i3pystatus Documentation

Release

Author

July 14, 2016

Contents

Contents:

CHAPTER 1

Configuration

The config file is just a normal Python script.

A simple configuration file could look like this (note the additional dependencies from *network*, *wireless* and *pulseaudio* in this example):

```
# -*- coding: utf-8 -*-

import subprocess

from i3pystatus import Status

status = Status(standalone=True)

# Displays clock like this:
# Tue 30 Jul 11:59:46 PM KW31
#                          ^-- calendar week
status.register("clock",
    format="%a %-d %b %X KW%V",)

# Shows the average load of the last minute and the last 5 minutes
# (the default value for format is used)
status.register("load")

# Shows your CPU temperature, if you have a Intel CPU
status.register("temp",
    format="{temp:.0f}°C",)

# The battery monitor has many formatting options, see README for details

# This would look like this, when discharging (or charging)
# ↓14.22W 56.15% [77.81%] 2h:41m
# And like this if full:
# =14.22W 100.0% [91.21%]
#
# This would also display a desktop notification (via dbus) if the percentage
# goes below 5 percent while discharging. The block will also color RED.
status.register("battery",
    format="{status}/{consumption:.2f}W {percentage:.2f}% [{percentage_design:.2f}%] {remaining:%E%h
    alert=True,
    alert_percentage=5,
    status={
        "DIS": "↓",
        "CHR": "↑",
```

```
        "FULL": "=",
    },)

# This would look like this:
# Discharging 6h:51m
status.register("battery",
    format="{status} {remaining:%E%hh:%Mm}",
    alert=True,
    alert_percentage=5,
    status={
        "DIS":  "Discharging",
        "CHR":  "Charging",
        "FULL": "Bat full",
    },)

# Displays whether a DHCP client is running
status.register("runwatch",
    name="DHCP",
    path="/var/run/dhclient*.pid",)

# Shows the address and up/down state of eth0. If it is up the address is shown in
# green (the default value of color_up) and the CIDR-address is shown
# (i.e. 10.10.10.42/24).
# If it's down just the interface name (eth0) will be displayed in red
# (defaults of format_down and color_down)
#
# Note: the network module requires PyPI package netifaces
status.register("network",
    interface="eth0",
    format_up="{v4cidr}",)

# Has all the options of the normal network and adds some wireless specific things
# like quality and network names.
#
# Note: requires both netifaces and basiciw
status.register("wireless",
    interface="wlan0",
    format_up="{essid} {quality:03.0f}%",)

# Shows disk usage of /
# Format:
# 42/128G [86G]
status.register("disk",
    path="/",
    format="{used}/{total}G [{avail}G]",)

# Shows pulseaudio default sink volume
#
# Note: requires libpulseaudio from PyPI
status.register("pulseaudio",
    format="♪{volume}",)

# Shows mpd status
# Format:
# Cloud connectedReroute to Remain
status.register("mpd",
    format="{title}{status}{album}",
    status={
```

```
        "pause": "",
        "play": "",
        "stop": "",
    },)

status.run()
```

Also change your i3wm config to the following:

```
# i3bar
bar {
    status_command    python ~/.path/to/your/config/file.py
    position          top
    workspace_buttons yes
}
```

CHAPTER 2

Formatting

All modules let you specifiy the exact output formatting using a format string, which gives you a great deal of flexibility.

If a module gives you a float, it probably has a ton of uninteresting decimal places. Use `{somefloat:.0f}` to get the integer value, `{somefloat:0.2f}` gives you two decimal places after the decimal dot

2.1 formatp

Some modules use an extended format string syntax (the mpd module, for example). Given the format string below the output adapts itself to the available data.

```
[{artist}/{album}/]{title}{status}
```

Only if both the artist and album is known they're displayed. If only one or none of them is known the entire group between the brackets is excluded.

"is known" is here defined as "value evaluating to True in Python", i.e. an empty string or 0 (or 0.0) counts as "not known".

Inside a group always all format specifiers must evaluate to true (logical and).

You can nest groups. The inner group will only become part of the output if both the outer group and the inner group are eligible for output.

2.2 TimeWrapper

Some modules that output times use TimeWrapper to format these. TimeWrapper is a mere extension of the standard formatting method.

The time format that should be used is specified using the format specifier, i.e. with some_time being 3951 seconds a format string like `{some_time:%h:%m:%s}` would produce `1:5:51`.

- `%h`, `%m` and `%s` are the hours, minutes and seconds without leading zeros (i.e. 0 to 59 for minutes and seconds)
- `%H`, `%M` and `%S` are padded with a leading zero to two digits, i.e. 00 to 59
- `%l` and `%L` produce hours non-padded and padded but only if hours is not zero. If the hours are zero it produces an empty string.
- `%%` produces a literal %

- `%E` (only valid on beginning of the string) if the time is null, don't format anything but rather produce an empty string. If the time is non-null it is removed from the string.
- When the module in question also uses formatp, 0 seconds counts as "not known".
- The formatted time is stripped, i.e. spaces on both ends of the result are removed.

CHAPTER 3

Module reference

System *clock* - *disk* - *load* - *mem* - *cpu_usage*

Audio *alsa* - *pulseaudio*

Hardware *battery* - *backlight* - *temp*

Network *network* - *wireless*

Music *now_playing* - *mpd*

Websites & stuff *weather* - *bitcoin* - *reddit* - *parcel*

Other *mail* - *pyload* - *text*

Advanced *file* - *regex* - *runwatch* - *shell*

Note: List of all modules:

- *network*
- *network_graph*
- *network_traffic*
- *now_playing*
- *parcel*
- *pomodoro*
- *pulseaudio*
- *pyload*
- *reddit*
- *regex*
- *runwatch*
- *shell*
- *spotify*
- *temp*
- *text*
- *uptime*
- *weather*
- *wireless*

---

class i3pystatus.alsa.**ALSA**

Shows volume of ALSA mixer. You can also use this for inputs, btw.

