

# Wayland Desktop Environments

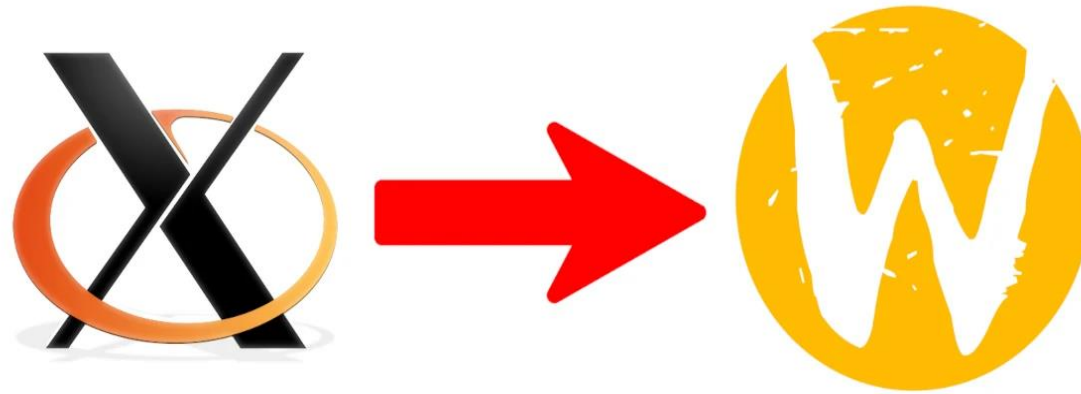
Weston on MA35

江天文

2024/08/15

Joy of innovation  
**nuvoTon**

# | Introduction to Wayland



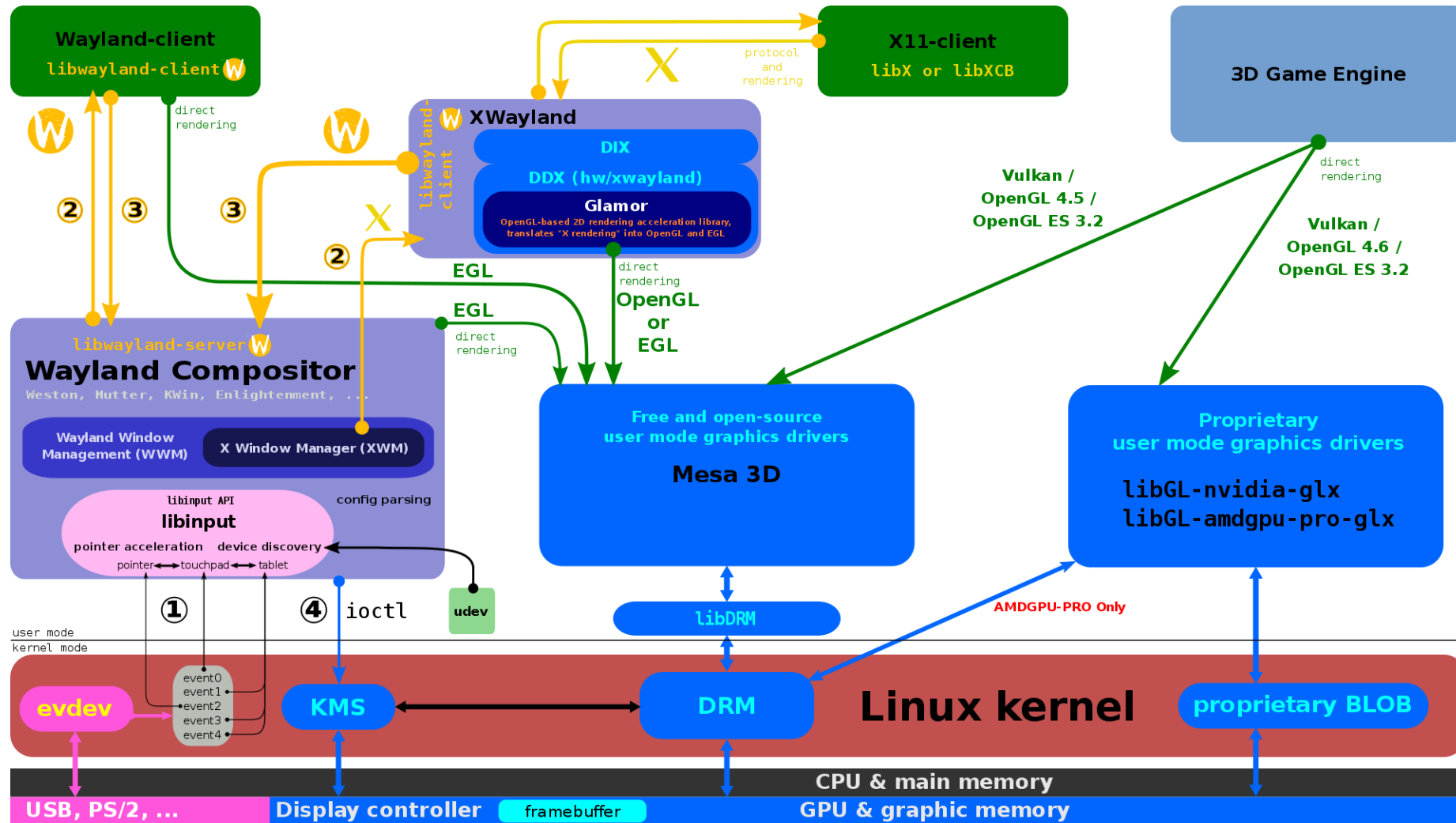
<https://wayland.freedesktop.org>

# | Wayland Protocol

- Wayland is a communication protocol that specifies the communication between a display server and its client, as well as a C library implementation of that protocol. A display server using the Wayland protocol is called a **Wayland compositor**, because it additionally performs the task of a compositing window manager.

