Conky Status Window

Setup and Configuration Guide

Contents

Project Overview

Project Overview

These articles are a personal project to document the Conky window that displays the time and other system information on my Linux Debian laptop. The truth is, my Conky window is nothing special compared to others out there. I only created this documentation to show my knowledge of the DITA framework. Therefore, this documentation, and not the Conky window itself, is the real project.

How I Generated This Documentation

I wrote all content using the open source XML Copy Editor, which is not a WYSIWYG editor with a lot of helpful tools. It is barebones and requires the writer to create all content directly in XML markup.

I then used the DITA OT tool directly from the Linux terminal to generate PDF and markdown outputs.

- I publish the PDF output to GitHub alongside the rest of the project's files.
- I publish the markdown output to a separate, private GitHub repository that I've synchronized GitBook. You can view that output here.

About Conky

Conky is an application that can display any kind of information on your screen. Conky has many built-in variables and tools that let you display information about your system or details from external sources like email clients and music players. You can even extend Conky's functionality using scripts written in the Lua programming language.

How To Install Conky

See the separate topic here.

How To Configure Conky

There is no graphical interface for configuring Conky to set up a window. Instead, you edit a configuration file called `.conkyrc`. On Linux systems, you typically save this file in your home directory at `/home/username/.config/conky/`. (You also will save any Lua scripts you create to this same directory.) You can then add a command to the autostart process of your desktop environment or window manager to launch Conky every time you boot your system.

The topics where I've documented my personal Conky window contain further details and hints about how to configure a conky of your own.

Additional Resources

- Find complete documentation on the Conky project's web site.
  - Configuration settings used in the `.conkyrc` file's `conky.config` section are here.
  - Variables used in the `.conkyrc` file's `conky.text` section are here.
- Find source code and additional information on GitHub.
- Although you can install Conky as an .AppImage from the conky project's SourceForge page, first check if a package is available in your OS's software repositories. Installing it from the official repository ensures Conky is updated when new features are released. See the Installing Conky topic for how to install Conky on Debian-based Linux distributions.

Installing Conky

Conky is very popular so it's highly likely that it already appears in your Linux distribution's software repositories. Use your distro's package manager or software store to search for and install Conky. Installing the program this way is easier and also ensures that you install any additional software and/or libraries that Conky depends on.

For example, in Debian-based systems (including Ubuntu and its derivatives), open a terminal and use the APT package manager to search for conky: `apt search conky`. You'll likely see multiple results like conky, conky-all, and more.

Optionally, you can see conky's dependencies and other details before you install it by typing `apt info conky`. You'll see that (as of May 2025) the Conky package depends on the other conky-related packages (conky-std, conky-cli, conky-all) you saw when you ran the previous search. This means installing Conky also will install those dependencies.

To install conky, type `sudo apt install Conky -y`. (The `-y` option tells APT to automatically answer "yes" to any confirmation messages.) Once finished, you can verify Conky was installed correctly by typing `conky --version`, which shows the version number and other information.

If Conky does not appear in your software repositories, you can download it through other channels. Understand that the following methods are outside of your operating system's package manager which means you may need to perform additional steps to get Conky working on your system and you will need to install future updates manually. (This is beyond the scope of this documentation.)

- Download the standalone .AppImage file for Conky from the project's SourceForge page.
- Download an .AppImage file or a .zip file of the source code (to compile yourself) from the project's GitHub page.

About My Conky

The Elements

My Conky window is simple. It shows the current time and date, and beneath that three meters that show usage information for memory, disk space, and CPU.

Another, more subtle element is a radial gradient color that appears behind the text and meters. It's controlled by a separate Lua script, and it helps separate my Conky window's content from wallpaper backgrounds that may have conflicting colors.

The Files

There are two files that control my Conky window: `.conkyrc` and `draw_bg.lua`. In Linux systems, you should save both files together in your home directory at `/home/username/.config/conky/`. Both files are discussed in more detail in the following topics.

The `.conkyrc` File

The `.conkyrc` file defines the characteristics and appearance of the overall Conky window and also its contents. The topics in this section summarize the file's different parts and highlights a few important options you might want to change. For details about how to use all options in the `.conkyrc` file, see the Conky project's additional resources.

There are two parts to the `.conkyrc` file: `conky.config` and `conky.text`.

conky.config

Here you configure the appearance of the Conky window itself (not the content within). This includes parameters like size, position, and borders, but also the colors and fonts used later when defining the content in the file's `conky.text` section.