Requires pyalsaaudio

Available formatters

- *{volume}* — the current volume in percent
- *{muted}* — the value of one of the *muted* or *unmuted* settings
- *{card}* — the associated soundcard
- *{mixer}* — the associated ALSA mixer

Settings

- **format** (default: ♪: `{volume}`)
- **format_muted** (default: *empty*) – optional format string to use when muted
- **mixer** (default: `Master`) – ALSA mixer
- **mixer_id** (default: `0`) – ALSA mixer id
- **card** (default: `0`) – ALSA sound card
- **increment** (default: `5`) – integer percentage of max volume to in/decrement volume on mousewheel

- **muted** (default: `M`)
- **unmuted** (default: *empty*)
- **color_muted** (default: `#AAAAAA`)
- **color** (default: `#FFFFFF`)
- **channel** (default: `0`)
- **interval** (default: `1`)

class `i3pystatus.backlight.`**Backlight**
Screen backlight info

Available formatters

- *{brightness}* — current brightness relative to max_brightness
- *{max_brightness}* — maximum brightness value
- *{percentage}* — current brightness in percent

Settings

- **format** (default: `{brightness}/{max_brightness}`) – format string, formatters: brightness, max_brightness, percentage
- **backlight** (default: `acpi_video0`) – backlight, see */sys/class/backlight/*
- **color** (default: `#FFFFFF`)
- **interval** (default: `5`)

class `i3pystatus.battery.`**BatteryChecker**
This class uses the /sys/class/power_supply/. . . /uevent interface to check for the battery status

Available formatters

- *{remaining}* — remaining time for charging or discharging, uses TimeWrapper formatting, default format is *%E%h:%M*
- *{percentage}* — battery percentage relative to the last full value
- *{percentage_design}* — absolute battery charge percentage
- *{consumption (Watts)}* — current power flowing into/out of the battery
- *{status}*
- *{battery_ident}* — the same as the setting
- *{bar}* —bar displaying the percentage graphically

Settings

- **battery_ident** (default: `BAT0`) – The name of your battery, usually BAT0 or BAT1
- **format** (default: `{status} {remaining}`)
- **not_present_text** (default: `Battery not present`) – Text displayed if the battery is not present. No formatters are available
- **alert** (default: `False`) – Display a libnotify-notification on low battery
- **alert_percentage** (default: `10`)
- **alert_format_title** (default: `Low battery`) – The title of the notification, all formatters can be used
- **alert_format_body** (default: `Battery {battery_ident} has only {percentage:.2f}% ({remaining:%E%hh:%Mm}) remaining!`) – The body text of the notification, all formatters can be used
- **path** (default: *empty*) – Override the default-generated path
- **status** (default: `{'DIS': 'DIS', 'FULL': 'FULL', 'CHR': 'CHR'}`) – A dictionary mapping ('DIS', 'CHR', 'FULL') to alternative names
- **color** (default: `#ffffff`) – The text color
- **full_color** (default: `#00ff00`) – The full color
- **charging_color** (default: `#00ff00`) – The charging color
- **critical_color** (default: `#ff0000`) – The critical color
- **not_present_color** (default: `#ffffff`) – The not present color.
- **no_text_full** (default: `False`) – Don't display text when battery is full - 100%
- **interval** (default: `5`)

class i3pystatus.bitcoin.**Bitcoin**

This module fetches and displays current Bitcoin market prices and optionally monitors transactions to and from a list of user-specified wallet addresses. Market data is pulled from the BitcoinAverage Price Index API <https://bitcoinaverage.com> while transaction data is pulled from blockchain.info <https://blockchain.info/api/blockchain_api>.

Available formatters

- {last_price}
- {ask_price}
- {bid_price}
- {daily_average}
- {volume}
- {status}
- {last_tx_type}
- {last_tx_addr}
- {last_tx_value}
- {balance_btc}

- {balance_fiat}

Settings

- **format** (default: Ƀ `{status}{last_price}`) – Format string used for output.
- **currency** (default: `USD`) – Base fiat currency used for pricing.
- **wallet_addresses** (default: *empty*) – List of wallet address(es) to monitor.
- **color** (default: `#FFFFFF`) – Standard color
- **colorize** (default: `False`) – Enable color change on price increase/decrease
- **color_up** (default: `#00FF00`) – Color for price increases
- **color_down** (default: `#FF0000`) – Color for price decreases
- **leftclick** (default: `electrum`) – URL to visit or command to run on left click
- **rightclick** (default: `https://bitcoinaverage.com/`) – URL to visit or command to run on right click
- **interval** (default: `600`) – Update interval.
- **status** (default: `{'price_up': '', 'price_down': ''}`)

class `i3pystatus.clock.`**`Clock`**
This class shows a clock

Settings

- **format** (default: *empty*) – list of tuple (stftime format string, optional timezone), *None* means to use the default, locale-dependent format. Can cycle between formats with mousewheel
- **color** (default: `#ffffff`) – RGB hexadecimal code color specifier, default to #ffffff, set to *i3Bar* to use i3 bar default
- **interval** (default: `1`)

class `i3pystatus.cmus.`**`Cmus`**
gets the status and current song info using cmus-remote

Settings

- **format** (default: `{status} {song_elapsed}/{song_length} {artist}-{title}`)
- **color** (default: `#909090`)
- **interval** (default: `1`)

class `i3pystatus.cpu_usage.`**`CpuUsage`**
Shows CPU usage. The first output will be inacurate.

Linux only

Available formatters

- {usage} usage average of all cores
- {usage_cpu*} usage of one specific core. replace "*" by core number starting at 0
- {usage_all} usage of all cores separate. usess natsort when available(relevant for more than 10 cores)

Settings

- **format** (default: `{usage:02}%`) – format string.
- **format_all** (default: `{core}:{usage:02}%`) – format string used for {usage_all} per core. Available formaters are {core} and {usage}.
- **exclude_average** (default: `False`) – If True usage average of all cores will not be in format_all.
- **interval** (default: `5`)

calculate_usage(*cpu*, *total*, *busy*)
calculates usage

gen_format_all(*usage*)
generates string for format all

get_cpu_timings()
reads and parses /proc/stat returns dictionary with all available cores including global average

get_usage()
parses /proc/stat and calcualtes total and busy time (more specific USER_HZ see man 5 proc for further informations)

class i3pystatus.cpu_usage_bar.**CpuUsageBar**
Shows CPU usage as a bar (made with unicode box characters). The first output will be inacurate.