Referral link: [https://en.wikipedia.org/wiki/Wayland\\_\(protocol\)](https://en.wikipedia.org/wiki/Wayland_(protocol))

# Wayland Architecture



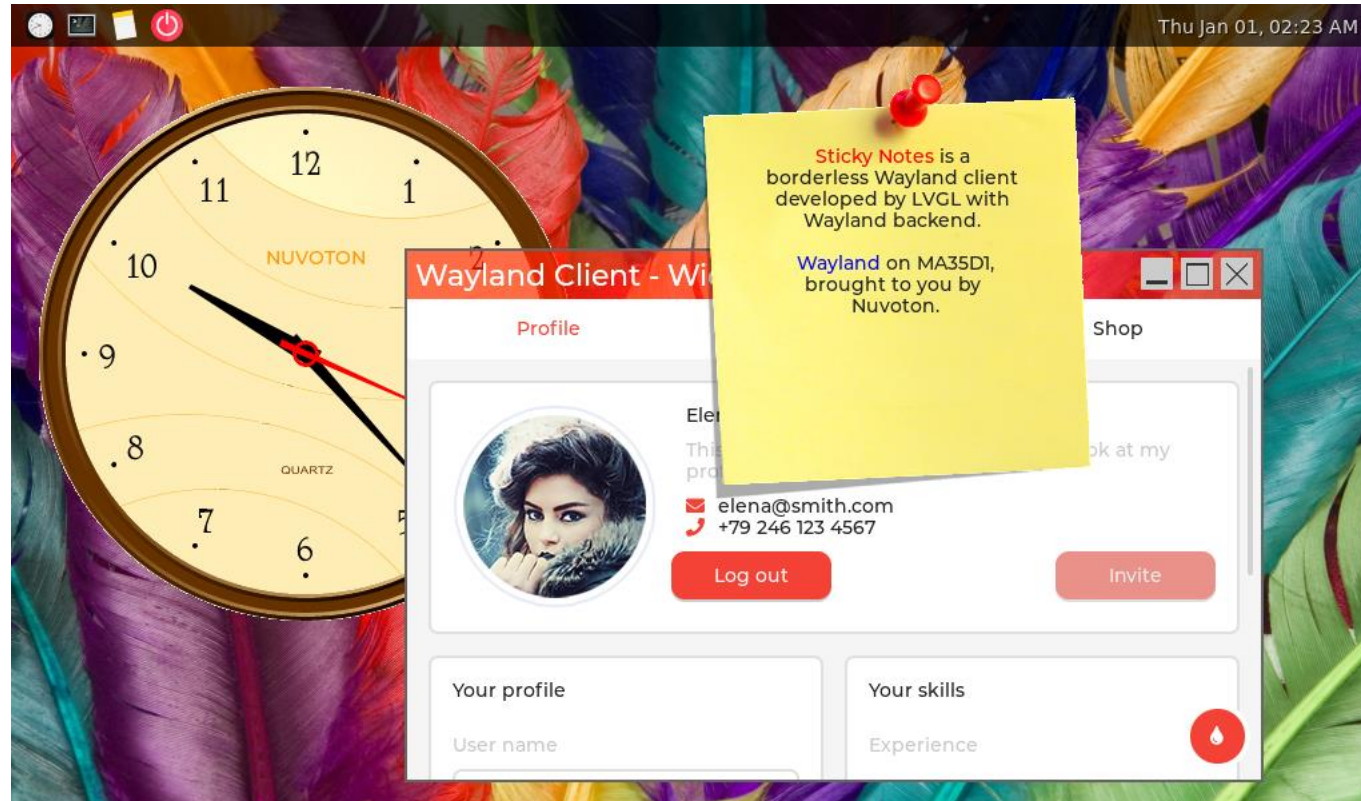
# | Weston Compositor

- Weston is the reference implementation of a Wayland compositor. When running on Linux, handling of the input hardware relies on evdev. Out of the box, Weston provides a very basic desktop, or a full-featured environment for non-desktop uses such as automotive, embedded, in-flight, industrial, kiosks, set-top boxes and TVs.





