Vicious

Release 2.4.2

vicious-widgets

May 04, 2022

CONTENTS

Vicious is a modular widget library for window managers, but mostly catering to users of the awesome window manager. It was derived from the old *wicked* widget library, and has some of the old *wicked* widget types, a few of them rewritten, and a good number of new ones.

Vicious widget types are a framework for creating your own widgets. Vicious contains modules that gather data about your system, and a few *awesome* helper functions that make it easier to register timers, suspend widgets and so on. Vicious doesn't depend on any third party Lua library, but may depend on additional system utilities.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Usage as a Lua Library

When provided by an operating system package, or installed from source into the Lua library path, Vicious can be used as a regular Lua library, to be used stand-alone or to feed widgets of any window manager (e.g. Ion, WMII). It is compatible with Lua 5.1 and above.

```
> widgets = require("vicious.widgets.init")
> print(widgets.volume(nil, "Master")[1])
100
```

1.2 Usage within Awesome

To use Vicious with awesome, install the package from your operating system provider, or download the source code and move it to your awesome configuration directory in `$XDG_CONFIG_HOME` (usually `~/.config`):

```
git clone https://github.com/vicious-widgets/vicious.git
mv vicious $XDG_CONFIG_HOME/awesome/
```

Vicious will only load modules for widget types you intend to use in your awesome configuration, to avoid having useless modules sitting in your memory.

Then add the following to the top of your `rc.lua`:

```
local vicious = require("vicious")
```

1.2.1 vicious.register

Once you create a widget (a textbox, graph or a progressbar), call `vicious.register` to register it with Vicious:

`vicious.`**`register`**(*widget*, *wtype*, *format*, *interval*, *warg*)

Register a widget.

Parameters

- **`widget`** – awesome widget created from `awful.widget` or `wibox.widget`
- **`wtype`** – either of
  - Vicious widget type: any widget type *provided by Vicious* or customly defined.

  - `function`: custom function from your own awesome configuration can be registered as widget types (see *Custom Widget Types*).

- **`format`** – either of
  - string: `$key` will be replaced by respective value in the table `t` returned by the widget type, i.e. use `$1`, `$2`, etc. to retrieve data from an integer-indexed table (a.k.a. array); `${foo bar}` will be substituted by `t["{foo bar}"]`.
  - `function (widget, args)` can be used to manipulate data returned by the widget type (see *Format Functions*).
- **`interval`** – number of seconds between updates of the widget (default: 2). See *Power and Caching* for more information.
- **`warg`** – arguments to be passed to widget types, e.g. the battery ID.

`vicious.register` alone is not much different from awful.widget.watch, which has been added to Awesome since version 4.0. However, Vicious offers more advanced control of widgets' behavior by providing the following functions.

1.2.2 vicious.unregister

`vicious.`**`unregister`**`(`*widget*, *keep*`)`

Unregister a widget.

Parameters

- **`widget`** – awesome widget created from `awful.widget` or `wibox.widget`
- **`keep`** (`bool`) – if true suspend `widget` and wait for activation

1.2.3 vicious.suspend

`vicious.`**`suspend`**`()`

Suspend all widgets.

See example automation script for the "laptop-mode-tools" start-stop module.

1.2.4 vicious.activate

`vicious.`**`activate`**`(`[*widget*]`)`

Restart suspended widget(s).

Parameters **`widget`** – if provided only that widget will be activated

1.2.5 vicious.cache

`vicious.`**`cache`**`(`*wtype*`)`

Enable caching of values returned by a widget type.

1.2.6 vicious.force

`vicious.force(wtable)`

Force update of given widgets.

Parameters `wtable` – table of one or more widgets to be updated

1.2.7 vicious.call[_async]

`vicious.call(wtype, format, warg)`

Get formatted data from a synchronous widget type (*example*).

Parameters

- `wtype` – either of
  - Vicious widget type: any synchronous widget type *provided by Vicious* or customly defined.
  - `function`: custom function from your own awesome configuration can be registered as widget types (see *Custom Widget Types*).
- `format` – either of
  - string: `$key` will be replaced by respective value in the table `t` returned by the widget type, i.e. use `$1`, `$2`, etc. to retrieve data from an integer-indexed table (a.k.a. array); `${foo bar}` will be substituted by `t["{foo bar}"]`.
  - `function (widget, args)` can be used to manipulate data returned by the widget type (see *Format Functions*).
- `warg` – arguments to be passed to the widget type, e.g. the battery ID.

Returns `nil` if the widget type is asynchronous, otherwise the formatted data from with widget type.

`vicious.call_async(wtype, format, warg, callback)`

Get formatted data from an asynchronous widget type.

Parameters

- `wtype` – any asynchronous widget type *provided by Vicious* or customly defined.
- `format` – either of
  - string: `$key` will be replaced by respective value in the table `t` returned by the widget type, i.e. use `$1`, `$2`, etc. to retrieve data from an integer-indexed table (a.k.a. array); `${foo bar}` will be substituted by `t["{foo bar}"]`.
  - `function (widget, args)` can be used to manipulate data returned by the widget type (see *Format Functions*).
- `warg` – arguments to be passed to the widget type.
- `callback` – function taking the formatted data from with widget type. If the given widget type happens to be synchronous, `nil` will be passed to `callback`.

1.3 Usage Examples

Start with a simple widget, like `date`, then build your setup from there, one widget at a time. Also remember that besides creating and registering widgets you have to add them to a `wibox` (statusbar) in order to actually display them.

1.3.1 Date Widget

Update every 2 seconds (the default interval), use standard date sequences as the format string:

```
datewidget = wibox.widget.textbox()
vicious.register(datewidget, vicious.widgets.date, "%b %d, %R")
```

1.3.2 Memory Widget

Update every 13 seconds, append `MiB` to 2nd and 3rd returned values and enables caching.

```
memwidget = wibox.widget.textbox()
vicious.cache(vicious.widgets.mem)
vicious.register(memwidget, vicious.widgets.mem, "$1 ($2MiB/$3MiB)", 13)
```

1.3.3 HDD Temperature Widget

Update every 19 seconds, request the temperature level of `/dev/sda` and append *°C* to the returned value. Since the listening port is not provided, default one is used.

```
hddtempwidget = wibox.widget.textbox()
vicious.register(hddtempwidget, vicious.widgets.hddtemp, "${/dev/sda} °C", 19)
```

1.3.4 Mbox Widget

Updated every 5 seconds, provide full path to the mbox as argument:

```
mboxwidget = wibox.widget.textbox()
vicious.register(mboxwidget, vicious.widgets.mbox, "$1", 5,
                 "/home/user/mail/Inbox")
```

1.3.5 Battery Widget

Update every 61 seconds, request the current battery charge level and displays a progressbar, provides `BAT0` as battery ID:

```
batwidget = wibox.widget.progressbar()

-- Create wibox with batwidget
batbox = wibox.layout.margin(
    wibox.widget{ { max_value = 1, widget = batwidget,
                    border_width = 0.5, border_color = "#000000",
```

