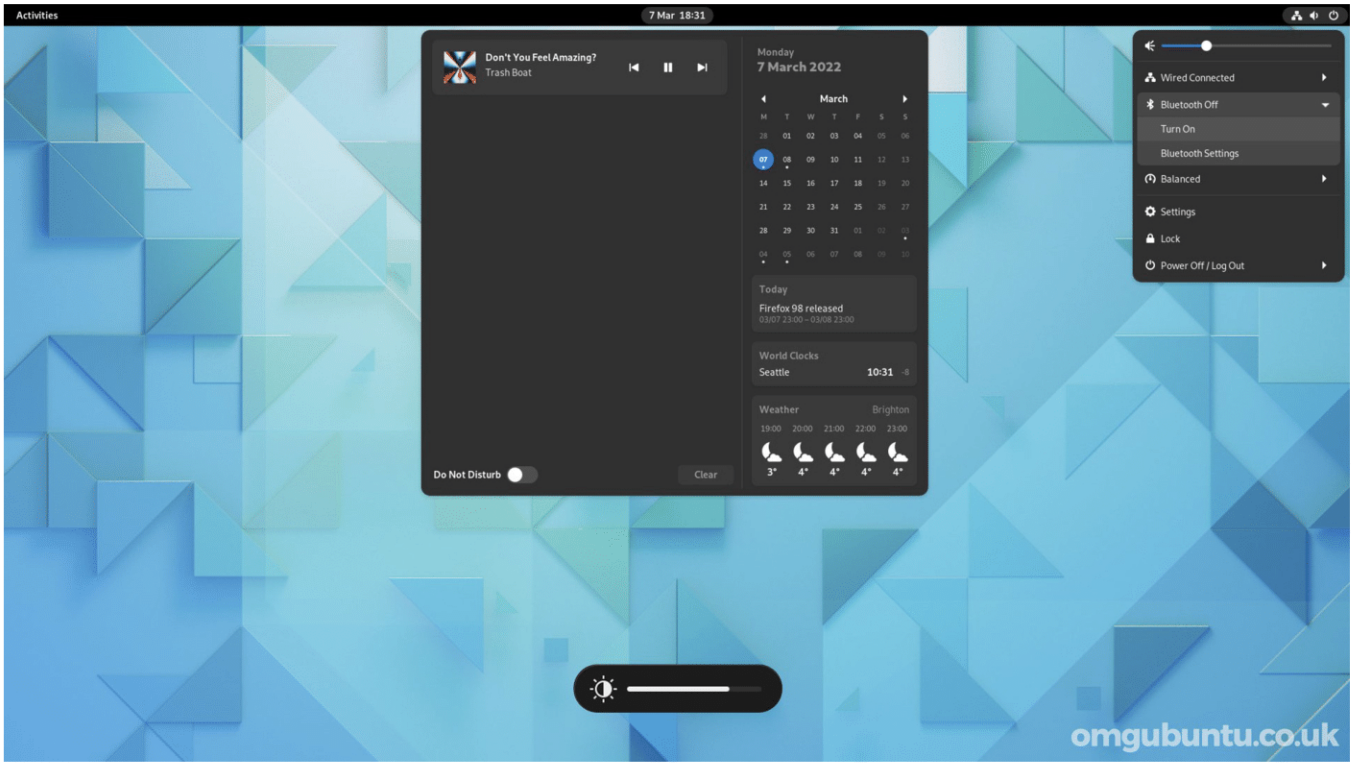
Documentation creating a Gnome Shell extension

First, what is GNOME?

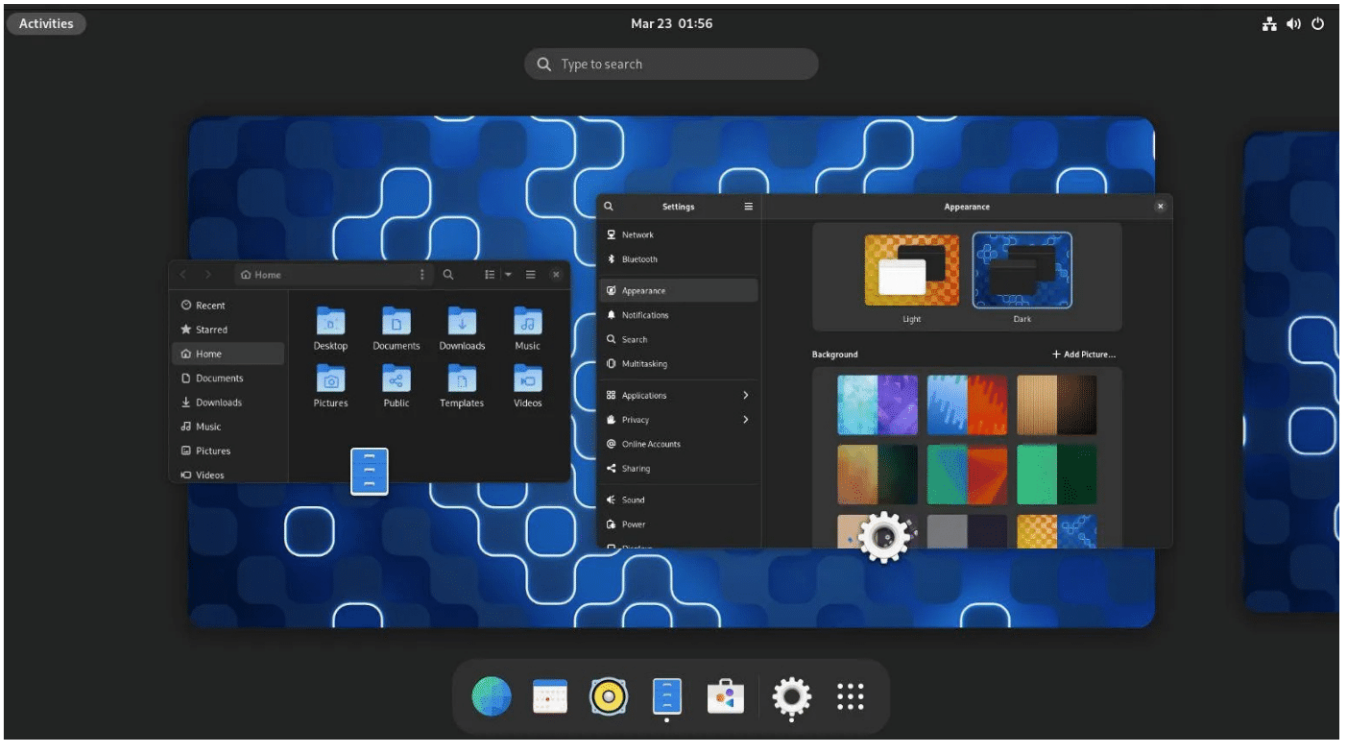
GNOME (GNU Network Object Model Environment) is a desktop environment for GNU/Linux platforms and UNIX systems, it allows you to use the various features of a PC via a graphical interface (terminal, text editor, file manager, task manager ...)