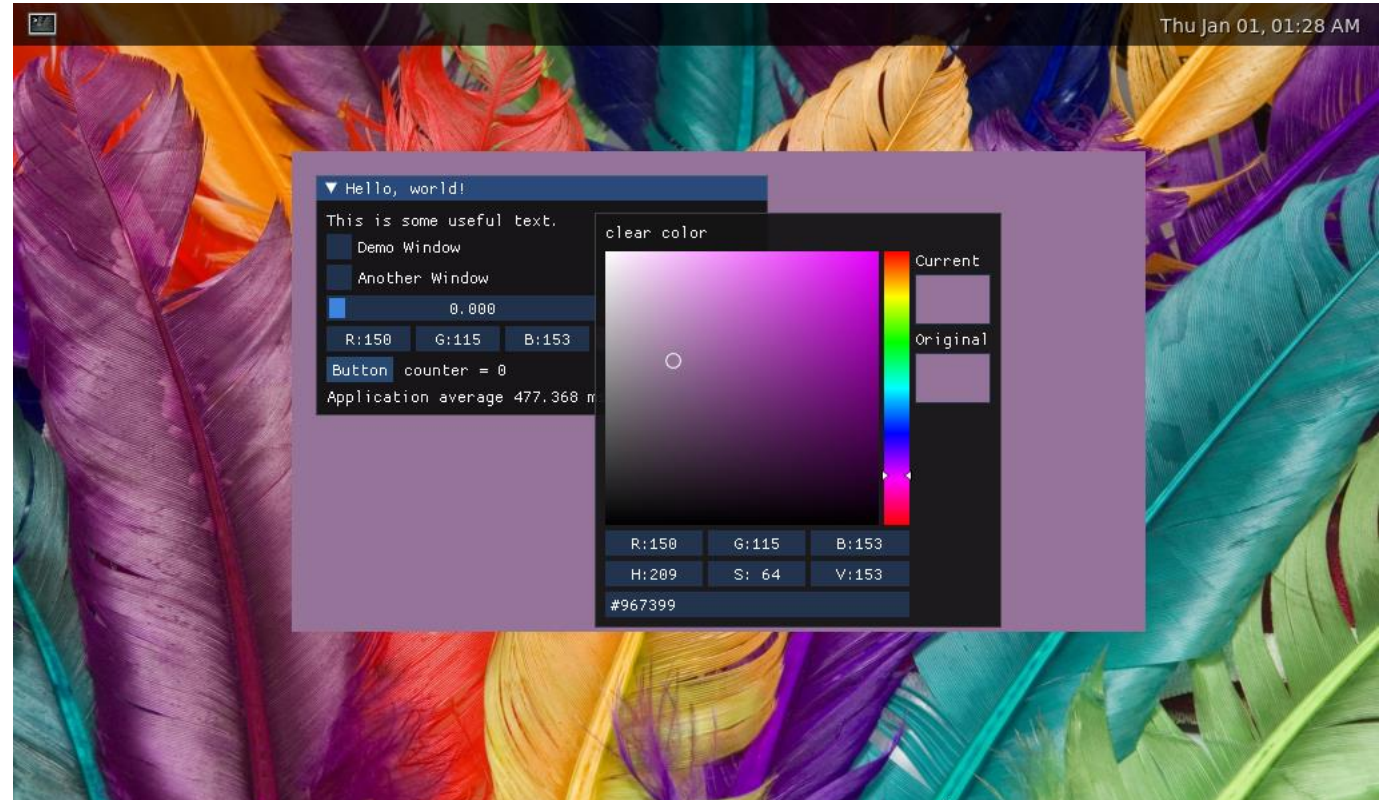
# | Customizing Weston



changing wallpaper, adding launch icons, setting idle lock time

# Changing Wallpaper

- Prepare a wallpaper on target location  
*`/usr/share/weston/wall.jpg`*
- Edit *`/etc/xdg/weston/weston.ini`*,  
under section 'shell',  
modify the entry  
'background-image'  
`[shell]`  
`background-image=/usr/share/weston/wall.jpg`



# | Modifying Task Panel Color

- Edit */etc/xdg/weston/weston.ini*, under section 'shell', modify the entry 'panel-color'

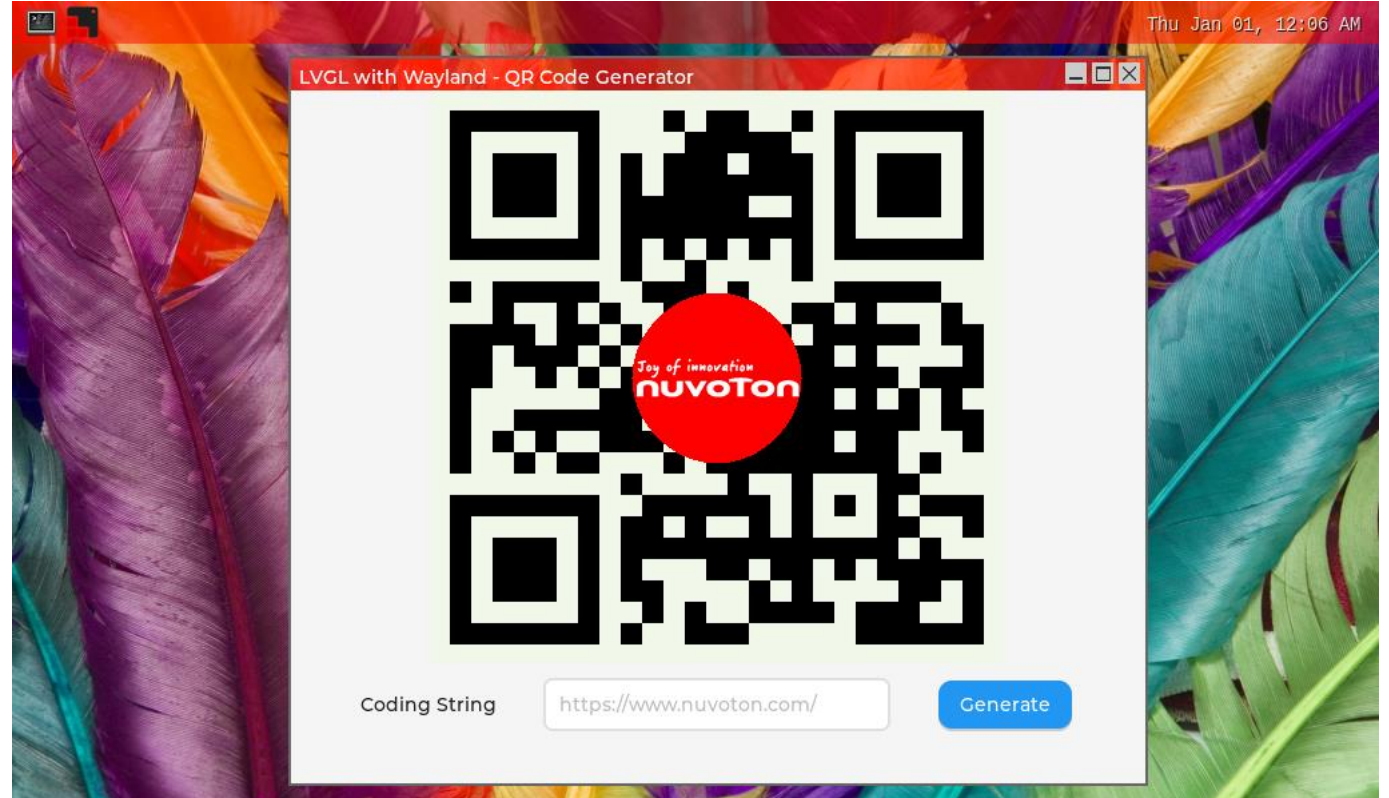
[shell]