The following is not a detailed description of each parameter available. I only highlight the important ones you might want to edit.

Window Behavior and Appearance

The window behavior section defines the general characteristics of the Conky window, specifically the various `own_window_...` options which you can research separately if you want to edit them.

The first two lines deserve attention because they are what integrates the Lua script into this Conky window. As described here, this script draws a color gradient behind the window.

- `lua_load` identifies the location of the Lua script to include.
- `lua_draw_hook_pre` identifies the function within the script that Conky should run.

For details about this script, see this topic that describes the `draw_bg.lua` file.

Window Size and Placement

The parameters in this section control the size of the Conky window and its location on the screen.

- The `minimum_width` and `minimum_height` parameters are self explanatory. They dictate that the Conky window cannot be any smaller than the two values shown.
- Similarly, the `maximum_width` parameter dictates the window cannot be any wider than the value shown. For this Conky window, there is no `minimum_height` parameter, although you can add it if you want.

With the values I currently have, the physical size of the Conky window is much larger than the content I'm displaying. This was intentional, so be careful if you change these values. When either of the minimums were smaller, the background color gradient (controlled by the Lua script) didn't have enough space to fully bloom and the edges of the window cut off the gradient, leaving an obvious line.

Window location:

- The `alignment` parameter specifies the origin anchor for the Conky window. I chose bottom-left. If you choose something else, you will need to compensate by adjusting the gap values below.
- The `gap_x` and `gap_y` parameters then give a horizontal and vertical offset (respectively). So, from the bottom-left anchor point, I've offset the Conky window horizontally (moved to the right) by +1350px. Then I offset the window vertically (moved up) by -120px.

Fonts

The first parameter is the most important; it's where you type the name of the default font you want to use in the Conky window. You can always override this default by specifying an inline font in the `conky.text` section. My Conky windows does this, but I specify a default font here just in case I make a mistake in the `conky.text` section.

Not sure what's the name of the font you want to use? Open a terminal and type `fc-list :family | sort | uniq` to list all fonts installed on your system and sort them by family (removing any duplicates). Each entry of the response includes the path to the font's .tff file and also its name.

In the `.conkyrc` file there is a commented-out line for an icon font, if you want to use icons in your window.

Colors

I'll focus on the custom colors section. You can add and use as many colors as you like; just follow the format. I use only two colors: color4 and color8. (There used to be more colors but I removed the majority and didn't take the time to rename these two.)

- `color8` is the primary font color. Currently there are two variants; one for when the Conky window appears against a darker wallpaper background and another for when the window appears against a lighter background.
- `color4` is an accent color used for the date and some percentage values in the meters. Like the primary color (`color8`), there are two variants depending on the overall color of your background.

Borders

Nothing special here. In the first line, `draw_borders` is set to false, so the border parameters beneath that line don't have any effect and the Conky window is drawn without any borders. If you like borders, go for it.

Miscellaneous

Most of these are self-explanatory, and if you want to change them you can. There's nothing critical to highlight here.

conky.text

This section of the `.conkyrc` file is where you combine variables and text to define what content appears in your Conky window. Here are some basic conventions about this section:

- Declare this section with the `conky.text` parameter followed by a special set of delimiters that mark the beginning and end of the section. These delimiters are custom; I chose `[=[` and `]=]`. You can change them, but make sure the beginning and ending characters match.
- The `conky.text` section uses variables in the format `${...}`. If you type or copy/paste variables, make sure you include everything. Leaving out a variable's `$` character, for example, will break the variable. Also, for variables that take numbers as agruments, those numbers can be positive or negative (e.g., `${voffset -30}` or `${goto 50}`). For more information about variables, follow the link in the additional resources topic.
- In this section you can specify fonts and sizes inline, which will override the default font you set in the `conky.config` section.
- For the purposes of this documentation, I divided the `conky.text` into subsections (SS-1, SS-2, etc.) so I can talk about them separately.
- The `conky.text` section uses the # character for comments because this section uses Conky's own variable language. This differs from the `--` and `--[[...]]` characters used to comment text in the `conky.config` section of this file, because that section follows the syntax for the Lua programming language.