GNOME is the default environment installed on recent versions of Ubuntu, which is the distribution chosen for our touchscreens.



Great, but what is GNOME Shell?

It is the heart of the GNOME environment's graphical interface. Among other things, the GNOME Shell defines the GNOME dashboard, notification area and window selector.



GNOME Shell uses the Wayland or Xorg windowing system to work. For our extension and for our touchscreens we need to use Xorg (we need it for the screen flipping offered by `xrandr` and to install some plugins to better handle multi-touch).

GNOME Shell also includes a system of extensions (written in JavaScript) that allow you to customize the GNOME Shell and add/modify features. There is also a web service (https://extensions.gnome.org/) where any developer can propose and share their extensions.

We will therefore use this system to develop a button accessible in the top bar (accessible at any time), and which will allow the user of the touch tablet to turn the content displayed so that it is always well oriented.

Let's get to the heart of the matter, how do you code a GNOME Shell extension?

Inspired by: https://gjs.guide/extensions/overview/anatomy.html#extension-meta-object (https://gjs.guide/extensions/overview/anatomy.html#extension-meta-object)

Any Gnome Shell extension is composed of 2 files to be placed in `~/.local/share/gnome-shell/extensions/[ExtensionFolderName]/` :

- metadata.json : some fields are mandatory including :

```
"uuid": "example@vincent",
"name": "Example",
"description": "Extension description.",
"shell-version": [ "3.38", "40" ],
"url": "https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INFO4/22-23/1
"version": 1
```

  - *uuid* : unique identification field of the extension, broken down into 2 parts by @. It is recommended to have a small text describing the purpose of the extension on the left and a domain name (if available) on the right. This name must be identical to the name of the folder where the files of the extension are located.
  - *name*: short descriptive name of the extension.
  - *description*: a little longer description of the extension, you can use the \n and \t.
  - *shell-version* : list of strings to describe the gnome versions supported by the extension. For versions up to GNOME 3.38, both version components are required, from GNOME 40 onwards this is no longer necessary.
  - *url* : an address to a git repo where the source code is available and problems can be reported. Not mandatory if the goal is not to propose the extension on the web service.
- extension.js : heart of the extension. It contains three function hooks `init()` , `enable()` , `disable()` used by GNOME Shell to load and (de)activate the application. The rest is free, for example you can call the Streamlit (ST) libraries to program a button and ui.main to place it in the GNOME Shell interface.

You can also add a css file to change the visual of the extension widgets or any GNOME Shell.

Finally, we can also define a prefs.js file to modify the extension parameters from the following page: https://extensions.gnome.org/local/ (after installing an extension in the browser and a connector to the host (API to access local applications from the browser)

Once the 2 files are programmed, refresh gnome-shell by pressing Alt+F2, then enter 'r' in the proposed field and press Enter to validate.

Then activate the extension by running the following command:

```
gnome-extensions enable [NomDsMetadata]
```

Finally refresh gnome-shell as seen before, your extension is now usable.

Programming a button to turn the screen 90 degrees

We followed the previous steps, but the function called by the button (programmed in js via the St library) is the following:

```
function _rotateButton () {
Util.spawnCommandLine("bash /home/vincentdcr/rotateScript.sh")
}
```

So we run a bash script that will use xrandr (a command line tool that uses an extension of the Xorg windowing system to modify the display parameters of a screen). In particular, using the following command will turn the screen to the left: `xrandr -o left`

In our case, we will also need to return the digitizer's interpretation of our inputs at the same time (otherwise, the buttons are "physically" always in the same place even when the table has been rotated). To do this we need to execute this line: `xinput set-prop <peripherical_which_manages_the_tactile> --type=float "Coordinate Transformation Matrix" $coords` where the variable coords corresponds to a transformation matrix which is as follows:

Orientation	Matrix (3x3 in a row)
Left	"0 -1 1 1 0 0 0 0 1"
Inverted	"-1 0 1 0 -1 1 0 0 1"
Right	"0 1 0 -1 0 1 0 0 1"
Normal	"0 0 0 0 0 0 0 0 0"