panel-color=0xffff0000

- Add translucent color

[shell]

panel-color=0x80ff0000





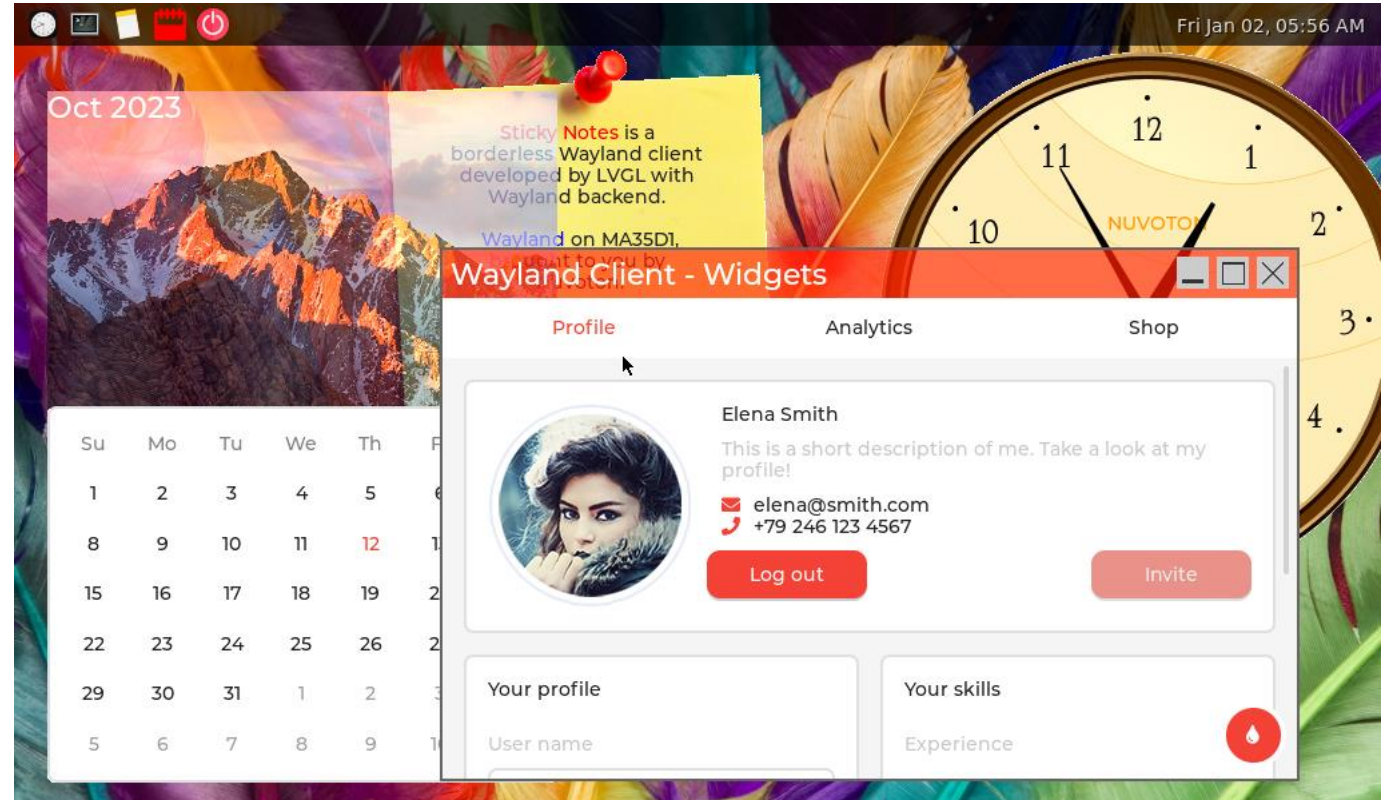
# | Adding Launch Icons in Task Panel

- Edit `/etc/xdg/weston/weston.ini`, add section 'launcher', append entries 'icon' and 'path' under the section 'launcher'.

[launcher]

icon=/usr/share/weston/wayland24.png

path=/usr/bin/lvgl\_fb



# | Setting Idle Lock Time

- Edit */etc/xdg/weston/weston.ini*, under section '**core**', modify the entry '**idle-time**' (unit: seconds)

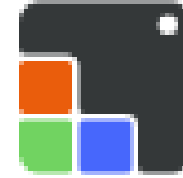
[core]

idle-time=3000



# | Toolkits with Wayland support

- Light and Versatile Graphics Library  
<https://lvgl.io>
- The Qt Group (Qt5)  
<https://www.qt.io>
- The GTK Project (GTK3)  
<https://www.gtk.org>
- The Simple DirectMedia Layer (SDL2)  
<https://libsdl.org>





# | Shaped Windows

- Analog Clock implemented with various widget toolkits GTK3/Qt5/LVGL





# LVGL with Wayland



# Light and Versatile Graphics Library

LVGL is the most popular free and open-source embedded graphics library to create beautiful UIs for any MCU, MPU and display type.