(continues on next page)

(continued from previous page)

```
                    color = { type = "linear",
                              from = { 0, 0 },
                              to = { 0, 30 },
                              stops = { { 0, "#AECF96" },
                                        { 1, "#FF5656" } } } },
                  forced_height = 10, forced_width = 8,
                  direction = 'east', color = beautiful.fg_widget,
                  layout = wibox.container.rotate },
    1, 1, 3, 3)

-- Register battery widget
vicious.register(batwidget, vicious.widgets.bat, "$2", 61, "BAT0")
```

1.3.6 CPU Usage Widget

Update every 3 seconds, feed the graph with total usage percentage of all CPUs/cores:

```
cpuwidget = awful.widget.graph()
cpuwidget:set_width(50)
cpuwidget:set_background_color"#494B4F"
cpuwidget:set_color{ type = "linear", from = { 0, 0 }, to = { 50, 0 },
                     stops = { { 0, "#FF5656" },
                               { 0.5, "#88A175" },
                               { 1, "#AECF96" } } }
vicious.register(cpuwidget, vicious.widgets.cpu, "$1", 3)
```

1.4 Officially Supported Widget Types

Widget types consist of worker functions that take two arguments `format` and `warg` (in that order), which were previously passed to *vicious.register()*, and return a table of values to be formatted by `format`.

1.4.1 vicious.widgets.bat

Provides state, charge, and remaining time for a requested battery.

Supported platforms: GNU/Linux (require `sysfs`), FreeBSD (require `acpiconf`) and OpenBSD (no extra requirements).

- `warg` (from now on will be called *argument*):
  - On GNU/Linux: battery ID, e.g. `"BAT0"`
  - On FreeBSD (optional): battery ID, e.g. `"batt"` or `"0"`
  - On OpenBSD (optional): `bat` followed by battery index, e.g. `"bat0"` or `"bat1"` on systems with more than one battery
- Returns an array (integer-indexed table) consisting of:
  - `$1`: State of requested battery
  - `$2`: Charge level in percent

  - $3: Remaining (charging or discharging) time
  - $4: Wear level in percent
  - $5: Current (dis)charge rate in Watt

1.4.2 vicious.contrib.cmus

Provides cmus player information using `cmus-remote`.

Supported platforms: platform independent.

- Argument: a table whose first field is the socket including host (or nil).
- Returns a table with string keys: `${status}`, `${artist}`, `${title}`, `${duration}`, `${file}`, `${continue}`, `${shuffle}`, `${repeat}`.

1.4.3 vicious.widgets.cpu

Provides CPU usage for all available CPUs/cores. Since this widget type give CPU utilization between two consecutive calls, it is recommended to enable caching if it is used to register multiple widgets (#71).

Supported platforms: GNU/Linux, FreeBSD, OpenBSD.

On FreeBSD and Linux returns an array containing:

- $1: usage of all CPUs/cores
- $2, $3, etc. are respectively the usage of 1st, 2nd, etc. CPU/core

On OpenBSD returns an array containing:

- $1: usage of all CPUs/cores

1.4.4 vicious.widgets.cpufreq

Provides freq, voltage and governor info for a requested CPU.

Supported platforms: GNU/Linux, FreeBSD.

- Argument: CPU ID, e.g. `"cpu0"` on GNU/Linux, `"0"` on FreeBSD
- Returns an array containing:
  - $1: Frequency in MHz
  - $2: Frequency in GHz
  - $3: Voltage in mV
  - $4: Voltage in V
  - $5: Governor state
  - On FreeBSD: only the first two are supported (other values will always be `"N/A"`)

1.4.5 vicious.widgets.cpuinf

Provides speed and cache information for all available CPUs/cores.

Supported platforms: GNU/Linux.

Returns a table whose keys using CPU ID as a base, e.g. `${cpu0 mhz}`, `${cpu0 ghz}`, `${cpu0 kb}`, `${cpu0 mb}`, `${cpu1 mhz}`, etc.

1.4.6 vicious.widgets.date

Provides access to Lua's `os.date`, with optional settings for time format and time offset.

Supported platforms: platform independent.

- `format` (optional): a strftime(3) format specification string (format functions are not supported). If not provided, use the prefered representation for the current locale.
- Argument (optional): time offset in seconds, e.g. for different a time zone. If not provided, current time is used.
- Returns the output of `os.date` formatted by `format` *string*.

1.4.7 vicious.widgets.dio

Provides I/O statistics for all available storage devices.

Supported platforms: GNU/Linux.

Returns a table with string keys: `${sda total_s}`, `${sda total_kb}`, `${sda total_mb}`, `${sda read_s}`, `${sda read_kb}`, `${sda read_mb}`, `${sda write_s}`, `${sda write_kb}`, `${sda write_mb}`, `${sda iotime_ms}`, `${sda iotime_s}`, `${sdb1 total_s}`, etc.

1.4.8 vicious.widget.fanspeed

Provides fanspeed information for specified fans.

Supported platforms: FreeBSD.

- Argument: full `sysctl` string to one or multiple entries, e.g. `"dev.acpi_ibm.0.fan_speed"`
- Returns speed of specified fan in RPM, `"N/A"` on error (probably wrong string)

1.4.9 vicious.widgets.fs

Provides usage of disk space.

Supported platforms: platform independent.

- Argument (optional): if true includes remote filesystems, otherwise fallback to default, where only local filesystems are included.
- Returns a table with string keys, using mount points as a base, e.g. `${/ size_mb}`, `${/ size_gb}`, `${/ used_mb}`, `${/ used_gb}`, `${/ used_p}`, `${/ avail_mb}`, `${/ avail_gb}`, `${/ avail_p}`, `${/home size_mb}`, etc. mb and gb refer to mebibyte and gibibyte respectively.

1.4.10 vicious.widgets.gmail

Provides count of new and subject of last e-mail on Gmail.

Supported platform: platform independent, requiring `curl`.

This widget expects login information in your `~/.netrc` file, e.g. `machine mail.google.com login user password pass`. Use your app password if you can, or disable two step verification and allow access for less secure apps.

Caution: Making these settings is a security risk!

- Arguments (optional): either a number or a table
  - If it is a number, subject will be truncated.
  - If it is a table whose first field is the maximum length and second field is the widget name (e.g. `"gmailwidget"`), scrolling will be used.
- Returns a table with string keys: `${count}` and `${subject}`

1.4.11 vicious.widgets.hddtemp

Provides hard drive temperatures using the hddtemp daemon.

Supported platforms: GNU/Linux, requiring `hddtemp` and `curl`.

- Argument (optional): `hddtemp` listening port (default: 7634)
- Returns a table with string keys, using hard drives as a base, e.g. `${/dev/sda}` and `${/dev/sdc}`.

