g., **−−mute** and **−−open**), it is undefined which will actually take effect.

−−cal Tell the daemon to return to reporting state based on calendar availability. (This signals that a Zoom call has ended.)

−−kill Tell the daemon to terminate immediately.

−−mute Tell the daemon that we are in a Zoom call with the microphone muted.

−−open Tell the daemon that we are in a Zoom call with the microphone open.

−−reload Force the daemon to re-poll the calendar service to get updates to the schedule rather than waiting for the next periodic poll time.

−−urgent Toggle flashing an urgent-status indication.

−−zzz Toggle the active state of the daemon. If it was active, it turns off the signal light and stops polling the calendar service. Otherwise, it resumes active state, polls the calendar service and updates the signal light appropriately.

See the description of the daemon's handling of the **SIGWINCH** signal for more details.

busylight-standalone

The **busylight-standalone** program is intended more for testing the hardware than actual production use. It connects directly to a (hardcoded) port and commands the light device to display a color based on the option given, which must be one (and only one) of the following. Note that all other lights are turned off when making any of these changes.

−−blue Turn on the blue light.

−−green Turn on the green light.

−−list List the available ports on the system. (Currently it is necessary to change the target port in the source code.)

−−red Turn on the lower red light.

−−red2 Turn on the upper red light.

−−redblue Flash the upper red light and the blue light alternately.

−−redred Flash both red lights alternately.

−−reds Turn on both red lights.

−−off Turn all lights off.

−−yellow Turn on the yellow light.

DESCRIPTION

The tools described here control a hardware status signal attached to the computer's USB port. This is a custom hardware device which employs a simple serial protocol and is not necessarily compatible with anything else.

The normal course of operations is to start up the status monitor daemon, **busylightd**, in the background. This will poll the user's Google calendar(s) to see when they are busy or free, and will continue to poll every hour to keep up with changing schedules throughout the day.

The daemon also monitors the state of a video conferencing meeting such as Zoom, to arrange a set of signals to anyone in visual range of the light, such as:

green	Currently free, and able to be approached/interrupted at will.
yellow	Marked busy on a calendar, and thus may be working on something less amenable to interruption.
red	Actually joined a conference call via Zoom, etc., so should not be interrupted (and possibly on-camera so anyone who comes in camera range may be visible to meeting participants).
flashing red	In a conference call and the microphone is open, so any nearby sounds may be heard by all meeting participants.
flashing red/blue	Indicates an urgent need for help.

The actual monitoring of video meetings and microphone statuses is assumed to be done by some other automation which signals the daemon by sending signals to its process or running the **busylight** CLI tool. The author uses a hammerspoon script to accomplish this.

See the SIGNALS section below for a description of how sending signals to the daemon affect its operation. The **busylight** CLI program is simply a convenient way to inform the daemon of a status change. It locates the daemon process and sends signal(s) to it according to the command-line flags passed to it.

When the daemon starts, it flashes the blue light twice. When exiting, it flashes the red light twice and then turns the light off completely.

The **upcoming** program polls the Google calendars and displays to standard output the busy/free time ranges for the next 8 hours.

Finally, the **busylight-standalone** program provides a way to test the light hardware without the daemon running. It opens the port directly and sends hardware commands to set the light displays as desired.

CONFIGURATION

These tools require a few files to be placed in the user's **~/.busylight** directory. The overall tool configuration will be in a file called **config.json** in that directory.

This file provides all of the configuration parameters needed for the ongoing operation of the system. As the name implies, it is in JSON format, as a single object with the following fields:

Calendars

This is a map of Google calendar IDs to objects which describe those calendars. The data associated with each key is an object with the following fields:

Title

An arbitrary name for the calendar that will explain its purpose.

IgnoreAllDayEvents

A boolean value; if true, **busylightd** will ignore any busy periods for that calendar which span the entire 8-hour period being queried. Defaults to false.

The key **"primary"** may be used in place of the Google ID to refer to the user's primary calendar.

TokenFile
The name of a file in which the program can cache authentication tokens to allow it to continue polling Google calendars. This should be a filename in the **.busylight** directory with restricted permissions to avoid unauthorized viewing.

CredentialFile
The name of a JSON file containing the API access credentials obtained from Google.

LogFile
The name of a file into which **busylightd** should record a log of its activities.

PidFile
The name of the file **busylightd** should use to indicate its PID while running.

Device
The system device name of the busylight signal hardware.

DeviceDir
If **Device** is omitted or blank, then a suitable device will be searched for in the directory named here. See also **DeviceRegexp**.

DeviceRegexp
If searching for a device name in **DeviceDir**, the first device whose name matches the regular expression given here and can be successfully opened as a serial port will be used.

BaudRate
The speed the hardware expects to be used to communicate with it.