# | Configuring LVGL with Wayland in Buildroot

- Fetch the Buildroot 2024 ([https://github.com/OpenNuvoton/buildroot\\_2024.git](https://github.com/OpenNuvoton/buildroot_2024.git))
- Load a specific board configuration (\$ make list-defconfigs | grep ma35)

→ System configuration → /dev management

(X) Dynamic using devtmpfs + eudev

→ Target packages → Graphics libraries and applications

[\*] mesa3d

[\*] Gallium swrast driver

[\*] OpenGL EGL


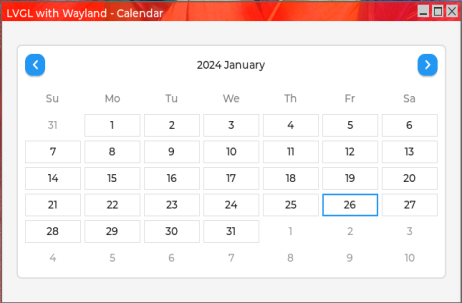
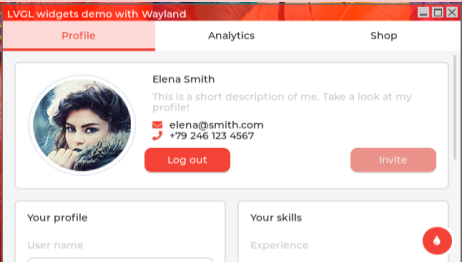
[\*] OpenGL ES

[\*] weston

default compositor (fbdev)