SS-1

Here you'll find the variables and text used to define the Conky window's date and time. The first line defines the format for the clock:

- The vertical offset variable (`${voffset 130}`) pushes the starting point down from the top of the Conky window. Even though the window is aligned bottom-left in the `conky.config` section, I established the overall size of the window larger than necessary to accommodate the full spread of the color gradient that appears in the background. And since the starting point for anything in the `conky.text` section is the top of the window, I needed to push the text down a little.
- The next variable (`${alignc}`) aligns the clock text in the horizontal center of the window.
- The color variable (`${color8}`) tells Conky that any following content should be in the primary color.
- The font variable (`${font...} {font}`) tells Conky that any content inside the variable should have that specific font and size.
- The time variable (`${time}`) establishes the format for the clock's hours and minutes. This variable accepts standard arguments for string format time. (Not sure what those arguments are? Just ask AI to generate a list of the common `strftime` placeholders and their definitions.)

The second line defines the day/date/year:

- In the `conky.text` section, a manual `return` tells Conky to automatically put the following content on the next line. Therefore, there's no need to insert a vertical offset variable (`${voffset}`). I just accepted Conky's default distance for the next line. If you wanted more or less space, however, you can add some offset.
- As with the first line, there is a center alignment variable (`${alignc}`), a color variable(`${color8}`) , and a font variable (`${font...} {font}`) that all perform the same functions described previously.
- The day of the week (`${time %A}`) uses the primary color (`${color8}`), but then switches to the accent color (`${color4}`) for the date (`${time %d}`) and month (`${time %B}`) before returning to the primary color for the year (`${time %Y}`).

SS-2

This section defines what information about system memory usage to display in the Conky window. There's a label, an overall percentage of memory used, a parenthetical depicting the amount of memory used (in GB) out of how much is available (in GB), and a bar to give a visual. The code is spread across two lines, but I'll explain it together.

- There's a vertical offset (`${voffset -17}`) that pushes the meter down a little to give it separation from the date/time information above.
- Then, color (`${color8}`) and font (`${font...} {font}`) variables control the appearance of the "ram" label.
- A horizontal offset variable (`${goto 65}`) tells Conky to jump to the right before drawing the next part. A vertical offset variable (`${voffset -1}`) then pulls it up slightly.
- I change to the accent color (`${color4}`) and add a font variable (`${font...} {font}`) to control the appearance of the memory percentage.
- The memory percentage uses the `${memperc}` variable (which displays just a number) which I follow by manually typing the % symbol.
- Another set of horizontal and vertical offset variables tell Conky where to go to draw the next element.
- I switch back to the primary color (`${color8}`) and adjust the font size before displaying a parenthetical containing the amount of memory used out of the total available. Following is a breakdown of `(${mem -cached} / ${memmax})`:
  - I manually typed the opening and closing parentheses.
  - The first variable is a calculation. It gives a more realistic picture of how much memory is actually in use by subtracting cached memory from the overall memory used.
  - I manually type a slash, and then add the `${memmax}` variable that displays the total amount of installed RAM.
- On the final line, horizontal and vertical offset variables and a color variable affect how Conky draws a percentage-based memory bar. The `${membar 15,140}` variable takes two numeric arguments that draw a bar 15px tall and 140px long.

SS-3

This section defines which information about disk usage to display in the Conky window. There's a label, an overall percentage of disk space used, a parenthetical depicting the amount of space used (in GB) out of how much is available (in GB), and a bar to give a visual. The code for this information is spread across two lines, but I'll explain it together.

- There's a vertical offset (`${voffset -17}`) that raises this information up a little, and a horizontal offset (`${goto 282}`) that pushes the "disk" label to the right.
- A color and font variable control the appearance of the "disk" label.
- A horizontal offset variable (`${goto 332}`) tells Conky to jump to the right before drawing the next part. A vertical offset variable then pulls it up slightly.
- I change to the accent color (`${color4}`) and add a font variable to control the appearance of the disk usage percentage.
- The disk usage percentage uses the `${fs_used_perc}` variable (which displays only a number) followed by a % symbol that I typed manually.
- Another set of horizontal and vertical offset variables tell Conky where to go to draw the next element.
- I switch back to the primary color (`${color8}`) and adjust the font size before displaying a parenthetical containing the amount of disk space used out of the total available. Following is a breakdown of `(${fs_used} / ${fs_size})`:
  - I manually typed the opening and closing parentheses.
  - The first variable is the total amount of disk space currently used.
  - Then I manually typed a slash before adding a second variable that displays the total size of the disk.
- On the final line, horizontal and vertical offset variables and a color variable affect how Conky draws a percentage-based disk usage bar. The `${fs_bar 15,140}` variable takes two numeric arguments that draw a bar 15px tall and 140px long.