Linux only

Requires the PyPI package *colour*.

Available formatters

- {usage_bar} usage average of all cores
- {usage_bar_cpuN} usage of one specific core. replace "N" by core number starting at 0

Settings

- **format** (default: `{usage_bar}`) – format string
- **bar_type** (default: `horizontal`) – whether the bar should be vertical or horizontal. Allowed values: *vertical* or *horizontal*
- **cpu** (default: `usage_bar`) – cpu to base the colors on. Choices are 'usage_cpu' for all or 'usage_cpu*'. Replace '*' by core number starting at 0.
- **start_color** (default: `#00FF00`) – Hex or English name for start of color range, eg '#00FF00' or 'green'
- **end_color** (default: `red`) – Hex or English name for end of color range, eg '#FF0000' or 'red'

- **interval** (default: `5`)

class `i3pystatus.cpu_usage_graph.`**`CpuUsageGraph`**
Shows CPU usage as a Unicode graph. The first output will be inacurate.

Depends on the PyPI colour module - https://pypi.python.org/pypi/colour/0.0.5

Linux only

Available formatters

- {cpu_graph} graph of cpu usage.
- {usage} usage average of all cores
- {usage_cpu*} usage of one specific core. replace "*" by core number starting at 0
- {usage_all} usage of all cores separate. usess natsort when available(relevant for more than 10 cores)

Settings

- **cpu** (default: `usage_cpu`) – cpu to monitor, choices are 'usage_cpu' for all or 'usage_cpu*'. Replace '*' by core number starting at 0.
- **start_color** (default: `#00FF00`) – Hex or English name for start of color range, eg '#00FF00' or 'green'
- **end_color** (default: `red`) – Hex or English name for end of color range, eg '#FF0000' or 'red'
- **interval** (default: `5`)

class `i3pystatus.disk.`**`Disk`**
Gets `{used}`, `{free}`, `{available}` and `{total}` amount of bytes on the given mounted filesystem.

These values can also be expressed as percentages with the `{percentage_used}`, `{percentage_free}` and `{percentage_avail}` formats.

Settings

- **format** (default: `{free}/{avail}`)
- **path** (required)
- **divisor** (default: `1073741824`) – divide all byte values by this value, default is 1024**3 (gigabyte)
- **display_limit** (default: `inf`) – if more space is available than this limit the module is hidden
- **critical_limit** (default: `0`) – critical space limit (see critical_color)
- **critical_color** (default: `#FF0000`) – the critical color
- **color** (default: `#FFFFFF`) – the common color
- **round_size** (default: `2`) – precision, None for INT
- **interval** (default: `5`)

class `i3pystatus.file.`**`File`**
Rip information from text files

components is a dict of pairs of the form:

```
name => (callable, file)
```

- Where *name* is a valid identifier, which is used in the format string to access the value of that component.
- *callable* is some callable to convert the contents of *file*. A common choice is float or int.
- *file* names a file, relative to *base_path*.

transforms is a optional dict of callables taking a single argument (a dictionary containing the values of all components). The return value is bound to the key.

Settings

- **format** (required)
- **components** (required)
- **transforms** (default: `{}`)
- **base_path** (default: `/`)
- **color** (default: `#FFFFFF`)
- **interval** (default: `5`)

class `i3pystatus.load.`**`Load`**
Shows system load

Settings

- **format** (default: `{avg1} {avg5}`) – format string used for output. {avg1}, {avg5} and {avg15} are the load average of the last one, five and fifteen minutes, respectively. {tasks} is the number of tasks (i.e. 1/285, which indiciates that one out of 285 total tasks is runnable).
- **color** (default: `#ffffff`) – The text color
- **critical_limit** (default: `1`) – Limit above which the load is considered critical
- **critical_color** (default: `#ff0000`) – The critical color
- **interval** (default: `5`)

class `i3pystatus.mail.`**`Mail`**
Generic mail checker

The *backends* setting determines the backends to use. For available backends see *Mail Backends*.

Settings

- **backends** – List of backends (instances of `i3pystatus.mail.xxx.zzz`, e.g. *`imap.IMAP`*)
- **color** (default: `#ffffff`)
- **color_unread** (default: `#ff0000`)
- **format** (default: `{unread} new email`)
- **format_plural** (default: `{unread} new emails`)

- **hide_if_null** (default: `True`) – Don't output anything if there are no new mails
- **email_client** (default: *empty*) – The command to run on left click. For example, to launch Thunderbird set `email_client` to ``thunderbird``. Alternatively, to bring Thunderbird into focus, set `email_client` to `i3-msg -q [class="^Thunderbird$"] focus`. Hint: To discover the X window class of your email client run 'xprop | grep -i class' and click on it's window
- **interval** (default: `5`)

class `i3pystatus.mem.`**Mem**

Shows memory load

Available formatters

- {avail_mem}
- {percent_used_mem}
- {used_mem}
- {total_mem}

Requires psutil (from PyPI)

Settings

- **format** (default: `{avail_mem} MiB`) – format string used for output.
- **divisor** (default: `1048576`) – divide all byte values by this value, default 1024**2(mebibytes
- **warn_percentage** (default: `50`) – minimal percentage for warn state
- **alert_percentage** (default: `80`) – minimal percentage for alert state
- **color** (default: `#00FF00`) – standard color
- **warn_color** (default: `#FFFF00`) – defines the color used wann warn percentage ist exceeded
- **alert_color** (default: `#FF0000`) – defines the color used when alert percentage is exceeded
- **round_size** (default: `1`) – defines number of digits in round
- **interval** (default: `5`)

class `i3pystatus.mem_bar.`**MemBar**

Shows memory load as a bar.