[\*] lvgl 8.3.0 with Wayland modified by Nuvoton

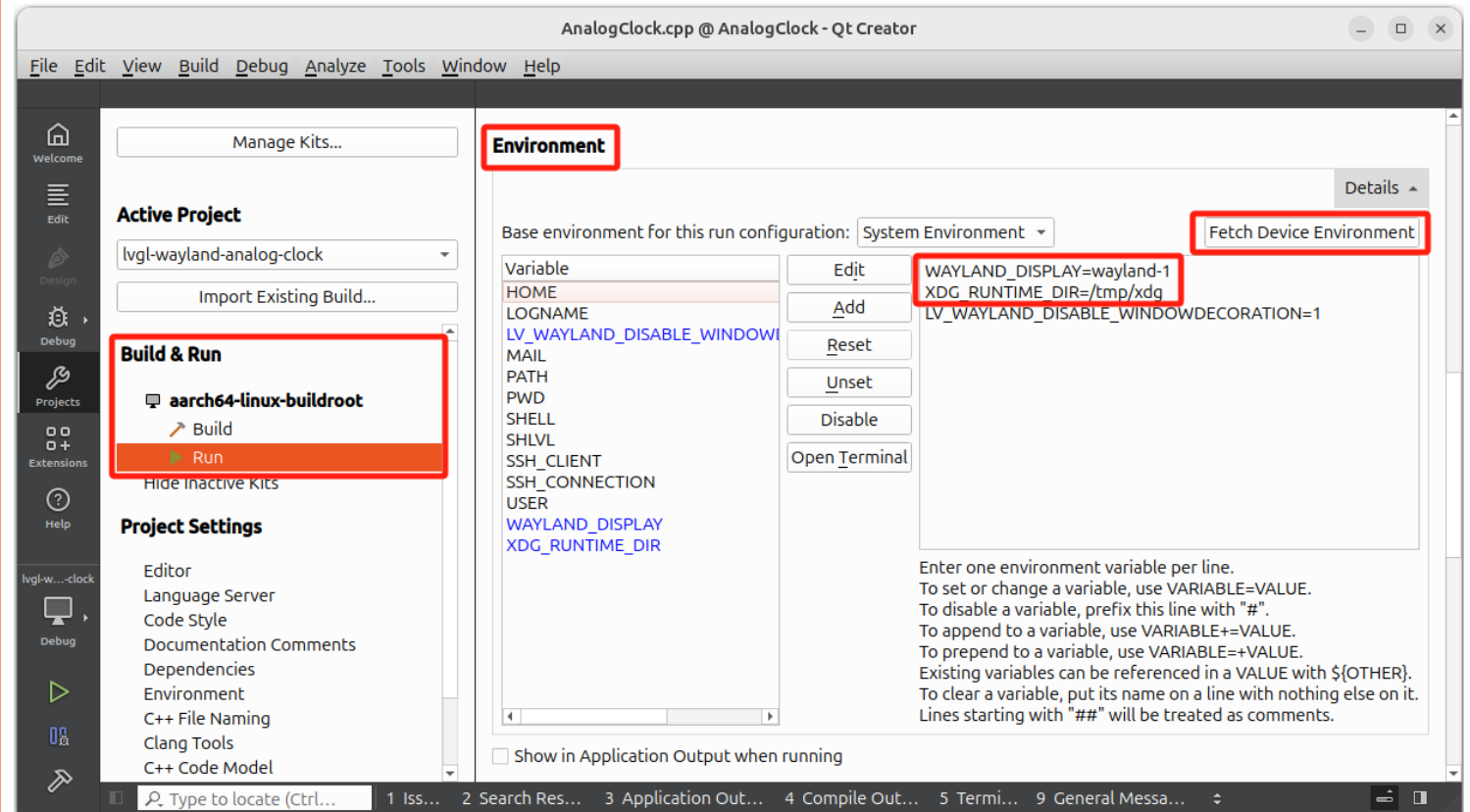
# LVGL with Wayland examples

Apps	Location ( <a href="https://github.com/symfund/ma35d1-portal">https://github.com/symfund/ma35d1-portal</a> ) \$ git clone -b weston-10.0.5 <a href="https://github.com/symfund/ma35d1-portal.git">https://github.com/symfund/ma35d1-portal.git</a>	Thumbnails
Minimal	examples/wayland/lvgl/lvgl-wayland-minimal	
Calendar	examples/wayland/lvgl/lvgl-wayland-calendar	
Widgets	examples/wayland/lvgl/lvgl-wayland-widgets-demo	

# Remote Debugging LVGL Apps with Qt Creator

- Modify *CMakeLists.txt*, change the environment variable **BR2\_DIR** to point to the actual root path of *Buildroot*.
- Launch *Qt Creator* to open project with *CMakeLists.txt*.
- Configure the settings of **Build & Run**, fetch device environment, and add two variables:

**WAYLAND\_DISPLAY**=wayland-1  
**XDG\_RUNTIME\_DIR**=/tmp/xdg





# | Qt5 with Wayland

## Qt Framework

The complete software development framework

The Qt framework contains a comprehensive set of highly intuitive and modularized C++ library classes and is loaded with APIs to simplify your application development. Qt produces highly readable, easily maintainable and reusable code with high runtime performance and small footprint – and it's cross-platform.

[Try Qt for Free](#)

[Buy Qt](#)



# | Configuring Qt5 with Wayland in Buildroot

- Fetch the Buildroot 2024 ([https://github.com/OpenNuvoton/buildroot\\_2024.git](https://github.com/OpenNuvoton/buildroot_2024.git))
- Load a specific board configuration (\$ make list-defconfigs | grep ma35)

→System configuration →/dev management  
(X) Dynamic using devtmpfs + eudev

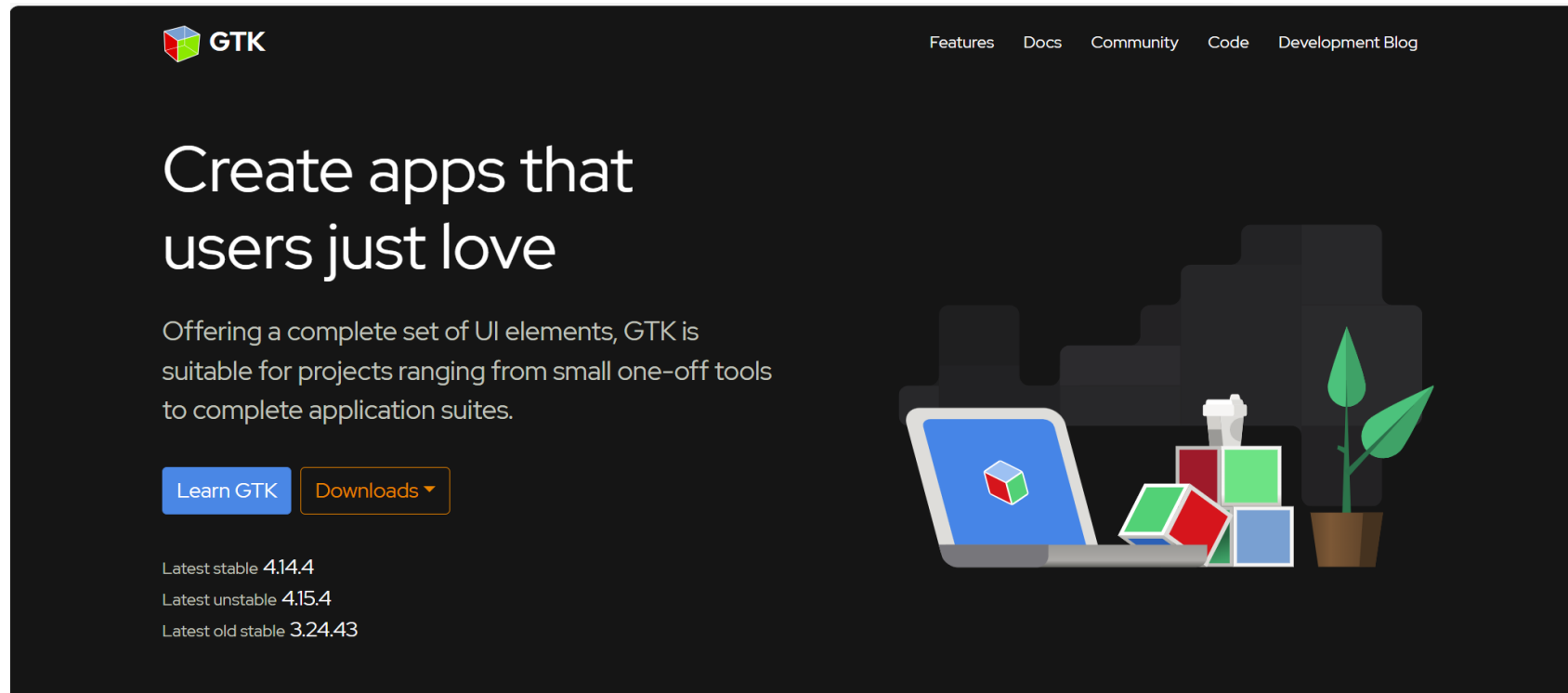
→Target packages →Fonts, cursors, icons, sounds and themes

