Wayland

Wayland (https://wayland.freedesktop.org/) is a display server protocol. It is aimed to become the successor of the **X Window System**. You can find a **comparison between Wayland and Xorg on Wikipedia**.

Display servers using the Wayland protocol are called **compositors** because they also act as **compositing window managers**. Below you can find a **list of Wayland compositors**.

Related articles

- KMS
- Xorg
- Screen capture#Wayland

For compatibility with native X11 applications to run them seamlessly, **Xwayland** can be used, which provides an X Server in Wayland.

1 Requirements

Most Wayland compositors only work on systems using **Kernel mode setting**. Wayland by itself does not provide a graphical environment; for this you also need a compositor (see the following section), or a desktop environment that includes a compositor (e.g. **GNOME** or **Plasma**).

For the GPU driver and Wayland compositor to be compatible they must support the same buffer API. There are two main APIs: **GBM** and **EGLStreams (https://www.phoronix.com/scan.php?page=news_item&px=XDC2016-Device-Memory-API)**.

Buffer API	GPU driver support	Wayland compositor support
GBM	All except NVIDIA < 495*	All
EGLStreams	NVIDIA	GNOME

* NVIDIA ≥ 495 supports both EGLStreams and GBM.**[1] (https://www.phoronix.com/scan.php?page=news_item&px=NVIDIA-495.44-Linux-Driver)**

Since NVIDIA introduced GBM support, many compositors (including Mutter and KWin) started using it by default for NVIDIA ≥ 495. GBM is generally considered better with wider support, and EGLStreams only had support because NVIDIA did not provide any alternative way to use their GPUs under Wayland with their proprietary drivers. Furthermore, KWin **dropped support for EGLStreams (https://invent.kde.org/plasma/kwin/-/merge_requests/1638)** after GBM was introduced into NVIDIA.

If you use a popular desktop environment/compositor and a GPU still supported by NVIDIA, you are most likely already using GBM backend. To check, run `journalctl -b 0 --grep "renderer for"`. To force GBM as a backend, set the following **environment variables**:

```
GBM_BACKEND=nvidia-drm
__GLX_VENDOR_LIBRARY_NAME=nvidia
```

2 Compositors

See Window manager#Types for the difference between **Stacking**, **Tiling** and **Dynamic**.

2.1 Stacking

- **COSMIC Compositor** — Compositor for the COSMIC desktop environment.

  https://github.com/pop-os/cosmic-comp || cosmic-comp (https://archlinux.org/packages/?name=cosmic-comp)

- **Enlightenment** — See Enlightenment#Manually. More Info: [2] (https://git.enlightenment.org/enlightenment/enlightenment/src/branch/master/README.md) [3] (https://www.enlightenment.org/about-wayland)

  https://www.enlightenment.org/ || enlightenment (https://archlinux.org/packages/?name=enlightenment)

- **hikari** — wlroots-based compositor inspired by cwm which is actively developed on FreeBSD but also supports Linux.

  https://web.archive.org/web/20241220075628/https://hikari.acmelabs.space/ || hikari (https://aur.archlinux.org/packages/hikari/)[AUR]

- **KDE KWin** — See KDE#Starting Plasma.

  https://userbase.kde.org/KWin || kwin (https://archlinux.org/packages/?name=kwin)

- **Liri Shell** — Part of Liri, built using QtQuick and QtCompositor as a compositor for Wayland.

  https://github.com/lirios/shell || liri-git-meta (https://aur.archlinux.org/packages/liri-git-meta/)[AUR]

- **waybox** — a *box-style (minimalist) Wayland compositor modeled largely on Openbox

  https://github.com/wizbright/waybox || waybox (https://aur.archlinux.org/packages/waybox/)[AUR]

- **labwc** — wlroots-based compositor inspired by Openbox.

  https://github.com/labwc/labwc || labwc (https://archlinux.org/packages/?name=labwc)

- **Mutter** — See GNOME#Starting.

  https://gitlab.gnome.org/GNOME/mutter || mutter (https://archlinux.org/packages/?name=mutter)

- **wayfire** — 3D compositor inspired by Compiz and based on wlroots.

  https://wayfire.org/ || wayfire (https://aur.archlinux.org/packages/wayfire/) AUR

- Weston — Wayland compositor designed for correctness, reliability, predictability, and performance.

  https://gitlab.freedesktop.org/wayland/weston || weston (https://archlinux.org/packages/?name=weston)