Available formatters

- {used_mem_bar}

Requires psutil and colour (from PyPI)

Settings

- **format** (default: `{used_mem_bar}`) – format string used for output.
- **warn_percentage** (default: `50`) – minimal percentage for warn state
- **alert_percentage** (default: `80`) – minimal percentage for alert state
- **color** (default: `#00FF00`) – standard color
- **warn_color** (default: `#FFFF00`) – defines the color used wann warn percentage ist exceeded
- **alert_color** (default: `#FF0000`) – defines the color used when alert percentage is exceeded
- **multi_colors** (default: `False`) – whether to use range of colors from 'color' to 'alert_color' based on memory usage.
- **interval** (default: `5`)

class `i3pystatus.modsde.`**ModsDeChecker**

This class returns i3status parsable output of the number of unread posts in any bookmark in the mods.de forums.

Settings

- **format** (default: `{unread} new posts in bookmarks`) – Use {unread} as the formatter for number of unread posts
- **offset** (default: `0`) – subtract number of posts before output
- **color** (default: `#7181fe`)
- **username** (required)
- **password** (required)
- **interval** (default: `5`)

class `i3pystatus.mpd.`**MPD**

Displays various information from MPD (the music player daemon)

Available formatters (uses *formatp*)

- *{title}* — (the title of the current song)
- *{album}* — (the album of the current song, can be an empty string (e.g. for online streams))
- *{artist}* — (can be empty, too)
- *{filename}* — (file name with out extension and path; empty unless title is empty)
- *{song_elapsed}* — (Position in the currently playing song, uses *TimeWrapper*, default is *%m:%S*)
- *{song_length}* — (Length of the current song, same as song_elapsed)
- *{pos}* — (Position of current song in playlist, one-based)
- *{len}* — (Songs in playlist)
- *{status}* — (play, pause, stop mapped through the *status* dictionary)
- *{bitrate}* — (Current bitrate in kilobit/s)
- *{volume}* — (Volume set in MPD)

Left click on the module play/pauses, right click (un)mutes.

Settings

- **host** (default: `localhost`)
- **port** (default: `6600`) – MPD port
- **format** (default: `{title} {status}`) – formatp string
- **status** (default: `{'pause': '', 'play': '', 'stop': ''}`) – Dictionary mapping pause, play and stop to output
- **color** (default: `#FFFFFF`) – The color of the text
- **interval** (default: `1`)

class `i3pystatus.network.`**Network**

Display network information about a interface.

Requires the PyPI package *netifaces*.

Available formatters

- *{interface}* — same as setting
- *{name}* — same as setting
- *{v4}* — IPv4 address
- *{v4mask}* — subnet mask
- *{v4cidr}* — IPv4 address in cidr notation (i.e. 192.168.2.204/24)
- *{v6}* — IPv6 address
- *{v6mask}* — subnet mask
- *{v6cidr}* — IPv6 address in cidr notation
- *{mac}* — MAC of interface

Not available addresses (i.e. no IPv6 connectivity) are replaced with empty strings.

Settings

- **interface** (default: `eth0`) – Interface to obtain information for
- **format_up** (default: `{interface}: {v4}`)
- **color_up** (default: `#00FF00`)
- **format_down** (default: `{interface}`)
- **color_down** (default: `#FF0000`)
- **detached_down** (default: `True`) – If the interface doesn't exist, display it as if it were down
- **unknown_up** (default: `False`) – If the interface is in unknown state, display it as if it were up
- **name** (default: `eth0`)
- **interval** (default: `5`)

class i3pystatus.network_graph.**NetworkGraph**
Shows Network activity as a Unicode graph

Linux only

Requires the PyPI packages *psutil* and *colour*.

Available formatters

- {kbs} – Float representing kbs
- {network_graph} – Unicode network graph

Settings

- **format** (default: `{network_graph}{kbs}KB/s`) – format string
- **graph_width** (default: `15`) – Width of the graph
- **upper_limit** (default: `150.0`) – Expected max kb/s. This value controls how the graph is drawn and in what color
- **graph_type** (default: `input`) – Whether to draw the graph for input or output. Allowed values 'input' or 'output'
- **divisor** (default: `1024`) – divide all byte values by this value
- **interface** (default: `eth0`) – Interface to watch, eg 'eth0'
- **start_color** (default: `#00FF00`) – Hex or English name for start of color range, eg '#00FF00' or 'green'
- **end_color** (default: `red`) – Hex or English name for end of color range, eg '#FF0000' or 'red'
- **interval** (default: `1`)

class i3pystatus.network_traffic.**NetworkTraffic**
Network traffic per interface, i.e., packets/bytes sent/received per second.

Requires the PyPI packages *psutil*.

Available formatters

- *{interface}* — the configured network interface
- *{bytes_sent}* — bytes sent per second (divided by divisor)
- *{bytes_recv}* — bytes received per second (divided by divisor)
- *{packets_sent}* — bytes sent per second (divided by divisor)
- *{packets_recv}* — bytes received per second (divided by divisor)

Settings

- **format** (default: `{interface} {bytes_sent}kB/s {bytes_recv}kB/s`) – format string
- **interface** (default: `eth0`) – network interface
- **divisor** (default: `1024`) – divide all byte values by this value

- **round_size** (default: *empty*) – defines number of digits in round
- **interval** (default: `1`)

class `i3pystatus.now_playing.`**NowPlaying**

Shows currently playing track information, supports most media players

Available formatters (uses *formatp*)

- *{title}* — (the title of the current song)
- *{album}* — (the album of the current song, can be an empty string (e.g. for online streams))
- *{artist}* — (can be empty, too)
- *{filename}* — (file name with out extension and path; empty unless title is empty)
- *{song_elapsed}* — (Position in the currently playing song, uses *TimeWrapper*, default is *%m:%S*)
- *{song_length}* — (Length of the current song, same as song_elapsed)
- *{status}* — (play, pause, stop mapped through the *status* dictionary)
- *{volume}* — (Volume)

Left click on the module play/pauses, right click goes to the next track.

Requires python-dbus available from every distros' package manager.

Settings

- **player** (default: *empty*) – Player name
- **status** (default: `{'pause': '', 'play': '', 'stop': ''}`) – Dictionary mapping pause, play and stop to output text
- **color** (default: `#FFFFFF`) – Text color
- **format** (default: `{title} {status}`) – formatp string
- **hide_no_player** (default: `True`) – Hide output if no player is detected
- **interval** (default: `1`)

class `i3pystatus.parcel.`**ParcelTracker**

Used to track parcel/shipments.