- [\*] Liberation (Free fonts)
- [\*] mono fonts
- [\*] sans fonts
- [\*] serif fonts

→Target packages →Graphics libraries and applications

- [\*] weston  
    default compositor (fbdev)
- [\*] Qt5
  - \*- qt5base
    - [\*] gui module
    - [\*] widgets module
    - Default graphical platform (wayland)
    - [\*] fontconfig support
    - [\*] harfbuzz support
    - [\*] JPEG support
    - [\*] PNG support
    - [\*] qt5wayland

# | GTK3 with Wayland



# | Configuring GTK3 with Wayland in Buildroot

- Fetch the Buildroot 2024 ([https://github.com/OpenNuvoton/buildroot\\_2024.git](https://github.com/OpenNuvoton/buildroot_2024.git))
- Load a specific board configuration (\$ make list-defconfigs | grep ma35)

→ System configuration → /dev management  
(X) Dynamic using devtmpfs + eudev

→ Target packages → Graphics libraries and applications  
[\*] weston  
    default compositor (fbdev)

→ Target packages → Libraries → Graphics  
[\*] libgtk3  
    [\*] Wayland GDK backend  
    [\*] Install libgtk3 demo program  
    [\*] Install libgtk3 tests



# | GTK3 with Wayland example (minimal app)

- The project structure of GTK 3 with Wayland example (minimal app)

```
---gtk3app
    |__ CMakeLists.txt
    |__ main.c
```

- Change the environment variable **BR2\_DIR** to point to the actual root path of **Buildroot**, then launch Qt Creator to open project with **CMakeLists.txt**.

## gtk3app/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)
project(gtk3app LANGUAGES C)
set(BR2_DIR "/home/arthur/Projects/buildroot_2024")
set(CMAKE_SYSROOT "${BR2_DIR}/output/host/aarch64-nuvoton-linux-gnu/sysroot")

include_directories(
    ${CMAKE_SYSROOT}/usr/include/gtk-3.0
    ${CMAKE_SYSROOT}/usr/include/glib-2.0
    ${CMAKE_SYSROOT}/usr/lib/glib-2.0/include
    ${CMAKE_SYSROOT}/usr/include/pango-1.0
    ${CMAKE_SYSROOT}/usr/include/harfbuzz
    ${CMAKE_SYSROOT}/usr/include/cairo
    ${CMAKE_SYSROOT}/usr/include/gdk-pixbuf-2.0
    ${CMAKE_SYSROOT}/usr/include/atk-1.0)

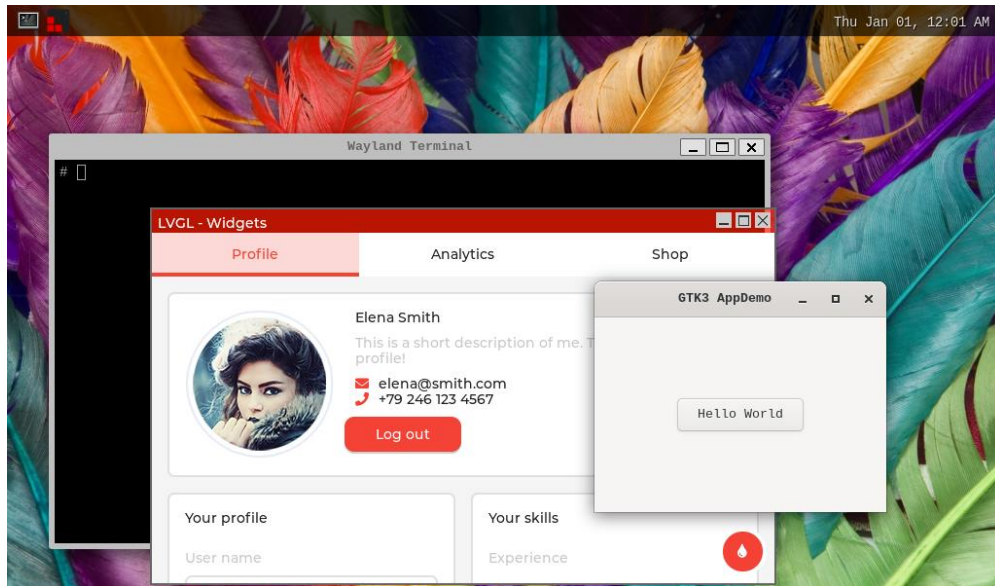
add_executable(gtk3app
    main.c
)

target_link_libraries(
    gtk3app
    glib-2.0
    gtk-3
    gdk_pixbuf-2.0
    gobject-2.0 gio-2.0
    pango-1.0
)

include(GNUInstallDirs)
install(TARGETS gtk3app
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

# | GTK3 with Wayland example (minimal app)

- The GTK 3 minimal app running in Wayland desktop environment



## gtk3app/main.c

```
#include <gtk/gtk.h>

static void button_clicked(GtkWidget *widget, gpointer data) {
    g_print("Hello World\n");
}

static void activate(GtkApplication *app, gpointer user) {
    GtkWidget *widget, *button, *button_box;

    widget = gtk_application_window_new(app);
    gtk_window_set_title(GTK_WINDOW(widget), "GTK3 AppDemo");
    gtk_window_set_default_size(GTK_WINDOW(widget), 300, 200);
    button_box = gtk_button_box_new(GTK_ORIENTATION_HORIZONTAL);
    gtk_container_add(GTK_CONTAINER(widget), button_box);
    button = gtk_button_new_with_label("Hello World");
    g_signal_connect(button, "clicked", G_CALLBACK(button_clicked), NULL);
    g_signal_connect_swapped(button, "clicked", G_CALLBACK(gtk_widget_destroy), widget);
    gtk_container_add(GTK_CONTAINER(button_box), button);
    gtk_widget_show_all(widget);
}

int main(int argc, char *argv[]) {
    GtkApplication *app;
    int status;

    app = gtk_application_new("com.nuvoton.gtk3app", G_APPLICATION_FLAGS_NONE);
    g_signal_connect(app, "activate", G_CALLBACK(activate), NULL);
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}
```

# | SDL2 with Wayland

Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. It is used by video playback software, emulators, and popular games including Valve's award winning catalog and many Humble Bundle games.