- **wio** — wlroots-based compositor that aims to replicate the look and feel of Plan 9's Rio desktop.

  https://gitlab.com/Rubo/wio || wio-wl (https://aur.archlinux.org/packages/wio-wl/) AUR

- **wlmaker** — wlroots-based compositor that's inspired by Window Maker.

  https://github.com/phkaeser/wlmaker || wlmaker (https://aur.archlinux.org/packages/wlmaker/) AUR

2.2 Tiling

- Cagebreak — Based on cage, inspired by ratpoison.

  https://github.com/project-repo/cagebreak || cagebreak (https://aur.archlinux.org/packages/cagebreak/) AUR

- **miracle-wm** — A Wayland compositor based on Mir in the style of i3 and sway with the intention to be flashier and more feature-rich than either, like swayfx.

  https://github.com/miracle-wm-org/miracle-wm || miracle-wm (https://aur.archlinux.org/packages/miracle-wm/) AUR

- niri — A scrollable-tiling Wayland compositor.

  https://github.com/YaLTeR/niri/ || niri (https://archlinux.org/packages/?name=niri)

- Qtile — A full-featured, hackable tiling window manager and Wayland compositor written and configured in Python.

  https://github.com/qtile/qtile || qtile (https://archlinux.org/packages/?name=qtile)

- Sway — i3-compatible Wayland compositor based on wlroots.

  https://github.com/swaywm/sway || sway (https://archlinux.org/packages/?name=sway)

- **SwayFx** — Sway, but with eye candy!

  https://github.com/WillPower3309/swayfx || swayfx (https://aur.archlinux.org/packages/swayfx/)[AUR]

- **Velox** — Simple window manager based on swc, inspired by dwm and xmonad.

  https://github.com/michaelforney/velox || velox-git (https://aur.archlinux.org/packages/velox-git/)[AUR]

2.3 Dynamic

- cwc — awesome-like Wayland compositor based on wlroots.

  https://cudiph.github.io/cwc/apidoc/ || cwc (https://aur.archlinux.org/packages/cwc/)[AUR]

- dwl — dwm-like Wayland compositor based on wlroots.

  https://codeberg.org/dwl/dwl || dwl (https://aur.archlinux.org/packages/dwl/)[AUR]

- Hyprland — A dynamic tiling Wayland compositor that does not sacrifice on its looks.

  https://hypr.land || hyprland (https://archlinux.org/packages/?name=hyprland)

- **japokwm** — Dynamic Wayland tiling compositor based around creating layouts, based on wlroots.

  https://github.com/werererer/japokwm || japokwm-git (https://aur.archlinux.org/packages/japokwm-git/)[AUR]

- river — Dynamic tiling Wayland compositor inspired by dwm and bspwm.

  https://codeberg.org/river/river || river (https://archlinux.org/packages/?name=river)

- **Vivarium** — A dynamic tiling Wayland compositor using wlroots, with desktop semantics inspired by xmonad.

  https://github.com/inclement/vivarium || vivarium-git (https://aur.archlinux.org/packages/vivarium-git/)[AUR]

2.4 Other

- **Cage** — Displays a single fullscreen application like a kiosk.

  https://www.hjdskes.nl/projects/cage/ || cage (https://archlinux.org/packages/?name=cage)

- **phoc** — A tiny wlroots-based compositor for mobile devices.

  https://gitlab.gnome.org/World/Phosh/phoc || phoc (https://archlinux.org/packages/?name=phoc)

- **Wayback** — X11 compatibility layer which allows for running full X11 desktop environments using Wayland components.

  https://wayback.freedesktop.org/ || wayback-x11 (https://aur.archlinux.org/packages/wayback-x11/)[AUR]

Some of the above may support display managers. Check `/usr/share/wayland-sessions/compositor.desktop` to see how they are started.

3 Display managers

Display managers listed below support launching Wayland compositors.

Name	Runs on	Description
emptty (https://archlinux.org/packages/?name=emptty)	tty	Simple CLI Display Manager on TTY.
GDM	Wayland/Xorg	GNOME display manager.
greetd	Wayland/Xorg/tty See Greetd#Greeters.	Minimal and flexible login daemon.
lemurs (https://archlinux.org/packages/?name=lemurs)	tty	TUI display manager written in Rust.
lidm (https://aur.archlinux.org/packages/lidm/)[AUR]	tty	A fully colorful customizable TUI display manager made in C.
LightDM	Xorg[4] (https://github.com/canonical/lightdm/issues/267)	Cross-desktop display manager.
ly (https://archlinux.org/packages/?name=ly)	tty	TUI display manager written in C
SDDM	Wayland/Xorg	QML-based display manager.
tbsm (https://aur.archlinux.org/packages/tbsm/)[AUR]	tty	Simple CLI session launcher written in pure bash.
uwsm	tty	Session and XDG autostart manager for standalone compositors. Provides a TUI menu, but can also be used with other display managers.

