NAME

i3status − Generates a status line for i3bar, dzen2, xmobar or lemonbar

SYNOPSIS

i3status [−c configfile] [−h] [−v]

OPTIONS

−c

Specifies an alternate configuration file path. By default, i3status looks for configuration files in the following order:

1. ~/.config/i3status/config (or $XDG_CONFIG_HOME/i3status/config if set)
2. /etc/xdg/i3status/config (or $XDG_CONFIG_DIRS/i3status/config if set)
3. ~/.i3status.conf
4. /etc/i3status.conf

DESCRIPTION

i3status is a small program for generating a status bar for i3bar, dzen2, xmobar, lemonbar or similar programs. It is designed to be very efficient by issuing a very small number of system calls, as one generally wants to update such a status line every second. This ensures that even under high load, your status bar is updated correctly. Also, it saves a bit of energy by not hogging your CPU as much as spawning the corresponding amount of shell commands would.

CONFIGURATION

The basic idea of i3status is that you can specify which "modules" should be used (the order directive). You can then configure each module with its own section. For every module, you can specify the output format. See below for a complete reference.

Sample configuration.

```
general {
        output_format = "dzen2"
        colors = true
        interval = 5
}

order += "ipv6"
order += "disk /"
order += "run_watch DHCP"
order += "run_watch VPNC"
order += "path_exists VPN"
order += "wireless wlan0"
order += "ethernet eth0"
order += "battery 0"
order += "cpu_temperature 0"
order += "memory"
order += "load"
order += "tztime local"
order += "tztime berlin"

wireless wlan0 {
        format_up = "W: (%quality at %essid, %bitrate) %ip"
        format_down = "W: down"
}

ethernet eth0 {
        format_up = "E: %ip (%speed)"
```

```
        format_down = "E: down"
}

battery 0 {
        format = "%status %percentage %remaining %emptytime"
        format_down = "No battery"
        status_chr = " CHR"
        status_bat = " BAT"
        status_unk = "? UNK"
        status_full = " FULL"
        path = "/sys/class/power_supply/BAT%d/uevent"
        low_threshold = 10
}

run_watch DHCP {
        pidfile = "/var/run/dhclient*.pid"
}

run_watch VPNC {
        # file containing the PID of a vpnc process
        pidfile = "/var/run/vpnc/pid"
}

path_exists VPN {
        # path exists when a VPN tunnel launched by nmcli/nm-applet is active
        path = "/proc/sys/net/ipv4/conf/tun0"
}

tztime local {
        format = "%Y-%m-%d %H:%M:%S"
        hide_if_equals_localtime = true
}

tztime berlin {
        format = "%Y-%m-%d %H:%M:%S %Z"
        timezone = "Europe/Berlin"
}

load {
        format = "%5min"
}

cpu_temperature 0 {
        format = "T: %degrees °C"
        path = "/sys/devices/platform/coretemp.0/temp1_input"
}

memory {
        format = "%used"
        threshold_degraded = "10%"
        format_degraded = "MEMORY: %free"
}

disk "/" {
```

```
        format = "%free"
}

read_file uptime {
        path = "/proc/uptime"
}
```

General

The colors directive will disable all colors if you set it to false. You can also specify the colors that will be used to display "good", "degraded" or "bad" values using the color_good, color_degraded or color_bad directives, respectively. Those directives are only used if color support is not disabled by the colors directive. The input format for color values is the canonical RGB hexadecimal triplet (with no separators between the colors), prefixed by a hash character ("#").

Example configuration:

```
color_good = "#00FF00"
```

Likewise, you can use the color_separator directive to specify the color that will be used to paint the separator bar. The separator is always output in color, even when colors are disabled by the colors directive. This option has no effect when output_format is set to i3bar or none.

The interval directive specifies the time in seconds for which i3status will sleep before printing the next status line.