Official site: <https://libsdl.org>



# | Configuring SDL2 with Wayland in Buildroot

- Fetch the Buildroot 2024 ([https://github.com/OpenNuvoton/buildroot\\_2024.git](https://github.com/OpenNuvoton/buildroot_2024.git))
- Load a specific board configuration (\$ make list-defconfigs | grep ma35)

→ System configuration → /dev management  
(X) Dynamic using devtmpfs + eudev

→ Target packages → Graphics libraries and applications

```
[*] mesa3d
[*]   Gallium swrast driver
[*]   OpenGL EGL
[*]   OpenGL ES
[*] weston
    default compositor (fbdev)
[*] sdl2
[*]   OpenGL ES
[*] sdl2_gfx
[*] sdl2_image
[*] sdl2_ttf
```



# | SDL2 with Wayland example

- The project structure of SDL2 with Wayland example

---SDL2

|\_\_ CMakeLists.txt  
|\_\_ main.c

- Change the environment variable **BR2\_DIR** to point to the actual root path of **Buildroot**, then launch Qt Creator to open project with **CMakeLists.txt**.

## SDL2/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)

project(sdl2-wayland LANGUAGES C)

set(BR2_DIR "/home/arthur/Projects/buildroot_2024")
set(CMAKE_SYSROOT "${BR2_DIR}/output/host/aarch64-nuvoton-linux-gnu/sysroot")

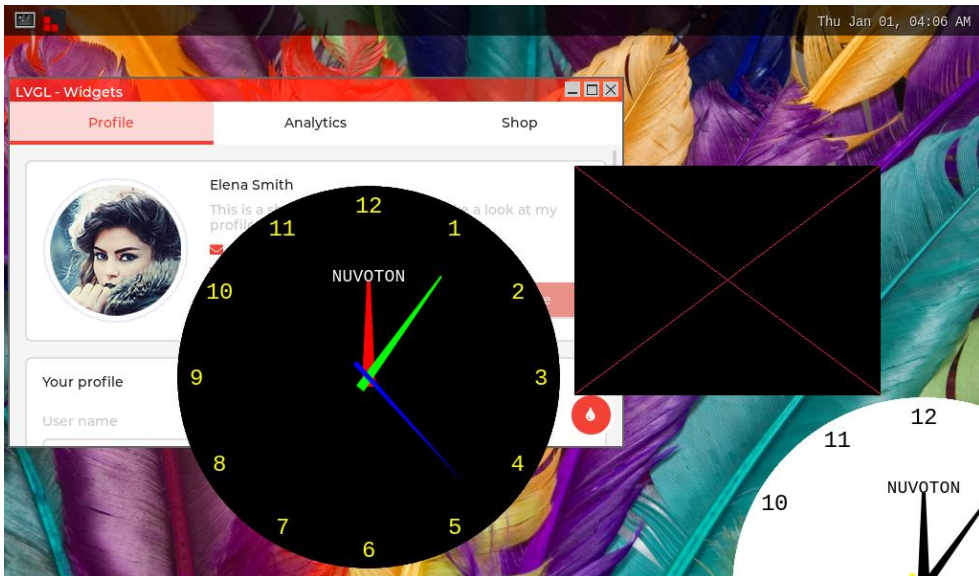
add_executable(sdl2-wayland
    main.c
)

target_link_libraries(sdl2-wayland
    SDL2
    wayland-client
)

include(GNUInstallDirs)
install(TARGETS sdl2-wayland
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

# | SDL2 with Wayland example

- The SDL2 with Wayland demo running in Wayland desktop environment



SDL2/main.c

```
#include <stdbool.h>
#include <SDL2/SDL.h>

int main(int argc, char* argv[]) {
    SDL_Window* window = NULL;
    SDL_Renderer* renderer = NULL;
    bool quit = true;

    if (SDL_Init(SDL_INIT_VIDEO) < 0) return 1;

    window = SDL_CreateWindow("SDL Tutorial", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
        640, 480, SDL_WINDOW_SHOWN);
    if (window == NULL) {
        SDL_Quit();
        return 1;
    }

    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
    if (renderer == NULL) {
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 1;
    }

    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
    SDL_RenderClear(renderer);

    while (quit) {
        SDL_SetRenderDrawBlendMode(renderer, SDL_BLENDMODE_ADD);
        SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
        SDL_RenderDrawLine(renderer, 0, 0, 639, 479);
        SDL_RenderDrawLine(renderer, 639, 0, 0, 479);
        SDL_RenderPresent(renderer);
        SDL_Delay(16);
    }

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

*Joy of innovation*  
**nuvoTon**

谢谢

謝謝

Děkuji

Bedankt

Thank you

Kiitos

Merci

Danke

Grazie

ありがとう

감사합니다

Dziękujemy

Obrigado

Спасибо

Gracias

Teşekkür ederim

Cảm ơn