Supported carriers: DHL, UPS, Itella

- parcel.UPS("<id_code>")
- parcel.DHL("<id_code>")
- parcel.Itella("<id_code>"[, "en"|"fi"|"sv"]) Second parameter is language. Requires beautiful soup 4 (bs4)

Requires lxml and cssselect.

Settings

- **instance** – Tracker instance, for example `parcel.UPS('your_id_code')`
- **format** (default: `{name}:{progress}`)
- **name**

- **interval** (default: `60`)

class `i3pystatus.pomodoro.`**Pomodoro**
This plugin shows Pomodoro timer.

Left click starts/restarts timer. Right click stops it.

Settings

- **sound** – Path to sound file to play as alarm. Played by "aplay" utility
- **pomodoro_duration** (default: `1500`) – Working (pomodoro) interval duration in seconds
- **break_duration** (default: `300`) – Short break duration in secods
- **long_break_duration** (default: `900`) – Long break duration in secods
- **short_break_count** (default: `3`) – Short break count before first long break
- **interval** (default: `1`)

class `i3pystatus.pulseaudio.`**PulseAudio**
Shows volume of default PulseAudio sink (output).

- Requires amixer for toggling mute and incrementing/decrementing volume on scroll.
- Depends on the PyPI colour module - https://pypi.python.org/pypi/colour/0.0.5

Available formatters:

- *{volume}* — volume in percent (0...100)
- *{db}* — volume in decibels relative to 100 %, i.e. 100 % = 0 dB, 50 % = -18 dB, 0 % = -infinity dB (the literal value for -infinity is -∞)
- *{muted}* — the value of one of the *muted* or *unmuted* settings
- *{volume_bar}* — unicode bar showing volume

Settings

- **format** (default: ♪: `{volume}`)
- **format_muted** (default: *empty*) – optional format string to use when muted
- **muted** (default: `M`)
- **unmuted** (default: *empty*)
- **color_muted** (default: `#FF0000`)
- **color_unmuted** (default: `#FFFFFF`)
- **step** (default: `5`) – percentage to increment volume on scroll
- **bar_type** (default: `vertical`) – type of volume bar. Allowed values are 'vertical' or 'horizontal'
- **multi_colors** (default: `False`) – whether or not to change the color from 'color_muted' to 'color_unmuted' based on volume percentage
- **vertical_bar_width** (default: `2`) – how many characters wide the vertical volume_bar should be

context_notify_cb(*context*, *_*)
Checks wether the context is ready

-Queries server information (server_info_cb is called) -Subscribes to property changes on all sinks (update_cb is called)

init()
Creates context, when context is ready context_notify_cb is called

request_update(*context*)
Requests a sink info update (sink_info_cb is called)

server_info_cb(*context*, *server_info_p*, *userdata*)
Retrieves the default sink and calls request_update

sink_info_cb(*context*, *sink_info_p*, *_*, *__*)
Updates self.output

update_cb(*context*, *t*, *idx*, *userdata*)
A sink property changed, calls request_update

class i3pystatus.pyload.**pyLoad**
Shows pyLoad status

Available formatters

- *{captcha}* (see captcha_true and captcha_false, which are the values filled in for this formatter)
- *{progress}* (average over all running downloads)
- *{progress_all}* (percentage of completed files/links in queue)
- *{speed}* (kilobytes/s)
- *{download}* (downloads enabled, also see download_true and download_false)
- *{total}* (number of downloads)
- *{free_space}* (free space in download directory in gigabytes)

Settings

- **address** (default: `http://127.0.0.1:8000`) – Address of pyLoad webinterface
- **format** (default: `{captcha} {progress_all:.1f}% {speed:.1f} kb/s`)
- **captcha_true** (default: `Captcha waiting`)
- **captcha_false** (default: *empty*)
- **download_true** (default: `Downloads enabled`)
- **download_false** (default: `Downloads disabled`)
- **username** (required)
- **password** (required)
- **interval** (default: `5`)

class i3pystatus.reddit.**Reddit**

This module fetches and displays posts and/or user mail/messages from reddit.com. Left-clicking on the display text opens the permalink/comments page using webbrowser.open() while right-clicking opens the URL of the submission directly. Depends on the Python Reddit API Wrapper (PRAW) <https://github.com/praw-dev/praw>.

Available formatters

- {submission_title}
- {submission_author}
- {submission_points}
- {submission_comments}
- {submission_permalink}
- {submission_url}
- {submission_domain}
- {submission_subreddit}
- {message_unread}
- {message_author}
- {message_subject}
- {message_body}

Settings

- **format** (default: `[{submission_subreddit}] {submission_title} ({submission_domain})`) – Format string used for output.
- **username** (default: *empty*) – Reddit username.
- **password** (default: *empty*) – Reddit password.
- **subreddit** (default: *empty*) – Subreddit to monitor. Uses frontpage if unspecified.
- **sort_by** (default: `hot`) – 'hot', 'new', 'rising', 'controversial', or 'top'.
- **color** (default: `#FFFFFF`) – Standard color.
- **colorize** (default: `True`) – Enable color change on new message.
- **color_orangered** (default: `#FF4500`) – Color for new messages.
- **mail_brackets** (default: `False`) – Display unread message count in square-brackets.
- **title_maxlen** (default: `80`) – Maximum number of characters to display in title.
- **interval** (default: `300`) – Update interval.
- **status** (default: `{'no_mail': '', 'new_mail': ''}`) – New message indicator.

class i3pystatus.regex.**Regex**

Simple regex file watcher

The groups of the regex are passed to the format string as positional arguments.

Settings

- **format** (default: `{0}`) – format string used for output
- **regex** (required)
- **file** – file to search for regex matches
- **flags** (default: `0`) – Python.re flags
- **interval** (default: `5`)

class `i3pystatus.runwatch.`**RunWatch**

Expands the given path using glob to a pidfile and checks if the process ID found inside is valid (that is, if the process is running). You can use this to check if a specific application, such as a VPN client or your DHCP client is running.