Using output_format you can choose which format strings i3status should use in its output. Currently available are:

i3bar
: i3bar comes with i3 and provides a workspace bar which does the right thing in multi−monitor situations. It also comes with tray support and can display the i3status output. This output type uses JSON to pass as much meta−information to i3bar as possible (like colors, which blocks can be shortened in which way, etc.).

dzen2
: Dzen is a general purpose messaging, notification and menuing program for X11. It was designed to be scriptable in any language and integrate well with window managers like dwm, wmii and xmonad though it will work with any window manager

xmobar
: xmobar is a minimalistic, text based, status bar. It was designed to work with the xmonad Window Manager.

lemonbar
: lemonbar is a lightweight bar based entirely on XCB. It has full UTF−8 support and is EWMH compliant.

term
: Use ANSI Escape sequences to produce a terminal−output as close as possible to the graphical outputs. This makes debugging your config file a little bit easier because the terminal−output of i3status becomes much more readable, but should only used for such quick glances, because it will only support very basic output−features (for example you only get 3 bits of color depth).

none
: Does not use any color codes. Separates values by the pipe symbol by default. This should be used with i3bar and can be used for custom scripts.

It's also possible to use the color_good, color_degraded, color_bad directives to define specific colors per

module. If one of these directives is defined in a module section its value will override the value defined in the general section just for this module.

If you don't fancy the vertical separators between modules i3status/i3bar uses by default, you can employ the separator directive to configure how modules are separated. You can also disable the default separator altogether by setting it to the empty string. You might then define separation as part of a module's format string. This is your only option when using the i3bar output format as the separator is drawn by i3bar directly otherwise. For the other output formats, the provided non−empty string will be automatically enclosed with the necessary coloring bits if color support is enabled.

i3bar supports Pango markup, allowing your format strings to specify font, color, size, etc. by setting the markup directive to "pango". Note that the ampersand ("&"), less−than ("<"), greater−than (">"), single−quote ("'"), and double−quote ("""") characters need to be replaced with "&", "<", ">", "'", and """ respectively. This is done automatically for generated content (e.g. wireless ESSID, time).

Example configuration:

```
general {
   output_format = "xmobar"
   separator = "  "
}

order += "load"
order += "disk /"

load {
   format = "[ load: %1min, %5min, %15min ]"
}
disk "/" {
   format = "%avail"
}
```

IPv6

This module gets the IPv6 address used for outgoing connections (that is, the best available public IPv6 address on your computer).

Example format_up: %ip

Example format_down: no IPv6

Disk

Gets used, free, available and total amount of bytes on the given mounted filesystem.

These values can also be expressed in percentages with the percentage_used, percentage_free, percentage_avail and percentage_used_of_avail formats.

Byte sizes are presented in a human readable format using a set of prefixes whose type can be specified via the "prefix_type" option. Three sets of prefixes are available:

binary
: IEC prefixes (Ki, Mi, Gi, Ti) represent multiples of powers of 1024. This is the default.

decimal
: SI prefixes (k, M, G, T) represent multiples of powers of 1000.

custom

The custom prefixes (K, M, G, T) represent multiples of powers of 1024.

It is possible to define a low_threshold that causes the disk text to be displayed using color_bad. The low_threshold type can be of threshold_type "bytes_free", "bytes_avail", "percentage_free", or "percentage_avail", where the former two can be prepended by a generic prefix (k, m, g, t) having prefix_type. So, if you configure low_threshold to 2, threshold_type to "gbytes_avail", and prefix_type to "binary", and the remaining available disk space is below 2 GiB, it will be colored bad. If not specified, threshold_type is assumed to be "percentage_avail" and low_threshold to be set to 0, which implies no coloring at all. You can customize the output format when below low_threshold with format_below_threshold.

You can define a different format with the option "format_not_mounted" which is used if the path does not exist or is not a mount point. Defaults to "".

Example order: disk /mnt/usbstick

Example format: %free (%avail)/ %total

Example format: %percentage_used used, %percentage_free free, %percentage_avail avail

Example prefix_type: custom

Example low_threshold: 5

Example format_below_threshold: Warning: %percentage_avail

Example threshold_type: percentage_free

Run−watch

Expands the given path 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. There also is an option "format_down". You can hide the output with format_down="".

Example order: run_watch DHCP

Example format: %title: %status

Path−exists

Checks if the given path exists in the filesystem. You can use this to check if something is active, like for example a VPN tunnel managed by NetworkManager. There also is an option "format_down". You can hide the output with format_down="".

Example order: path_exists VPN

Example format: %title: %status

Wireless

Gets the link quality, frequency and ESSID of the given wireless network interface. You can specify different format strings for the network being connected or not connected. The quality is padded with leading zeroes by default; to pad with something else use format_quality.

The special interface name _first_ will be replaced by the first wireless network interface found on the system (excluding devices starting with "lo").