1.4.12 vicious.widgets.hwmontemp

Provides name-based access to hwmon devices via sysfs.

Supported platforms: GNU/Linux

- Argument: an array with sensor name and input number (optional, falling back to 1), e.g. `{"radeon", 2}`
- Returns a table with just the temperature value: `$1`
- Usage example:

```
gputemp = wibox.widget.textbox()
vicious.register(gputemp, vicious.widgets.hwmontemp, " $1°C", 5, {"radeon"})
```

1.4.13 vicious.widgets.mbox

Provides the subject of last e-mail in a mbox file.

Supported platforms: platform independent.

- Argument: either a string or a table:
  - A string representing the full path to the mbox, or
  - Array of the form `{path, maximum_length[, widget_name]}`. If the widget name is provided, scrolling will be used.

  - Note: the path will be escaped so special variables like ~ will not work, use `os.getenv` instead to access environment variables.

- Returns an array whose first value is the subject of the last e-mail.

1.4.14 vicious.widgets.mboxc

Provides the count of total, old and new messages in mbox files.

Supported platforms: platform independent.

- Argument: an array full paths to mbox files.
- Returns an array containing:
  - `$1`: Total number of messages
  - `$2`: Number of old messages
  - `$3`: Number of new messages

1.4.15 vicious.widgets.mdir

Provides the number of unread messages in Maildir structures/directories.

Supported platforms: platform independent.

- Argument: an array with full paths to Maildir structures.
- Returns an array containing:
  - `$1`: Number of new messages
  - `$2`: Number of *old* messages lacking the *Seen* flag

1.4.16 vicious.widgets.mem

Provides RAM and Swap usage statistics.

Supported platforms: GNU/Linux, FreeBSD.

Returns (per platform): * GNU/Linux: an array consisting of:

- `$1`: Memory usage in percent
- `$2`: Memory usage in MiB
- `$3`: Total system memory in MiB
- `$4`: Free memory in MiB
- `$5`: Swap usage in percent
- `$6`: Swap usage in MiB
- `$7`: Total system swap in MiB
- `$8`: Free swap in MiB
- `$9`: Memory usage with buffers and cache, in MiB
- FreeBSD: an array including:
  - `$1`: Memory usage in percent

- `$2`: Memory usage in MiB
- `$3`: Total system memory in MiB
- `$4`: Free memory in MiB
- `$5`: Swap usage in percent
- `$6`: Swap usage in MiB
- `$7`: Total system swap in MiB
- `$8`: Free swap in MiB
- `$9`: Wired memory in percent
- `$10`: Wired memory in MiB
- `$11`: Unfreeable memory (basically active+inactive+wired) in percent
- `$12`: Unfreeable memory in MiB

1.4.17 vicious.widgets.mpd

Provides Music Player Daemon information.

Supported platforms: platform independent (required tools: `curl`).

- Argument: an array including password, hostname and port in that order. `nil` fields will be fallen back to default (`localhost:6600` without password).
- Returns a table with string keys: `${volume}`, `${bitrate}`, `${elapsed}` (in seconds), `${duration}` (in seconds), `${Elapsed}` (formatted as [hh:]mm:ss), `${Duration}` (formatted as [hh:]mm:ss), `${Progress}` (in percentage), `${random}`, `${repeat}`, `${state}`, `${Artist}`, `${Title}`, `${Album}`, `${Genre}` and optionally `${Name}` and `${file}`.

In addition, some common mpd commands are available as functions: `playpause`, `play`, `pause`, `stop`, `next`, `previous`. Arguments are of the same form as above, and no value is returned, e.g. `vicious.widgets.mpd.playpause()`.

1.4.18 vicious.widgets.net

Provides state and usage statistics of network interfaces.

Supported platforms: GNU/Linux, FreeBSD.

- Argument (FreeBSD only): desired interface, e.g. `"wlan0"`
- Returns (per platform):
  - GNU/Linux: a table with string keys, using net interfaces as a base, e.g. `${eth0 carrier}`, `${eth0 rx_b}`, `${eth0 tx_b}`, `${eth0 rx_kb}`, `${eth0 tx_kb}`, `${eth0 rx_mb}`, `${eth0 tx_mb}`, `${eth0 rx_gb}`, `${eth0 tx_gb}`, `${eth0 down_b}`, `${eth0 up_b}`, `${eth0 down_kb}`, `${eth0 up_kb}`, `${eth0 down_mb}`, `${eth0 up_mb}`, `${eth0 down_gb}`, `${eth0 up_gb}`, `${eth1 rx_b}`, etc.
  - FreeBSD: a table with string keys: `${carrier}`, `${rx_b}`, `${tx_b}`, `${rx_kb}`, `${tx_kb}`, `${rx_mb}`, `${tx_mb}`, `${rx_gb}`, `${tx_gb}`, `${down_b}`, `${up_b}`, `${down_kb}`, `${up_kb}`, `${down_mb}`, `${up_mb}`, `${down_gb}`, `${up_gb}`.

1.4.19 vicious.widgets.notmuch

Provides a message count according to an arbitrary Notmuch query.

Supported platforms: platform independent.

Argument: the query that is passed to Notmuch. For instance: `tag:inbox AND tag:unread` returns the number of unread messages with tag "inbox".

Returns a table with string keys containing:

- `${count}`: the count of messages that match the query

1.4.20 vicious.widgets.org

Provides agenda statistics for Emacs org-mode.

Supported platforms: platform independent.

- Argument: an array of full paths to agenda files, which will be parsed as arguments.
- Returns an array consisting of
  - `$1`: Number of tasks you forgot to do
  - `$2`: Number of tasks for today
  - `$3`: Number of tasks for the next 3 days
  - `$4`: Number of tasks to do in the week

1.4.21 vicious.widgets.os

Provides operating system information.

Supported platforms: platform independent.

Returns an array containing:

- `$1`: Operating system in use
- `$2`: Release version
- `$3`: Username
- `$4`: Hostname
- `$5`: Available system entropy
- `$6`: Available entropy in percent

1.4.22 vicious.widgets.pkg

Provides number of pending updates on UNIX systems. Be aware that some package managers need to update their local databases (as root) before showing the correct number of updates.

Supported platforms: platform independent, although it requires Awesome `awful.spawn` library for non-blocking spawning.

- Argument: distribution name, e.g. `"Arch"`, `"Arch C"`, `"Arch S"`, `"Debian"`, `"Ubuntu"`, `"Fedora"`, `"FreeBSD"`, `"Mandriva"`.

- Returns an array including:
  - `$1`: Number of available updates
  - `$2`: Packages available for update

1.4.23 vicious.widgets.raid

Provides state information for a requested RAID array.

Supported platforms: GNU/Linux.

- Argument: the RAID array ID.
- Returns an array containing:
  - `$1`: Number of assigned devices
  - `$2`: Number of active devices

1.4.24 vicious.widgets.thermal

Provides temperature levels of several thermal zones.