Available formatters are {pid} and {name}.

Settings

- **format_up** (default: `{name}`)
- **format_down** (default: `{name}`)
- **color_up** (default: `#00FF00`)
- **color_down** (default: `#FF0000`)
- **path** (required)
- **name** (required)
- **interval** (default: `5`)

class `i3pystatus.shell.`**Shell**

Shows output of shell command

Settings

- **command** – command to be executed
- **color** (default: `#FFFFFF`) – standard color
- **error_color** (default: `#FF0000`) – color to use when non zero exit code is returned
- **interval** (default: `5`)

class `i3pystatus.spotify.`**Spotify**

This class shows information from Spotify.

Left click will toggle pause/play of the current song. Right click will skip the song.

Dependent on Playerctl (https://github.com/acrisci/playerctl) and GLib

Settings

- **format** (default: `{artist} - {title}`) – Format string. {artist}, {title}, {album}, {volume}, and {length} are available for output.
- **color** (default: `#ffffff`) – color of the output

main_loop()
Mainloop blocks so we thread it.

class i3pystatus.temp.**Temperature**
Shows CPU temperature of Intel processors

AMD is currently not supported as they can only report a relative temperature, which is pretty useless

Settings

- **format** (default: `{temp} °C`) – format string used for output. {temp} is the temperature in degrees celsius
- **color** (default: `#FFFFFF`)
- **file** (default: `/sys/class/thermal/thermal_zone0/temp`)
- **interval** (default: `5`)

class i3pystatus.text.**Text**
Display static, colored text.

Settings

- **text** (required)
- **color** (default: *empty*) – HTML color code #RRGGBB
- **cmd_leftclick** (default: `test`) – Shell command to execute on left click
- **cmd_rightclick** (default: `test`) – Shell command to execute on right click

class i3pystatus.uptime.**Uptime**
Outputs Uptime

Settings

- **format** (default: `up {uptime}`) – Format string
- **color** (default: `#ffffff`) – String color
- **alert** (default: `False`) – If you want the string to change color
- **seconds_alert** (default: `3600`) – How many seconds necessary to start the alert
- **color_alert** (default: `#ff0000`) – Alert color
- **interval** (default: `5`)

class i3pystatus.weather.**Weather**

This module gets the weather from weather.com using pywapi module First, you need to get the code for the location from the www.weather.com .. rubric:: Available formatters

- {current_temp}
- {current_wind}
- {humidity}

Requires pywapi from PyPI.

Settings

- **location_code** (required)
- **colorize** (default: `False`) – Enable color with temperature and UTF-8 icons.
- **units** (default: `metric`) – Celsius (metric) or Fahrenheit (imperial)
- **format** (default: `{current_temp}`)
- **interval** (default: `20`)

class i3pystatus.wireless.**Wireless**

Display network information about a interface.

Requires the PyPI packages *netifaces* and *basiciw*.

This is based on the network module, so all options and formatters are the same, except for these additional formatters and that detached_down doesn't work.

- *{essid}* — ESSID of currently connected wifi
- *{freq}* — Current frequency
- *{quality}* — Link quality in percent
- *{quality_bar}* —Bar graphically representing link quality

Settings

- **interface** (default: `wlan0`) – Interface to obtain information for
- **format_up** (default: `{interface}: {v4}`)
- **color_up** (default: `#00FF00`)
- **format_down** (default: `{interface}`)
- **color_down** (default: `#FF0000`)
- **detached_down** (default: `True`) – If the interface doesn't exist, display it as if it were down
- **unknown_up** (default: `False`) – If the interface is in unknown state, display it as if it were up
- **name** (default: `eth0`)
- **interval** (default: `5`)

3.1 Mail Backends

class i3pystatus.imap.**IMAP**
Checks for mail on a IMAP server

Settings

- **host** (required)
- **port** (default: 993)
- **username** (required)
- **password** (required)
- **ssl** (default: True)
- **mailbox** (default: INBOX)

imap_class
alias of IMAP4

class i3pystatus.maildir.**MaildirMail**
Checks for local mail in Maildir

Settings

- **directory** (required)

class i3pystatus.mbox.**MboxMail**
Checks for local mail in mbox

class i3pystatus.notmuchmail.**Notmuch**
This class uses the notmuch python bindings to check for the number of messages in the notmuch database with the tags “inbox” and “unread”

Settings

- **db_path** (default: *empty*) – Path to the directory of your notmuch database

class i3pystatus.thunderbird.**Thunderbird**
This class listens for dbus signals emitted by the dbus-sender extension for thunderbird.

Requires python-dbus

CHAPTER 4

core Package

4.1 core Package

class i3pystatus.core.**CommandEndpoint**(*modules*, *io_handler_factory*)
Bases: object

Endpoint for i3bar click events: http://i3wm.org/docs/i3bar-protocol.html#_click_events

Parameters

- **modules** – dict-like object with item access semantics via .get()
- **io_handler_factory** – function creating a file-like object returning a JSON generator on .read()

start()
Starts the background thread

class i3pystatus.core.**Status**(*standalone=False*, *interval=1*, *input_stream=<_io.TextIOWrapper name='<stdin>' mode='r' encoding='ANSI_X3.4-1968'>*, *click_events=True*)
Bases: object

The main class used for registering modules and managing I/O

Parameters

- **standalone** – Whether i3pystatus should read i3status-compatible input from *input_stream*
- **interval** – Update interval in seconds
- **input_stream** – A file-like object that provides the input stream, if *standalone* is False.
- **click_events** – Enable click events

register(*module*, **args*, ***kwargs*)
Register a new module.

run()

4.2 desktop Module

class i3pystatus.core.desktop.**BaseDesktopNotification**(*title*, *body*, *icon='dialog-information'*, *urgency=1*, *timeout=0*)

Bases: object

Class to display a desktop notification

Parameters

- **title** – Title of the notification
- **body** – Body text of the notification, depending on the users system configuration HTML may be used, but is not recommended
- **icon** – A XDG icon name, see http://standards.freedesktop.org/icon-naming-spec/icon-naming-spec-latest.html
- **urgency** – A value between 1 and 3 with 1 meaning low urgency and 3 high urgency.
- **timeout** – Timeout in seconds for the notification. Zero means it needs to be dismissed by the user.