Example order: wireless wlan0

Example format_up: W: (%quality at %essid, %bitrate / %frequency) %ip

Example format_down: W: down

Example format_quality: "%03d%s"

Ethernet

Gets the IP address and (if possible) the link speed of the given ethernet interface. If no IPv4 address is available and an IPv6 address is, it will be displayed.

The special interface name _first_ will be replaced by the first non−wireless network interface found on the system (excluding devices starting with "lo").

Example order: ethernet eth0

Example format_up: E: %ip (%speed)

Example format_down: E: down

Battery

Gets the status (charging, discharging, unknown, full), percentage, remaining time and power consumption (in Watts) of the given battery and when it's estimated to be empty. If you want to use the last full capacity instead of the design capacity (when using the design capacity, it may happen that your battery is at 23% when fully charged because it's old. In general, I want to see it this way, because it tells me how worn off my battery is.), just specify last_full_capacity = true. You can show seconds in the remaining time and empty time estimations by setting hide_seconds = false.

If you want the battery percentage to be shown without decimals, add integer_battery_capacity = true.

If your battery is represented in a non−standard path in /sys, be sure to modify the "path" property accordingly, i.e. pointing to the uevent file on your system. The first occurrence of %d gets replaced with the battery number, but you can just hard−code a path as well.

It is possible to define a low_threshold that causes the battery text to be colored red. The low_threshold type can be of threshold_type "time" or "percentage". So, if you configure low_threshold to 10 and threshold_type to "time", and your battery lasts another 9 minutes, it will be colored red.

To show an aggregate of all batteries in the system, use "all" as the number. In this case (for Linux), the /sys path must contain the "%d" sequence. Otherwise, the number indicates the battery index as reported in /sys.

Optionally custom strings including any UTF−8 symbols can be used for different battery states. This makes it possible to display individual symbols for each state (charging, discharging, unknown, full) Of course it will also work with special iconic fonts, such as FontAwesome. If any of these special status strings are omitted, the default (CHR, BAT, UNK, FULL) is used.

Example order (for the first battery): battery 0

Example order (aggregate of all batteries): battery all

Example format: %status %remaining (%emptytime %consumption)

Example format_down: No battery

Example status_chr: CHR

Example status_bat: BAT

Example status_unk: ? UNK

Example status_full: FULL

Example low_threshold: 30

Example threshold_type: time

Example path (%d replaced by title number): /sys/class/power_supply/CMB%d/uevent

Example path (ignoring the number): /sys/class/power_supply/CMB1/uevent

CPU−Temperature

Gets the temperature of the given thermal zone. It is possible to define a max_threshold that will color the temperature red in case the specified thermal zone is getting too hot. Defaults to 75 degrees C. The output format when above max_threshold can be customized with format_above_threshold.

Example order: cpu_temperature 0

Example format: T: %degrees °C

Example max_threshold: 42

Example format_above_threshold: Warning T above threshold: %degrees °C

Example path: /sys/devices/platform/coretemp.0/temp1_input

CPU Usage

Gets the percentual CPU usage from /proc/stat (Linux) or sysctl(3) (FreeBSD/OpenBSD).

It is possible to define a max_threshold that will color the load value red in case the CPU average over the last interval is getting higher than the configured threshold. Defaults to 95. The output format when above max_threshold can be customized with format_above_threshold.

It is possible to define a degraded_threshold that will color the load value yellow in case the CPU average over the last interval is getting higher than the configured threshold. Defaults to 90. The output format when above degraded threshold can be customized with format_above_degraded_threshold.

For displaying the Nth CPU usage, you can use the %cpu<N> format string, starting from %cpu0. This feature is currently not supported in FreeBSD.

Example order: cpu_usage

Example format: all: %usage CPU_0: %cpu0 CPU_1: %cpu1

Example max_threshold: 75

Example format_above_threshold: Warning above threshold: %usage

Example degraded_threshold: 25

Example format_above_degraded_threshold: Warning above degraded threshold: %usage

Memory

Gets the memory usage from system on a Linux system from /proc/meminfo. Other systems are currently not supported.

As format placeholders, total, used, free, available and shared are available. These will print human readable values. It's also possible to prefix the placeholders with percentage_ to get a value in percent.