Supported platforms: GNU/Linux, FreeBSD.

- Argument (per platform):
  - GNU/Linux: either a string - the thermal zone, e.g. `"thermal_zone0"`, or a table of the form `{thermal_zone, data_source[, input_file]}`. Available `data_source`'s and corresponding default `input_file` are given in the table below. For instance, if `"thermal_zone0"` is passed, temperature would be read from `/sys/class/thermal/thermal_zone0/temp`. This widget type is confusing and ugly but it is kept for backward compatibility.
  - FreeBSD: either a full `sysctl` path to a thermal zone, e.g. `"hw.acpi.thermal.tz0.temperature"`, or a table with multiple paths.
- Returns (per platform):
  - GNU/Linux: an array whose first value is the requested temperature.
  - FreeBSD: a table whose keys are provided paths thermal zones.

`data_source`	Path	Default `input_file`
`"sys"`	/sys/class/thermal/	`"temp"`
`"core"`	/sys/devices/platform/	`"temp2_input"`
`"hwmon"`	/sys/class/hwmon/	`"temp1_input"`
`"proc"`	/proc/acpi/thermal_zone/	`"temperature"`

1.4.25 vicious.widgets.uptime

Provides system uptime and load information.

Supported platforms: GNU/Linux, FreeBSD.

Returns an array containing:

- `$1`: Uptime in days
- `$2`: Uptime in hours
- `$3`: Uptime in minutes
- `$4`: Load average in the past minute
- `$5`: Load average in the past 5 minutes
- `$6`: Load average in the past 15 minutes

1.4.26 vicious.widgets.volume

Provides volume levels and state of requested mixers.

Supported platforms: GNU/Linux (requiring `amixer`), FreeBSD.

- Argument (per platform):
  - GNU/Linux: either a string containing the ALSA mixer control (e.g. `"Master"`) or a table including command line arguments to be passed to amixer(1), e.g. `{"PCM", "-c", "0"}` or `{"Master", "-D", "pulse"}`
  - FreeBSD: the mixer control, e.g. `"vol"`
- Returns an array consisting of (per platform):
  - GNU/Linux: `$1` as the volume level and `$2` as the mute state of the requested control
  - FreeBSD: `$1` as the volume level of the *left* channel, `$2` as the volume level of the *right* channel and `$3` as the mute state of the desired control

1.4.27 vicious.widgets.weather

Provides weather information for a requested station.

Supported platforms: any having Awesome and `curl` installed.

- Argument: the ICAO station code, e.g. `"LDRI"`
- Returns a table with string keys: `${city}`, `${wind}`, `${windmph}`, `${windkmh}`, `${sky}`, `${weather}`, `${tempf}`, `${tempc}`, `${humid}`, `${dewf}`, `${dewc}` and `${press}`, `${when}`

1.4.28 vicious.widgets.wifi

Provides wireless information for a requested interface.

Supported platforms: GNU/Linux.

- Argument: the network interface, e.g. `"wlan0"`
- Returns a table with string keys: `${ssid}`, `${mode}`, `${chan}`, `${rate}` (Mb/s), `${freq}` (MHz), `${txpw}` (transmission power, in dBm), `${sign}` (signal level), `${link}` and `${linp}` (link quality per 70 and per cent)

1.4.29 vicious.widgets.wifiiw

Provides wireless information for a requested interface (similar to vicious.widgets.wifi, but uses `iw` instead of `iwconfig`).

Supported platforms: GNU/Linux.

- Argument: the network interface, e.g. `"wlan0"`
- Returns a table with string keys: `${bssid}`, `${ssid}`, `${mode}`, `${chan}`, `${rate}` (Mb/s), `${freq}` (MHz), `${linp}` (link quality in percent), `${txpw}` (transmission power, in dBm) and `${sign}` (signal level, in dBm)

1.5 Contrib Widget Types

Contrib libraries, or widget types, are extra snippets of code you can use. Some are for less common hardware, and other were contributed by Vicious users. The contrib directory also holds widget types that were obsoleted or rewritten. Contrib widgets will not be imported by init unless you explicitly enable it, or load them in your `rc.lua`.

1.5.1 Usage within Awesome

To use contrib widgets uncomment the line that loads them in `init.lua`. Or you can load them in your rc.lua after you require Vicious:

```
local vicious = require"vicious"
vicious.contrib = require"vicious.contrib"
```

1.5.2 Widget Types

Most widget types consist of worker functions that take the `format` argument given to *vicious.register()* as the first argument, `warg` as the second, and return a table of values to insert in the format string. But we have not insisted on this coding style in contrib. So widgets like PulseAudio have emerged that are different. These widgets could also depend on Lua libraries that are not distributed with the core Lua distribution. Ease of installation and use does not necessarily have to apply to contributed widgets.

vicious.contrib.ac

Provide status about the power supply (AC).

Supported platforms: GNU/Linux, requiring `sysfs`.

- Argument: the AC device, i.e `"AC"` or `"ACAD"`. The device is linked under `/sys/class/power_supply/` and should have a file called `online`.
- Returns `{"On"}` if AC is connected, else `{"Off"}`. If AC doesn't exist, returns `{"N/A"}`.

vicious.contrib.ati

Provides various info about ATI GPU status.

Supported platforms: GNU/Linux, requiring `sysfs`.

- Argument: card ID, e.g. `"card0"` (and where possible, uses `debugfs` to gather data on radeon power management)
- Returns a table with string keys: `${method}`, `${dpm_state}`, `${dpm_perf_level}`, `${profile}`, `${engine_clock mhz}`, `${engine_clock khz}`, `${memory_clock mhz}`, `${memory_clock khz}`, `${voltage v}`, `${voltage mv}`

vicious.contrib.batpmu

vicious.contrib.batproc

vicious.contrib.btc

Provides current Bitcoin price in any currency by [code](https://en.wikipedia.org/wiki/ISO_4217).

Platform independent, although requiring `curl` and either [lua-cjson](https://github.com/mpx/lua-cjson/) or [luajson](https://github.com/harningt/luajson/).

- Argument: currency code, e.g. `"usd"`, `"rub"` and other. Default to `"usd"`.
- Returns a table with string key `${price}`.

vicious.contrib.buildbot

Provides last build status for configured buildbot builders (http://trac.buildbot.net/).

Supported platforms: platform independent, though requiring Lua JSON parser [luajson](https://github.com/harningt/luajson/).

Returns build status in the format: `[<builderName>.<currentBuildNumber>.<lastSuccessfulBuildNumber>]`. If `<currentBuildNumber>` is the same as `<lastSuccessfulBuildNumber>` only one number is displayed. `<buildNumber>` colors: red—failed, green—successful, yellow—in progress.

vicious.contrib.countfiles

vicious.contrib.cmus

Note: This widget type has been promoted to *Officially Supported Widget Types*.

Provides cmus player information using `cmus-remote`.

Supported platforms: platform independent.