display()

Display this notification

Returns boolean indicating success

class i3pystatus.core.desktop.**DesktopNotification**(*title*, *body*, *icon='dialog-information'*, *urgency=1*, *timeout=0*)

Bases: *i3pystatus.core.desktop.DesktopNotification*

URGENCY_LUT = (<Mock name='mock.Notify.Urgency.LOW' id='140601494932504'>, <Mock name='mock.Notify.Urgen

display()

4.3 exceptions Module

exception i3pystatus.core.exceptions.**ConfigAmbigiousClassesError**(*module*, **args*, ***kwargs*)

Bases: *i3pystatus.core.exceptions.ConfigError*

format(*ambigious_classes*)

exception i3pystatus.core.exceptions.**ConfigError**(*module*, **args*, ***kwargs*)

Bases: Exception

ABC for configuration exceptions

format(**args*, ***kwargs*)

exception i3pystatus.core.exceptions.**ConfigInvalidModuleError**(*module*, **args*, ***kwargs*)

Bases: *i3pystatus.core.exceptions.ConfigError*

format()

exception i3pystatus.core.exceptions.**ConfigKeyError**(*module*, **args*, ***kwargs*)

Bases: *i3pystatus.core.exceptions.ConfigError*, KeyError

format(*key*)

exception i3pystatus.core.exceptions.**ConfigMissingError**(*module*, **args*, ***kwargs*)
Bases: *i3pystatus.core.exceptions.ConfigError*

format(*missing*)

4.4 imputil Module

class i3pystatus.core.imputil.**ClassFinder**(*baseclass*)
Bases: object

Support class to find classes of specific bases in a module

get_class(*module*)

get_matching_classes(*module*)

get_module(*module*)

instanciate_class_from_module(*module*, **args*, ***kwargs*)

predicate_factory(*module*)

4.5 io Module

class i3pystatus.core.io.**IOHandler**(*inp=<_io.TextIOWrapper name='<stdin>' mode='r' encoding='ANSI_X3.4-1968'>*, *out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='ANSI_X3.4-1968'>*)
Bases: object

read()
Iterate over all input lines (Generator)

read_line()
Interrupted respecting reader for stdin.

Raises EOFError if the end of stream has been reached

write_line(*message*)
Unbuffered printing to stdout.

class i3pystatus.core.io.**JSONIO**(*io*, *skiplines=2*)
Bases: object

parse_line(*line*)
Parse a single line of JSON and write modified JSON back.

read()
Iterate over all JSON input (Generator)

class i3pystatus.core.io.**StandaloneIO**(*click_events*, *interval=1*)
Bases: *i3pystatus.core.io.IOHandler*

I/O handler for standalone usage of i3pystatus (w/o i3status)

Writing works as usual, but reading will always return a empty JSON array, and the i3bar protocol header

n = -1

proto = [{'version': 1, 'click_events': True}, '[', '[]', ',[]']

read()

read_line()

4.6 modules Module

class i3pystatus.core.modules.**IntervalModule**(**args*, ***kwargs*)
Bases: *i3pystatus.core.modules.Module*

interval = 5

managers = {}

registered(*status_handler*)

run()
Called approximately every self.interval seconds

Do not rely on this being called from the same thread at all times. If you need to always have the same thread context, subclass AsyncModule.

settings = ('interval',)

class i3pystatus.core.modules.**IntervalModuleMeta**(*name*, *bases*, *namespace*)
Bases: type

Add interval setting to *settings* attribute if it does not exist.

class i3pystatus.core.modules.**Module**(**args*, ***kwargs*)
Bases: *i3pystatus.core.settings.SettingsBase*

inject(*json*)

move(*position*)

on_click(*button*)

on_downscroll()

on_leftclick()

on_rightclick()

on_upscroll()

output = None

position = 0

registered(*status_handler*)
Called when this module is registered with a status handler

run()

4.7 settings Module

class i3pystatus.core.settings.**SettingsBase**(**args*, ***kwargs*)
Bases: object

Support class for providing a nice and flexible settings interface

Classes inherit from this class and define what settings they provide and which are required.

The constructor is either passed a dictionary containing these settings, or keyword arguments specifying the same.

Settings are stored as attributes of self.

static flatten_settings(*settings*)

init()
Convenience method which is called after all settings are set

In case you don't want to type that super()... blabla :-)

required = ()
required can list settings which are required

settings = ()
settings should be tuple containing two types of elements:

- bare strings, which must be valid Python identifiers.
- two-tuples, the first element being a identifier (as above) and the second a docstring for the particular setting

4.8 threading Module

class i3pystatus.core.threading.**ExceptionWrapper**(*workload*)
Bases: *i3pystatus.core.threading.Wrapper*

class i3pystatus.core.threading.**Manager**(*target_interval*)
Bases: object

append(*workload*)

create_thread(*workloads*)

create_threads(*threads*)

partition_workloads(*workloads*)

start()

wrap(*workload*)

class i3pystatus.core.threading.**Thread**(*target_interval*, *workloads=None*, *start_barrier=1*)
Bases: threading.Thread

append(*workload*)

branch(*vtime*, *bound*)

execute_workloads()

pop()

run()

time

wait_for_start_barrier()

class i3pystatus.core.threading.**WorkloadWrapper**(*workload*)
Bases: *i3pystatus.core.threading.Wrapper*

time = 0.0

class i3pystatus.core.threading.**Wrapper**(*workload*)
Bases: object

4.9 util Module

class i3pystatus.core.util.**KeyConstraintDict**(*valid_keys*, *required_keys*)
Bases: collections.UserDict

A dict implementation with sets of valid and required keys

Parameters

- **valid_keys** – Set of valid keys
- **required_keys** – Set of required keys, must be a subset of valid_keys

exception MissingKeys(*keys*)
Bases: Exception

KeyConstraintDict.**missing**()
Returns a set of keys that are required but not set

class i3pystatus.core.util.**ModuleList**(*status_handler*, *class_finder*)
Bases: collections.UserList

append(*module*, **args*, ***kwargs*)

get(*find_id*)

class i3pystatus.core.util.**TimeWrapper**(*seconds*, *default_format='%m:%S'*)
Bases: object

A wrapper that implements __format__ and __bool__ for time differences and time spans.