SS-4

This section defines which information about CPU usage to display in the Conky window. There's a label, a percentage of CPU usage, and a bar to give a visual. Currently, I have one line that shows usage as an average across all CPU cores (using the `${cpu}` variable).

- A vertical offset raises this information up a little, and then a horizontal offset pushes it to the right.
- A color and font variable control the appearance of the "cpu" label.
- A horizontal offset variable then tells Conky to jump to the right before drawing the next part.
- I change to the accent color (`${color4}`) and add a font variable to control the appearance of the CPU usage percentage.
- The CPU usage percentage uses wthe `${cpu}` variable (which displays just a number) followed by a % symbol that I typed manually. This variable shows a single number that is an average of all cores in your machine.
- A horizontal and slight vertical offset tell Conky where to go to draw the next element, a bar.
- I switch back to the primary color (`${color8}`) and use the `${cpubar 15,190}` variable, which takes two arguments that draw a bar 15px tall and 190px long.

There are two nearly identical lines after this that are currently commented out. If you uncomment them they will show CPU usage for individual cores. If you want to see that level of detail, just copy/paste these lines until you match the number of cores in your computer. Then, for each line update the following two variables to reflect a unique core:

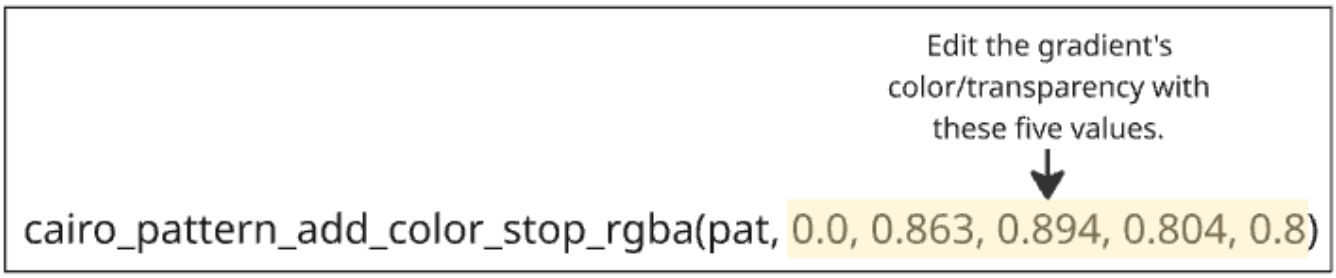
- `${cpu cpu`*`N`*`}` shows the percent usage. Replace *N* with the core's number.
- `${cpubar cpu`*`N`*` 15,190}` draws a bar 15px tall and 190px long. Replace *N* with the core's number.

The `draw_bg.lua` File

The `draw_bg.lua` file is a script (in the Lua programming language) that draws a radial color gradient behind the Conky window's content to help separate that content from any conflicting colors in the background wallpaper. The gradient currently has only two colors, light blue that's 80% opaque and white that's 0% opaque (fully transparent).

Full disclosure: I don't write code in Lua. I asked ChatGPT to write this script for me, and it worked great so I didn't ask questions. Therefore, I'm going to cover only the script's COLOR SECTION where you can change the colors in the gradient.

The gradient's starting color is at line 37: `cairo_pattern_add_color_stop_rgba(pat, 0.0, 0.863, 0.894, 0.804, 0.8)`. Changing the color means editing the decimal values that appear inside the parenthesis.



- The first value (after "pat") is what's known as the offset. A number of 0.0 means that this color is at the start of the gradient (center), and a value of 1.0 means the color is at the end.
- The middle three decimals together form the red-green-blue (RGB) values of the color itself. Converting a hex color (e.g., #DBE4CD) to decimal format involves:
  - Separating the hex into its three individual components (red=DB, green=E4, blue=CD).
  - Replacing each component with its numerical value on the 0-255 scale (go ask AI).
  - Dividing each value by 255 to get a decimal.

HEX color		RGB Parts	Convert to Numeric	Convert to Decimal (÷255)
	red	DB	219	0.863
#DBE4CD	green	E4	228	0.894
	blue	CD	205	0.804

- The final number in the parentheses is the opacity value, where 0.0 is fully transparent and 1.0 is fully opaque.
- To add more colors along the gradient, you can duplicate one of these color stop lines and change the values. The script includes two commented-out lines already set up for this; just change the offset, color values, and opacity of each.