An example configuration file would look like this:

```
{
  "Calendars": {
    "primary": {
      "Title": "My primary calendar"
    },
    "mycustomcalendar@group.calendar.google.com": {
      "Title": "Group calendar",
      "IgnoreAllDayEvents": true
    }
  },
  "TokenFile": "/Users/MYNAME/.busylight/auth.json",
  "CredentialFile": "/Users/MYNAME/.busylight/credentials.json",
  "LogFile": "/Users/MYNAME/.busylight/busylightd.log",
  "PidFile": "/Users/MYNAME/.busylight/busylightd.pid",
  "Device":  "/dev/tty.usbmodem2101",
  "BaudRate": 9600
}
```

If using a regular expression for the device rather than a fixed name, the **Device** entry of the above JSON would be replaced with these two:

```
  "DeviceDir": "/dev",
  "DeviceRegexp": "^tty\\.usbmodem\\d+$",
```

AUTHENTICATING

In order to use the daemon to query Google calendar busy/free times, you first need to obtain an API key from Google. This will go in your **~/.busylight/credentials.json** file (or whatever you named it in **~/.busylight/config.json**). An example of this file is:

```
{
  "installed" : {
    "client_id": "...",
    "project_id": "...",
```

```
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_secret":"...",
    "redirect_uris": ["urn:ietf:wg:oauth:2.0:oob","http://localhost"]
  }
}
```

Next, you will need to manually authenticate to Google once before the daemon can continue to poll the calendar API on its own. To do this, run the **upcoming** program. If you already have valid access tokens cached, it will simply report your busy/free times for the next 8 hours. Otherwise, it will print a lengthy URL on its standard output and wait for your response.

Copy that URL into a web browser. This will take you to Google where it will ask you to log in to the Google account whose calendars you wish to have monitored. You will also be asked if you are sure you want to give permissions to the app to have acceess to all of your calendars. If you agree, Google will give you an access token string.

Copy that string and paste it into the terminal where you are running **upcoming** so it is sent to **upcoming**'s standard input and press the return key.

This will authorize the client to access the calendar API, so **upcoming** will then print out its report of your upcoming appointment times. But in doing so it will also have cached your authentication token in the **~/.busylight/auth.json** file (or whatever you named it in **config.json**), so the programs documented here may freely poll the calendar service using that token.

If the busylight tools suddenly stop being able to access the calendar, simply delete the **auth.json** file and repeat this process to get a new token cached.

Security Implications

Protect the data in the **auth.json** file carefully. Any program with access to that data will have full rights to view and modify your Google calendars.

When you no longer wish to authorize these tools to access your calendars, you may go into your Google account settings on Google's website to revoke that authorization.

SIGNALS

The **busylightd** daemon responds to the following signals:

HUP The video conference call is over. The daemon changes the light signal to reflect the user's busy/free status as understood from the last poll of the Google calendars.

INFO The daemon will immediately poll the calendar API instead of waiting for the next scheduled poll time. This is useful if a last-minute change was made to the calendar. This does not otherwise alter the periodic polling schedule (e.g., if the daemon is polling at 5 minutes past each hour, and this signal is received at 3:45, the next poll will still take place at 4:05).

INT Upon receipt of this signal, the daemon gracefully shuts down and terminates.

VTALRM Toggles urgent indicator status. Initially it makes the light signal display an urgent flashing pattern. When received again, the daemon resumes normal display.

USR1 The user is in a video conference with the microphone muted. The light signal is changed to reflect this.

USR2 The user is in a video conference with the microphone open. The light signal is changed to reflect this.

WINCH Toggle whether the daemon is active or not. This is usually used to mark the start and end of the workday. When active, the daemon performs all of the functions documented here, polling the Google calendar hourly to pick up any changes to the schedule. When inactive, the light signal is shut off completely and the daemon stops polling the calendar service. Upon startup or resuming from inactive state, the daemon will immediately poll the calendar service, and

will then poll again an hour after that, and every hour thereafter.

When resuming active status after having been inactive, the daemon will reload the configuration file. This provides a convenient way to change configuration options by suspending operations and then resuming, without needing to completely restart the daemon. The PID and log files may not be changed without restarting the daemon completely. Also note that the API credentials for accessing Google calendars is not reloaded at this time. That also requires a full restart of the daemon process.

The serial port is closed while the daemon is in inactive state.

AUTHOR

Steve Willoughby <*steve@madscience.zone*>

PORTABILITY

The author's intended use for the daemon was on a Macintosh osx system, and the choice of signals was based on their availability on that platform. Other operating systems may not support all of those signals, so porting to those systems may involve a different selection of signals.

BUGS

The **busylight-standalone** program is not intended for production use and thus still has some needed refactoring. In particular, the device port is hardcoded into it rather than using a configuration file as the other tools do.