NAME

busylight – display busy/free status to passers-by

SYNOPSIS

busylight [**−cal**] [**−help**] [**−kill**] [**−list**] [**−mute**] [**−open**] [**−query**] [**−raw** *command*] [**−reload**] [**−status** *name*] [**−wake**] [**−zzz**]

busylightd

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** command manually sets the status display on the light device. If a daemon is running, it will attempt to notify the daemon for those states which it's also tracking (**−cal**, **−mute**, and **−open**), or for commands which directly manipulate the daemon itself (**−kill**, **−reload**, **−wake**, and **−zzz**).

The options recognized include the following:

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

−help Summarize the command-line options and exit.

−kill Tell the daemon to terminate immediately.

−list List all the status codes usable with the **−status** option and then exit.

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

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

−query Queries the hardware state and reports it to the user.

−raw *command* Send the *command* string as-is to the light controller device. See below.

−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.

−status *name* Set the light tree device to the status light pattern defined for the given *name* in the configuration file (does not notify the daemon).

−wake Tell the daemon to come on line if it was sleeping. The Google calendars are polled and resulting status is displayed by the daemon.

−zzz Tells the daemon to go to sleep; turns off the signal light and stops polling the calendar service.

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.

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 a convenient way to inform the daemon of a status change as an alternative to sending signals directly to the daemon. It is also used to directly send commands to the light control device regardless of whether the daemon is running or not.

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

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:

Colors
: This is a string containing single-character names for each of the LEDs as implemented on your particular hardware device. For example, a light with three LEDs—red, yellow, and green—with red in position #0 might be represented by a value of **"RYG"**. These are used for reporting device status via the **–query** option.

ColorValues
: This is an object mapping color letters as listed in the **Colors** field to actual color names or **#***rrggbb* values as accepted by tk.

StatusLights
: This is a map which defines a symbolic name for each signal pattern you wish to display. You are not limited to only these patterns, but the advantage is that you may refer to them by the defined names rather than the raw codes. The following names are needed by **busylightd** (although defaults will be used if they are not defined here in the config file):

  busy Signals that you are busy (per your calendar events). Defaults to "**S3**".

  free Signals that you are free (per your calendar events). Defaults to "**S4**".

  muted Signals that you are in a meeting with your microphone muted. Defaults to "**S2**".

  off Turns off all lights. Defaults to "**X**".

  open Signals that you are in a meeting with your microphone open. Defaults to "**F12$**".

  start Flashed twice rapidly when the daemon starts up or wakes. Defaults to "**S0**".

  stop Flashed twice rapidly when the daemon stops or sleeps. Defaults to "**S1**".

  In each case, the raw codes used to display light patterns may include any combination of the following:

  F*n...***$** Flash one or more lights in sequence. If a single value is given for *n*, that light is flashed. If multiple light numbers are given (e.g., "**F12$**") they are sequenced in a repeating cycle. Sequences of up to 64 elements are accepted. The "**$**" terminator may be an ASCII ESC character or a dollar sign.

  S*n* Turn on light #*n*. Only one of these may be on at once, and this is mutually exclusive with **F**.

*****_n_**...$** Strobe one or more lights in sequence. This may be combined with other effects. The light(s) indicated are very briefly flashed, with a longer pause between each flash. If there are no lights listed at all, this cancels the strobe effect. The terminator is as described above. Sequences of up to 64 elements are supported.

X Turn off all lights.

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:

```
{
  "Colors": "BrRYG",
  "ColorValues": {
        "B": "blue",
        "R": "red",
        "r": "red",
        "Y": "yellow",
        "G": "green"
  },
  "StatusLights": {
    "busy": "S3",
    "free": "S4",
    "urgent": "F01$"
  },
```

```
  "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 might 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 Instructs the daemon to wake up from sleep 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.

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 Put the daemon to sleep. This is usually used to mark the end of the workday. The light signal is shut off completely and the daemon stops polling the calendar service.

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.