It's possible to define a threshold_degraded and a threshold_critical to color the status bar output in yellow or red, if the available memory falls below the given threshold. Possible values of the threshold can be any integer, suffixed with an iec symbol (T, G, M, K). Alternatively, the integer can be suffixed by a percent sign, which then rets evaluated relatively to total memory.

If the format_degraded parameter is given and either the critical or the degraded threshold applies, format_degraded will get used as format string. It acts equivalently to format.

As Linux' meminfo doesn't expose the overall memory in use, there are multiple methods to distinguish the actually used memory.

Example memory_used_method: memavailable ("total memory" − "MemAvailable", matches free command)

Example memory_used_method: classical ("total memory" − "free" − "buffers" − "cache", matches gnome system monitor)

Example order: memory

Example format: %free %available (%used) / %total

Example format: %percentage_used used, %percentage_free free, %percentage_shared shared

Example threshold_degraded: 10%

Example threshold_critical: 5%

Example format_degraded: Memory LOW: %free

Load

Gets the system load (number of processes waiting for CPU time in the last 1, 5 and 15 minutes). It is possible to define a max_threshold that will color the load value red in case the load average of the last minute is getting higher than the configured threshold. Defaults to 5. The output format when above max_threshold can be customized with format_above_threshold.

Example order: load

Example format: %1min %5min %15min

Example max_threshold: "0.1"

Example format_above_threshold: Warning: %1min %5min %15min

Time

Outputs the current time in the local timezone. To use a different timezone, you can set the TZ environment variable, or use the tztime module. See strftime(3) for details on the format string.

Example order: time

Example format: %Y−%m−%d %H:%M:%S

TzTime

Outputs the current time in the given timezone. If no timezone is given, local time will be used. See strftime(3) for details on the format string. The system's timezone database is usually installed in /usr/share/zoneinfo. Files below that path make for valid timezone strings, e.g. for /usr/share/zoneinfo/Europe/Berlin you can set timezone to Europe/Berlin in the tztime module. To override the locale settings of your environment, set the locale option. To display time only when the set timezone has different time from localtime, set hide_if_equals_localtime to true.

Example order: tztime berlin

Example format: %Y−%m−%d %H:%M:%S %Z

Example timezone: Europe/Berlin

Example locale: de_DE.UTF−8

If you would like to use markup in this section, there is a separate format_time option that is automatically escaped. Its output then replaces %time in the format string.

Example configuration (markup):

```
tztime berlin {
    format = "<span foreground='#ffffff'>time:</span> %time"
    format_time = "%H:%M %Z"
    timezone = "Europe/Berlin"
    hide_if_equals_localtime = true
}
```

DDate

Outputs the current discordian date in user−specified format. See ddate(1) for details on the format string. **Note**: Neither **%.** nor **%X** are implemented yet.

Example order: ddate

Example format: %{%a, %b %d%}, %Y%N − %H

Volume

Outputs the volume of the specified mixer on the specified device. PulseAudio and ALSA (Linux only) are supported. If PulseAudio is absent, a simplified configuration can be used on FreeBSD and OpenBSD due to the lack of ALSA, the device and mixer options can be ignored on these systems. On these systems the OSS API is used instead to query /dev/mixer directly if mixer_idx is −1, otherwise /dev/mixer+mixer_idx+.

To get PulseAudio volume information, one must use the following format in the device line:

device = "pulse"

or

device = "pulse:N"

where N is the index or name of the PulseAudio sink. You can obtain the name of the sink with the following command:

```
$ pacmd list−sinks | grep name:
        name: <alsa_output.pci−0000_00_14.2.analog−stereo>
```

The name is what's inside the angle brackets, not including them. If no sink is specified the default sink is used. If the device string is missing or is set to "default", PulseAudio will be tried if detected and will fallback to ALSA (Linux) or OSS (FreeBSD/OpenBSD).

Example order: volume master

Example format: (%devicename): %volume

Example format_muted: (%devicename): 0%%

Example configuration:

```
volume master {
        format = ": %volume"
        format_muted = ": muted (%volume)"
        device = "default"
        mixer = "Master"
        mixer_idx = 0
}
```

Example configuration (PulseAudio):

```
volume master {
        format = ": %volume"
        format_muted = ": muted (%volume)"
        device = "pulse:1"
}

volume master {
        format = ": %volume"
        format_muted = ": muted (%volume)"
        device = "pulse:alsa_output.pci−0000_00_14.2.analog−stereo"
}
```

File Contents

Outputs the contents of the specified file. You can use this to check contents of files on your system, for example /proc/uptime. By default the function only reads the first 254 characters of the file, if you want to override this set the Max_characters option. It will never read beyond the first 4095 characters. If the file is not found "no file" will be printed, if the file can't be read "error read" will be printed.