- Argument: a table whose first field is the socket including host (or nil).
- Returns a table with string keys: `${status}`, `${artist}`, `${title}`, `${duration}`, `${file}`, `${continue}`, `${shuffle}`, `${repeat}`.

vicious.contrib.dio

Provides I/O statistics for requested storage devices.

- Argument: the disk as an argument, i.e. `"sda"`, or a specific partition, i.e. `"sda/sda2"`
- Returns a table with string keys: `${total_s}`, `${total_kb}`, `${total_mb}`, `${read_s}`, `${read_kb}`, `${read_mb}`, `${write_s}`, `${write_kb}`, `${write_mb}` and `${sched}`

vicious.contrib.mpc

vicious.contrib.netcfg

vicious.contrib.net

vicious.contrib.openweather

Provides weather information for a requested city from OpenWeatherMap (OWM)

- Argument: a table containing the fields `city_id` with the OWM city ID, e.g. `"2643743"` and `app_id` with the the OWM app ID, e.g `"4c57f0c88d9844630327623633ce269cf826ab99"`
- Returns a table with string keys: `${city}`, `${humid}`, `${press}`, `${sky}`, `${sunrise}`, `${sunset}`, `${temp c}`, `${temp max c}`, `${temp min c}`, `${weather}`, `${wind aim}`, `${wind deg}`, `${wind kmh}` and `${wind mps}`,

vicious.contrib.nvinf

Provides GPU utilization, core temperature, clock frequency information about Nvidia GPU from nvidia-settings

Supported Platforms: platform independent

- Argument (optional): card ID as an argument, e.g. `"1"`, default to ID 0
- Returns an array containing:
  - `$1`: Usage of GPU core
  - `$2`: Usage of GPU memory
  - `$3`: Usage of video engine

  - `$4`: Usage of PCIe bandwidth
  - `$5`: Uemperature of requested graphics device
  - `$6`: Urequency of GPU core
  - `$7`: Uemory transfer rate

vicious.contrib.nvsmi

Provides (very basic) information about Nvidia GPU status from SMI

Supported platforms: platform independent

- Argument (optional): card ID as an argument, e.g. `"1"`, default to ID 0
- Returns an array containing temperature of requested graphics device

vicious.contrib.ossvol

vicious.contrib.pop

vicious.contrib.pulse

Provides volume levels of requested pulseaudio sinks and functions to manipulate them

- Argument (optional): name of a sink as an optional argument. A number will be interpret as an index, if no argument is given, it will take the first-best. To get a list of available sinks run `pacmd list-sinks | grep 'name:'`.
- Returns an array whose only element is the volume level

vicious.contrib.pulse.add(percent[, sink])

- `percent` is the percentage to increment or decrement the volume from its current value
- Returns the exit status of `pacmd`

vicious.contrib.pulse.toggle([sink])

- Toggles mute state
- Returns the exit status of `pacmd`

vicious.contrib.rss

vicious.contrib.sensors

vicious.contrib.wpa

Provides information about the wifi status.

Supported Platforms: platform independent, requiring `wpa_cli`.

- Argument: the interface, e.g. `"wlan0"` or `"wlan1"`

- Returns a table with string keys: ${ssid}, ${qual}, ${ip}, ${bssid}

1.5.3 Usage Examples

PulseAudio Widget

```
vol = wibox.widget.textbox()
local sink = "alsa_output.pci-0000_00_1b.0.analog-stereo"
vicious.register(vol, vicious.contrib.pulse, " $1%", 2, sink)
vol:buttons(awful.util.table.join(
    awful.button({}, 1, function () awful.util.spawn("pavucontrol") end),
    awful.button({}, 4, function () vicious.contrib.pulse.add(5, sink) end),
    awful.button({}, 5, function () vicious.contrib.pulse.add(-5, sink) end)))
```

Buildbot Widget

```
buildbotwidget = wibox.widget.textbox()
vicious.register(
    buildbotwidget, vicious.contrib.buildbot, "$1,", 3600,
    { { builder="coverage", url="http://buildbot.buildbot.net" },
      { builder="tarball-slave", url="http://buildbot.buildbot.net" } })
```

1.6 Custom Widget Types

Use any of the existing widget types as a starting point for your own. Write a quick worker function that does the work and plug it in. How data will be formatted, will it be red or blue, should be defined in `rc.lua` (or somewhere else, outside the actual module).

Before writing a widget type you should check if there is already one in the contrib directory of Vicious. The contrib directory contains extra widgets you can use. Some are for less common hardware, and others were contributed by Vicious users. Most of the contrib widgets are obsolete. Contrib widgets will not be imported by init unless you explicitly enable it, or load them in your `rc.lua`.

Some users would like to avoid writing new modules. For them Vicious kept the old Wicked functionality, possibility to register their own functions as widget types. By providing them as the second argument to *vicious.register()*. Your function can accept `format` and `warg` arguments, just like workers.

1.7 Format Functions

You can use a function instead of a string as the format parameter. Then you are able to check the value returned by the widget type and change it or perform some action. You can change the color of the battery widget when it goes below a certain point, hide widgets when they return a certain value or maybe use `string.format` for padding.

Do not confuse this with just coloring the widget, in those cases standard Pango markup can be inserted into the format string.

The format function will get the widget as its first argument, table with the values otherwise inserted into the format string as its second argument, and will return the text/data to be used for the widget.

1.7.1 Examples

Hide mpd widget when no song is playing

```
mpdwidget = wibox.widget.textbox()
vicious.register(
    mpdwidget,
    vicious.widgets.mpd,
    function (widget, args)
        if args["{state}"] == "Stop" then
            return ''
        else
            return ('<span color="white">MPD:</span> %s - %s'):format(
                args["{Artist}"], args["{Title}"])
        end
    end)
```

Use string.format for padding

```
uptimewidget = wibox.widget.textbox()
vicious.register(uptimewidget, vicious.widgets.uptime,
                 function (widget, args)
                     return ("Uptime: %02d %02d:%02d "):format(
                         args[1], args[2], args[3])
                 end, 61)
```

When it comes to padding it is also useful to mention how a widget can be configured to have a fixed width. You can set a fixed width on your textbox widgets by changing their `width` field (by default width is automatically adapted to text width). The following code forces a fixed width of 50 px to the uptime widget, and aligns its text to the right:

```
uptimewidget = wibox.widget.textbox()
uptimewidget.width, uptimewidget.align = 50, "right"
vicious.register(uptimewidget, vicious.widgets.uptime, "$1 $2:$3", 61)
```

Stacked graph

Stacked graphs are handled specially by Vicious: `format` functions passed to the corresponding widget types must return an array instead of a string.

```
cpugraph = wibox.widget.graph()
cpugraph:set_stack(true)
cpugraph:set_stack_colors{ "red", "yellow", "green", "blue" }
vicious.register(cpugraph, vicious.widgets.cpu,
                 function (widget, args)
                     return { args[2], args[3], args[4], args[5] }
                 end, 3)
```

The snipet above enables graph stacking/multigraph and plots usage of all four CPU cores on a single graph.