Parameters

- **seconds** – seconds (numeric)
- **default_format** – the default format to be used if no explicit format_spec is passed to __format__

Format string syntax:

- %h, %m and %s are the hours, minutes and seconds without leading zeros (i.e. 0 to 59 for minutes and seconds)
- %H, %M and %S are padded with a leading zero to two digits, i.e. 00 to 59
- %l and %L produce hours non-padded and padded but only if hours is not zero. If the hours are zero it produces an empty string.
- %% produces a literal %
- %E (only valid on beginning of the string) if the time is null, don't format anything but rather produce an empty string. If the time is non-null it is removed from the string.

The formatted string is stripped, i.e. spaces on both ends of the result are removed

class TimeTemplate(*template*)
Bases: string.Template

delimiter = '%'

idpattern = '[a-zA-Z]'

pattern = re.compile('\n \\%(?:\n (?P<escaped>\\%) | # Escape sequence of two delimiters\n (?P<named>[a-zA-Z])

i3pystatus.core.util.**convert_position**(*pos*, *json*)

i3pystatus.core.util.**flatten**(*l*)

Flattens a hierarchy of nested lists into a single list containing all elements in order

Parameters **l** – list of arbitrary types and lists

Returns list of arbitrary types

i3pystatus.core.util.**formatp**(*string*, ***kwargs*)

Function for advanced format strings with partial formatting

This function consumes format strings with groups enclosed in brackets. A group enclosed in brackets will only become part of the result if all fields inside the group evaluate True in boolean contexts.

Groups can be nested. The fields in a nested group do not count as fields in the enclosing group, i.e. the enclosing group will evaluate to an empty string even if a nested group would be eligible for formatting. Nesting is thus equivalent to a logical or of all enclosing groups with the enclosed group.

Escaped brackets, i.e. \[and \] are copied verbatim to output.

Parameters

- **string** – Format string
- **kwargs** – keyword arguments providing data for the format string

Returns Formatted string

i3pystatus.core.util.**internet**()

Checks for a internet connection by connecting to a Google DNS server.

Returns True if internet connection is available

i3pystatus.core.util.**lchop**(*string*, *prefix*)

Removes a prefix from string

Parameters

- **string** – String, possibly prefixed with prefix
- **prefix** – Prefix to remove from string

Returns string without the prefix

i3pystatus.core.util.**make_bar**(*percentage*)

Draws a bar made of unicode box characters.

Parameters **percentage** – A value between 0 and 100

Returns Bar as a string

i3pystatus.core.util.**make_graph**(*values*, *upper_limit=100.0*)

Draws a graph made of unicode characters.

Parameters

- **values** – An array of values to graph.
- **upper_limit** – Maximum value for the y axis.

Returns Bar as a string

i3pystatus.core.util.**make_vertical_bar**(*percentage*, *width=1*)

Draws a vertical bar made of unicode characters.

Parameters

- **value** – A value between 0 and 100
- **width** – How many characters wide the bar should be.

Returns Bar as a String

i3pystatus.core.util.**partition**(*iterable*, *limit*, *key=<function <lambda>>*)

i3pystatus.core.util.**popwhile**(*predicate*, *iterable*)
Generator function yielding items of iterable while predicate holds for each item

Parameters

- **predicate** – function taking an item returning bool
- **iterable** – iterable

Returns iterable (generator function)

i3pystatus.core.util.**require**(*predicate*)
Decorator factory for methods requiring a predicate. If the predicate is not fulfilled during a method call, the method call is skipped and None is returned.

Parameters **predicate** – A callable returning a truth value

Returns Method decorator

See also:

internet()

i3pystatus.core.util.**round_dict**(*dic*, *places*)
Rounds all values in a dict containing only numeric types to *places* decimal places. If places is None, round to INT.

i3pystatus.core.util.**user_open**(*url_or_command*)
Open the specified paramater in the web browser if a URL is detected, othewrise pass the paramater to the shell as a subprocess. This function is inteded to bu used in on_leftclick()/on_rightclick() events.

Parameters **url_or_command** – String containing URL or command

4.10 color Module

class i3pystatus.core.color.**ColorRangeModule**
Bases: object

Class to dynamically generate and select colors.

Requires the PyPI package *colour*

end_color = 'red'

get_gradient(*value*, *colors*, *upper_limit=100*)
Map a value to a color :param value: Some value :return: A Hex color code

static get_hex_color_range(*start_color*, *end_color*, *quantity*)
Generates a list of quantity Hex colors from start_color to end_color.

Parameters

- **start_color** – Hex or plain English color for start of range
- **end_color** – Hex or plain English color for end of range

- **quantity** – Number of colours to return

Returns A list of Hex color values

static percentage(*part*, *whole*)
Calculate percentage

start_color = '#00FF00'

CHAPTER 5

Creating modules

Creating new modules ("things that display something") to contribute to i3pystatus is reasonably easy. If the module you want to write updates it's info periodically, like checking for a network link or displaying the status of some service, then we have prepared common tools for this which make this even easier:

- Common base classes: *i3pystatus.core.modules.Module* for everything and *i3pystatus.core.modules.IntervalModule* specifically for the aforementioned usecase of updating stuff periodically.
- Settings (already built into above classes) allow you to easily specify user-modifiable attributes of your class for configuration.

  Required settings and default values are also handled.

Check out i3pystatus' source code for plenty of (simple) examples on how to build modules.

Also note that the settings system is built to ease documentation. If you specify two-tuples `("setting", "description")` description then `i3pystatus.mkdocs` will automatically generate a nice table listing each option, it's default value and description.

The docstring of your module class is automatically used as the restructuredtext description for your module in the README file.

See also:

i3pystatus.core.settings.SettingsBase for a detailed description of the settings system

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

i

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

W