Example order: read_file UPTIME

Example format: "%title: %content"

Example format_bad: "%title − %errno: %error"

Example path: "/proc/uptime"

Example Max_characters: 255

UNIVERSAL MODULE OPTIONS

When using the i3bar output format, there are a few additional options that can be used with all modules to customize their appearance:

align
: The alignment policy to use when the minimum width (see below) is not reached. Either center

(default), right or left.

min_width
: The minimum width (in pixels) the module should occupy. If the module takes less space than the specified size, the block will be padded to the left and/or the right side, according to the defined alignment policy. This is useful when you want to prevent the whole status line from shifting when values take more or less space between each iteration. The option can also be a string. In this case, the width of the given text determines the minimum width of the block. This is useful when you want to set a sensible minimum width regardless of which font you are using, and at what particular size. Please note that a number enclosed with quotes will still be treated as a number.

separator
: A boolean value which specifies whether a separator line should be drawn after this block. The default is true, meaning the separator line will be drawn. Note that if you disable the separator line, there will still be a gap after the block, unless you also use separator_block_width.

separator_block_width
: The amount of pixels to leave blank after the block. In the middle of this gap, a separator symbol will be drawn unless separator is disabled. This is why the specified width should leave enough space for the separator symbol.

Example configuration:

```
disk "/" {
   format = "%avail"
   align = "left"
   min_width = 100
   separator = false
   separator_block_width = 1
}
```

USING I3STATUS WITH DZEN2

After installing dzen2, you can directly use it with i3status. Just ensure that output_format is set to dzen2. **Note**: min_width is not supported.

Example for usage of i3status with dzen2:

```
i3status | dzen2 -fg white -ta r -w 1280 \
-fn "-misc-fixed-medium-r-normal--13-120-75-75-C-70-iso8859-1"
```

USING I3STATUS WITH XMOBAR

To get xmobar to start, you might need to copy the default configuration file to ~/.xmobarrc. Also, ensure that the output_format option for i3status is set to xmobar. **Note**: min_width is not supported.

Example for usage of i3status with xmobar:

```
i3status | xmobar -o -t "%StdinReader%" -c "[Run StdinReader]"
```

WHAT ABOUT CPU FREQUENCY?

While talking about specific things, please understand this section as a general explanation why your favorite information is not included in i3status.

Let's talk about CPU frequency specifically. Many people don't understand how frequency scaling works precisely. The generally recommended CPU frequency governor ("ondemand") changes the CPU frequency far more often than i3status could display it. The display number is therefore often incorrect and doesn't tell you anything useful either.

In general, i3status wants to display things which you would look at occasionally anyways, like the current

date/time, whether you are connected to a WiFi network or not, and if you have enough disk space to fit that 4.3 GiB download.

However, if you need to look at some kind of information more than once in a while, you are probably better off with a script doing that, which pops up. After all, the point of computers is not to burden you with additional boring tasks like repeatedly checking a number.

EXTERNAL SCRIPTS/PROGRAMS WITH I3STATUS

In i3status, we don't want to implement process management again. Therefore, there is no module to run arbitrary scripts or commands. Instead, you should use your shell, for example like this:

Example for prepending the i3status output:

```
#!/bin/sh
# shell script to prepend i3status with more stuff

i3status | while :
do
        read line
        echo "mystuff | $line" || exit 1
done
```

Put that in some script, say .bin/my_i3status.sh and execute that instead of i3status.

Note that if you want to use the JSON output format (with colors in i3bar), you need to use a slightly more complex wrapper script. There are examples in the contrib/ folder, see
https://github.com/i3/i3status/tree/master/contrib

SIGNALS

When receiving SIGUSR1, i3status's nanosleep() will be interrupted and thus you will force an update. You can use killall −USR1 i3status to force an update after changing the system volume, for example.

SEE ALSO

strftime(3), date(1), glob(3), dzen2(1), xmobar(1)

AUTHORS

Michael Stapelberg and contributors

Thorsten Toepper

Baptiste Daroussin

Axel Wagner

Fernando Tarlá Cardoso Lemos