Substitute widget types' symbols

If you are not happy with default symbols used in volume, battery, cpufreq and other widget types, use your own symbols without any need to modify modules. The following example uses a custom table map to modify symbols representing the mixer state: on or off/mute.

```
volumewidget = wibox.widget.textbox()
vicious.register(volumewidget, vicious.widgets.volume,
                 function (widget, args)
                     local label = { ["] = "O", ["] = "M" }
                     return ("Volume: %d%% State: %s"):format(
                         args[1], label[args[2]])
                 end, 2, "PCM")
```

Get data from the widget

vicious.call() could be useful for naughty notification and scripts:

```
mybattery = wibox.widget.textbox()
vicious.register(mybattery, vicious.widgets.bat, "$2%", 17, "0")
mybattery:buttons(awful.util.table.join(awful.button(
    {}, 1,
    function ()
        naughty.notify{ title = "Battery indicator",
                        text = vicious.call(vicious.widgets.bat,
                                            "Remaining time: $3", "0") }
    end)))
```

Format functions can be used as well:

```
mybattery:buttons(awful.util.table.join(awful.button(
    {}, 1,
    function ()
        naughty.notify{
            title = "Battery indicator",
            text = vicious.call(
                vicious.widgets.bat,
                function (widget, args)
                    return ("%s: %10sh\n%s: %14d%%\n%s: %12dW"):format(
                        "Remaining time", args[3],
                        "Wear level", args[4],
                        "Present rate", args[5])
                end, "0") }
    end)))
```

1.8 Power and Caching

When a lot of widgets are in use they, and awesome, can generate a lot of wake-ups and also be very expensive for system resources. This is especially important when running on battery power. It was a big problem with awesome v2 and widgets that used shell scripts to gather data, and with widget libraries written in languages like Ruby.

Lua is an extremely fast and efficient programming language, and Vicious takes advantage of that. But suspending Vicious widgets is one way to prevent them from draining your battery, despite that.

Update intervals also play a big role, and you can save a lot of power with a smart approach. Don't use intervals like: 5, 10, 30, 60, etc. to avoid harmonics. If you take the 60-second mark as an example, all of your widgets would be executed at that point. Instead think about using only prime numbers, in that case you will have only a few widgets executed at any given time interval. When choosing intervals also consider what a widget actually does. Some widget types read files that reside in memory, others call external utilities and some, like the mbox widget, read big files.

Vicious can also cache values returned by widget types. Caching enables you to have multiple widgets using the same widget type. With caching its worker function gets executed only once—which is also great for saving power.

- Some widget types keep internal data and if you call one multiple times without caching, the widget that executes it first would modify stored values. This can lead to problems and give you inconsistent data. Remember it for widget types like CPU and Network usage, which compare the old set of data with the new one to calculate current usage.
- Widget types that require a widget argument to be passed should be handled carefully. If you are requesting information for different devices then caching should not be used, because you could get inconsistent data.

1.9 Security Notes

At the moment only one widget type (Gmail) requires authentication information in order to get to the data. In the future there could be more, and you should give some thought to the issue of protecting your data. The Gmail widget type by default stores login information in the `~/.netrc` file, and you are advised to make sure that file is only readable by the owner. Other than that we can not force all users to conform to one standard, one way of keeping it secure, like in some keyring.

First let's clear why we simply don't encrypt the login information and store it in ciphertext. By exposing the algorithm anyone can reverse the encryption steps. Some claim even that's better than plaintext but it's just security through obscurity.

Here are some ideas actually worth your time. Users that have KDE (or parts of it) installed could store their login information into the Kwallet service and request it via DBus from the widget type. It can be done with tools like `dbus-send` and `qdbus`. The Gnome keyring should support the same, so those with parts of Gnome installed could use that keyring.

Users of GnuPG (and its agent) could consider encrypting the netrc file with their GPG key. Through the GPG Passphrase Agent they could then decrypt the file transparently while their session is active.

1.10 Contribution Guidelines

1.10.1 Filing an Issue

- Ensure the bug was not already reported by searching GitHub Issues.
- If you're unable to find an open issue addressing the problem, open a new one. Be sure to include a title and clear description, as much relevant information as possible, such as Awesome errors, and a config sample or an executable test case (using Vicious as a stand-alone library) demonstrating the expected behavior that is not occurring.

Please re-read your issue once again to avoid a couple of common mistakes (you can and should use this as a checklist):

- Is the description of the issue itself sufficient? Make sure that it's obvious
  - What the problem is
  - How it could be fixed
  - How your proposed solution would look like
- Have you provide the versions of Vicious and related software? We would like to how you installed Vicious, which OS you're using, the version of the software or what kind of hardware you are trying to get information from.
- Is the issue already documented?
- Does the issue involve one problem, and one problem only? Some people seem to think there is a limit of issues they can or should open. There is no limit of issues they can or should open. While it may seem appealing to be able to dump all your issues into one ticket, that means that someone who solves one of your issues cannot mark the issue as closed.
- Is anyone going to need the feature? Only post features that you (or an incapacitated friend you can personally talk to) require. Do not post features because they seem like a good idea. If they're really useful, they'll be requested by someone who requires them.

1.10.2 Requesting for Merging a Patch

1. Fork this repository
2. Check out the source code with:

```
git clone git@github.com:YOUR_GITHUB_USERNAME/vicious
cd vicious
```

3. Start working on your patch. If you want to add a new widget type, see the `templates` directory for a more detailed guide.
4. Have a look at `helpers.lua` and `spawn.lua` for possible helper functions.
5. Make sure your code follows the coding conventions below and check the code with `luacheck`. This *should fail* at first, but you can continually re-run it until you're done:

```
luacheck --config tools/luacheckrc .
```

6. Make sure your code works under all Lua versions claimed supported by Vicious, namely 5.1, 5.2 and 5.3.
7. Update the copyright notices of the files you modified. Vicious is collectively licensed under GPLv2+, and to protect the freedom of the users, copyright holders need to be properly documented.

8. Try to note your changes under `CHANGELOG.rst`. If you find it is difficult to phrase the changes, you can leave it for us.
9. Add the changes, commit them and push the result, like this:

```
git add widgets/bar_baz.lua README.md
git commit -m '[bar_baz] Add widget type'
git add helpers.lua CHANGELOG.rst
git commit -m '[helpers] Fix foo'
git push
```

10. Finally, create a pull request. We'll then review and merge it.

In any case, thank you very much for your contributions!

1.10.3 Coding Conventions

This section introduces a guideline for writing idiomatic, robust and future-proof widget type code.

Whitespace in Expressions and Statements

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses or brackets. Braces, however, are exceptions to this rule:

```
foo(bar[1], { baz = 2 })      -- yes
foo( bar[ 1 ], {baz = 2} )    -- no
```

- Immediately before a comma, semicolon, or colon.
- Immediately before the open parenthesis, braces, quote, etc. that starts the argument list of a function call; or the open bracket that starts an indexing. In other words, prefer these:

```
foo(bar, baz)
foo{ bar, baz }
foo"bar"
foo[[bar]]
foo[bar]
```

- Trailing at the end of line or (newline) at the end of file.

Always surround these binary operators with a single space on either side: assignment (=), comparisons, Booleans (`and`, `or`, `not`). If operators with different priorities are used, consider adding whitespace around the operators with the lowest priorities. Use your own judgment; however, never use more than one space, and always have the same amount of whitespace on both sides of a binary operator.

Indentation

Use 4 *spaces* per indentation level.

Continuation lines should align wrapped elements either vertically inside parentheses, brackets and braces, or using a hanging indent (the opening parenthesis of a parenthesized statement is the last non-whitespace character of the line, with subsequent lines being indented until the closing parenthesis), e.g.

```
-- Vertically aligned
long_function_call{ foo, bar,
                    baz }

-- Hanging indentation
long_function_call(
    foo, bar
    baz)
```

The closing brace or bracket on multi-line constructs may either line up under the first character of the line that starts the construct, as in:

```
long_function_call{
    foo = 1, bar = 2,
    baz = 3,
}
```

In this case, and this case only, the trailing comma is acceptable to avoid diff noises when more values are added, but since Vicious often deal with system APIs which rarely ever change, it's occasionally helpful to do so.

Trailing right parentheses, however, are not allowed.

Maximum Line Length

If possible, try to limit all *code* lines to a maximum of 80 characters. In case you find some lines in your patch would be more readable exceeding this limit, feel free to discuss with us. Comments and long strings need not to follow this restriction however.

As one might have noticed, the syntactic sugars `f{<fields>}` (for `f({<fields>})`) and `f'<string>'` (or `f"<string>"`/`f[[<string>]]`, for `f('<string>')`) are especially preferred to squeeze the line length to this limit.

Blank Lines

Surround function definitions with a single blank line. Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations). Use blank lines in functions, sparingly, to indicate logical sections.

Requiring Libraries

All standard libraries should be localized before used for the matter of performance.

`require`'s should always be put at the top of the source file, just after the copyright header, and before module globals and constants, and grouped in the following order:

1. Standard libraries
2. Related third-party libraries
3. Local libraries

For example,

```
local type = type
local table = { concat = table.concat, insert = table.insert }

local awful = require("awful")

local helpers = require("vicious.helpers")
```

String Quotes

In Lua, single-quoted strings and double-quoted strings are the same, so the choice is totally up to you, but please be consistent within a module. When a string contains single or double quote characters, however, use the other one to avoid backslashes in the string. It improves readability:

```
'"key": "value"'        -- good
"\"key\": \"value\""    -- no good
```

It is preferable to add a newline immediately after the opening long bracket:

```
foo = [[
this is a really,
really,
really long text]]
```

Naming Conventions

Avoid using the characters `l` (lowercase letter el), `O` (uppercase letter oh), or `I` (uppercase letter eye) as single character variable names. In some fonts, these characters are indistinguishable from the 1's and 0's.

Constants

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include `MAX_OVERFLOW` and `TOTAL`.

Function and Variable Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Variable names follow the same convention as function names.

When you find it difficult to give descriptive names, use the functions and variable anonymously.

Performance Tips

Vicious is meant to be run as part of the Awesome window manager, thus any little overhead may defect the responsiveness of the UI. While Lua is famous for its performance, there are a few things one can do to make use of all of its power.

Never use global variables. This includes the standard libraries, which, again, must be localized before use. Remember, every widget type is to be called repeatedly every few seconds.

Use closures when possible:

- Define constants on the module level.
- Avoid re-fetching the values that are not not meant to change.

However, declare a variable only when you need it, to avoid declaring it without an initial value (and therefore you seldom forget to initialize it). Moreover, you shorten the scope of the variable, which increases readability.

Copyright Header

Vicious is released under the GNU GNU General Public License version 2 or later and each contributor holds the copyright on their contributions. To make this collective control effective, each source file must include a notice of the following format denoting the name of all authors

```
-- <one line to give the program's name and a brief idea of what it does.>
-- Copyright (C) <year>  <name of author> <<email that can be use for contact>>
--
-- This file is part of Vicious.
--
-- Vicious is free software: you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as
-- published by the Free Software Foundation, either version 2 of the
-- License, or (at your option) any later version.
--
-- Vicious is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with Vicious.  If not, see <https://www.gnu.org/licenses/>.
```

Comments

Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!

You should use two spaces after a sentence-ending period in multi-sentence comments, except after the final sentence.

Block Comments

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with `--` and a single space, unless text inside the comment is indented, or it is to comment out code.

Paragraphs inside a block comment are separated by a line containing `--` only. The best example is the copyright notice in the section above.

The `--[[...]]` style may only be used for commenting out source code.

Inline Comments

An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with `--` and one single space.

1.10.4 Influences

These contributing guideline are heavily influenced by that of `youtube-dl`, PEP 8, Programming in Lua and the performance tips in Lua Programming Gems.

1.11 License and Credits

Wicked was written by:

- Lucas de Vries <lucas glacicle.com>

Vicious was originally written by:

- Adrian C. (anrxc) <anrxc sysphere.org>.

Vicious is released under GNU GPLv2+ and is currently maintained by:

- Jörg Thalheim <joerg thalheim.io>
- @mutlusun (especially the FreeBSD port)
- Daniel Hahler <github thequod.de>
- Nguyn Gia Phong <mcsinyx disroot.org>
- Enric Morales <geekingaround enric.me> (especially the OpenBSD port)

Over the years, Vicious has also received various patches and improvements from the following contributors, listed in alphabetic order:

- 0x5b <dragen15051 gmail.com>
- Adam Lee <adam8157 gmail.com>

- Alexander Koch <lynix47 gmail.com>
- Amir Mohammad Saied <amirsaied gmail.com>
- Andrea Scarpino <me andreascarpino.it>
- Andreas Geisenhainer <psycorama datenhalde.de>
- Andrew Merenbach <andrew merenbach.com>
- Andrzej Bieniek <andyhelp gmail.com>
- Arthur Axel 'fREW' Schmidt <git frew.co>
- Arvydas Sidorenko <asido4 gmail.com>
- Benedikt Sauer <filmor gmail.com>
- Beniamin Kalinowski <beniamin.kalinowski gmail.com>
- Benoît Zugmeyer <bzugmeyer gmail.com>
- blastmaster <blastmaster tuxcode.org>
- Brandon Hartshorn <brandonhartshorn gmail.com>
- crondog <patches crondog.com>
- David Udelson <dru5 cornell.edu>
- Dodo The Last <dodo.the.last gmail.com>
- Elric Milon <whirm gmx.com>
- getzze <getzze gmail.com>
- Greg D. <jabbas jabbas.pl>
- Hagen Schink <troja84 googlemail.com>
- Henning Glawe <glaweh debian.org>
- Hiltjo Posthuma <hiltjo codemadness.org>
- James Reed
- Jay Kamat <jaygkamat gmail.com>
- Jeremy <jeremy.sainvil gmaill.com>
- jinleileiking <jinleileiking gmail.com>
- joe di castro <joe joedicastro.com>
- Joerg Jaspert <joerg debian.org>
- Jonathan McCrohan <jmccrohan gmail.com>
- Juan Carlos Menonita
- Juergen Descher <jhdl gmx.net>
- Julian Volodia <julianvolodia gmail.com>
- Keith Hughitt <keith.hughitt gmail.com>
- Lorenzo Gaggini <lg lgaggini.net>
- Lyderic Lefever <lyderic.lefever gmail.com>
- Martin Striz <striz raynet.cz>

- Martin Ueding <dev martin-ueding.de>
- Mellich <mellich gmx.net>
- Michael Kressibucher <mkressibucher hotmail.com>
- Michael Unterkalmsteiner <miciu gmx.de>
- niko <nikomomo gmail.com>
- Noah Tilton <code tilton.co>
- Normal Ra <normalrawr gmail.com>
- Perry Hargrave <perry.hargrave gmail.com>
- Rémy CLOUARD <shikamaru shikamaru.fr>
- Roberto
- Sébastien Luttringer <seblu seblu.net>
- Shadowmourne G <s10e live.com>
- starenka <starenka0 gmail.com>
- Suseika <wlasowegor gmail.com>
- Uli Schlachter <psychon znc.in>
- Wtfcoder <matt mattfreeman.co.uk>
- Xaver Hellauer <xaver hellauer.bayern>
- zhrtz <apaterson scramble.io>

and many others.

1.12 Changelog

1.12.1 Changes in 2.5.1

Fixed:

- Escaping of % in `helpers.format`, which affects mpd widget `${Progress}`
- Possible deadlock of when `update` widgets
- [contrib.openweather] New API compatibility, which requires an API key
- [gmail] Authentication documentation

Added:

- [mpd] Support for sending arbitrary commands
- [contrib.openweather] Various new return values

1.12.2 Changes in 2.5.0

Fixed:

- `vicious.call` freezing awesome when used with asynchronous widget types

Added:

- `vicious.call_async` asynchronous analogous to `vicious.call`

Moved:

- Most of the documentation in READMEs to `docs/`
- `Changes.md` to `CHANGELOG.rst`
- `CONTRIBUTING.md` to `CONTRIBUTING.rst`
- Meta helpers to `tools/`

1.12.3 Changes in 2.4.2

Feature: [hwmontemp] Bring back sysfs path cache

1.12.4 Changes in 2.4.1

Fixed:

- [pkg] Fallback the number of lines before packages listing to 0. This fixes crashes on Arch, FreeBSD and Mandriva.
- [mdir] Remove trailing semicolon at the end of command.

1.12.5 Changes in 2.4.0

Important: `volume` now uses and instead of and to show mute state. This BREAKS backward compatibility if users substitute custom symbols from these default.

Added:

- notmuch_all, cpu_freebsd widget types.
- [cmus_all] Promote to `widgets/`.
- [wifiiw_linux] Expose BSSID.
- [wifi_linux] Expose frequency and transmission power.
- `spawn` as a fallback for `awful.spawn` in case Vicious is used as a stand-alone library. This wrapper, however, does NOT provide the facilities to asynchronously spawn new processes. It also lacks a few features such as parsing `stderr` and returning PID.
- `helpers.setasyncall` to avoid writing redundant workers for asynchronous widget types. Note that these workers are only needed in case Vicious is used as a stand-alone library.
- `helpers.setcall` for registering functions as widget types.
- `headergen` script for automatic generation of copyright notices.

- `templates` for the ease of adding new widget types.
- `CONTRIBUTING.md` which guide contributors through the steps of filing an issue or submitting a patch.

Fixed:

- Deprecate the use of `io.popen` in following widgets:
  - wifi_linux, wifiiw_linux, hwmontemp_linux, hddtemp_linux
  - bat_freebsd, mem_freebsd, net_freebsd, thermal_freebsd, uptime_freebsd,
  - cpu_freebsd, cpufreq_freebsd, fanspeed_freebsd
  - bat_openbsd
  - volume, gmail, mdir, mpd, fs
- [mpd] Lua 5.3 compatibility (for real this time); also correct a typo
- [mbox] Update the deprecated `string.gfind` to `string.gmatch`
- [pkg,weather,contrib/btc] Allow function call without Awesome
- [pkg] Use more updated front-ends for Debian/Ubuntu (apt) and Fedora (dnf)
- [os] Splitted os_all into os_linux and os_bsd (and refactored to async)
- Tweak `.luacheckrc` to suit functional style and soft-limit text width to 80
- Update copyright headers for libraries and widget types

Removed:

- `helpers.sysctl` and `helpers.sysctl_table` were removed in favour of `helpers.sysctl_async`.

1.12.6 Changes in 2.3.3

Feature: Add battery widget type for OpenBSD

Fixes:

- [mpd] Lua 5.3 compatibility
- [bat_freebsd] Update battery state symbols

1.12.7 Changes in 2.3.2

Features:

- Support stacked graphs
- [hwmontemp_linux] Provide name-based access to hwmon sensors via sysfs
- [mpd_all] Expose more informations and format time in [hh:]mm:ss

Fixes:

- Improve defaults and mechanism for data caching
- Escape XML entities in results by default
- [weather_all] Update NOAA link and use Awesome asynchronous API
- [mem_linux] Use MemAvailable to calculate free amount
- [mem_freebsd] Correct calculation and switch to swapinfo for swap

- [bat_freebsd] Add critical charging state
- [fs_all] Fix shell quoting of option arguments

Moreover, `.luacheckrc` was added and `README.md` was refomatted for the ease of development.

1.12.8 Changes in 2.3.1

Fixes:

- widgets can be a function again (regression introduced in 2.3.0)

1.12.9 Changes in 2.3.0

Features:

- add btc widget
- add cmus widget
- alsa mixer also accept multiple arguments

Fixes:

- pkg now uses non-blocking asynchronous api

1.12.10 Changes in 2.2.0

Notable changes:

- moved development from git.sysphere.org/vicious to github.com/Mic92/vicious
- official freebsd support
- escape variables before passing to shell
- support for gear timers
- fix weather widget url
- add *vicious.call()* method to obtain data outside of widgets

For older versions please see `git log`.

CHAPTER

TWO

SEE ALSO

- Manual pages: awesome(1), awesomerc(5)
- Awesome declarative layout system
- My first awesome
- Example awesome configuration (outdated)

INDEX

V