4 Xwayland

Xwayland(1) (https://man.archlinux.org/man/Xwayland.1) is an X server that runs under Wayland and provides compatibility for native X11 applications that are yet to provide Wayland support. To use it, install the xorg-xwayland (https://archlinux.org/packages/?name=xorg-xwayland) package.

Xwayland is started via a compositor, so you should check the documentation for your chosen compositor for Xwayland compatibility and instructions on how to start Xwayland.

> **Note**
>
> - Security: Xwayland is an X server, so it does not have the security features of Wayland
> - Performance: Xwayland has a nearly identical performance (https://openbenchmarking.org/result/2202053-NE-NVIDIARTX35) to that of X11. In some cases you might notice degraded performance, especially on NVIDIA cards.
> - Compatibility: Xwayland is not fully backward compatible with X11. Some applications may not work properly under Xwayland.

4.1 NVIDIA driver

> **Note**
>
> NVIDIA drivers prior to version 470 (e.g. nvidia-390xx-dkms (https://aur.archlinux.org/packages/nvidia-390xx-dkms/)[AUR]) do not support hardware accelerated Xwayland, causing non-Wayland-native applications to suffer from poor performance in Wayland sessions.

Enabling DRM KMS is required. There may be additional information in the official documentation (https://download.nvidia.com/XFree86/Linux-x86_64/515.48.07/README/xwayland.html) regarding your display manager (e.g. GDM).

4.2 Kwin Wayland debug console

If you use kwin (https://archlinux.org/packages/?name=kwin), execute the following to see which windows use Xwayland or native Wayland, surfaces, input events, clipboard contents, and more.

```
$ qdbus6 org.kde.KWin /KWin org.kde.KWin.showDebugConsole
```

4.3 Detect Xwayland applications

To determine whether an application is running via Xwayland, you can run extramaus (https://aur.archlinux.org/packages/extramaus/)[AUR]. Move your mouse pointer over the window of an application. If the red mouse moves, the application is running via Xwayland.

Alternatively, you can use **xorg-xeyes (https://archlinux.org/packages/?name=xorg-xeyes)** and see if the eyes are moving, when moving the mouse pointer over an application window.

Another option is to run *xwininfo* (from **xorg-xwininfo (https://archlinux.org/packages/?name=xorg-xwininfo)**) in a terminal window: when hovering over an Xwayland window the mouse pointer will turn into a + sign. If you click the window it will display some information and end, but it will not do anything with native Wayland windows.You can use `Ctrl+C` to end it.

You can also use *xlsclients* (from the **xorg-xlsclients (https://archlinux.org/packages/?name=xorg-xlsclients)** package). To list all applications running via Xwayland, run `xlsclients -l` .

5 GUI libraries

5.1 GTK

The **gtk3 (https://archlinux.org/packages/?name=gtk3)** and **gtk4 (https://archlinux.org/packages/?name=gtk4)** packages have the Wayland backend enabled. GTK will default to the Wayland backend, but it is possible to override it to Xwayland by modifying an environment variable: `GDK_BACKEND=x11` .

For theming issues, see **GTK#Wayland backend**.

5.2 Qt

To enable Wayland support in **Qt** 5 or 6, install the **qt5-wayland (https://archlinux.org/packages/?name=qt5-wayland)** or **qt6-wayland (https://archlinux.org/packages/?name=qt6-wayland)** package, respectively. Qt applications will then run under Wayland on a Wayland session.

While it should not be necessary, to explicitly run a Qt application with the Wayland plugin **[5] (https://wiki.qt.io/QtWayland#How_do_I_use_QtWayland.3F)**, use `-platform wayland` or `QT_QPA_PLATFORM=wayland` **environment variable**.

To force the usage of **X11** on a Wayland session, use `QT_QPA_PLATFORM=xcb` .

This might be necessary for some proprietary applications that do not use the system's implementation of Qt. `QT_QPA_PLATFORM="wayland;xcb"` allows Qt to use the xcb (X11) plugin instead if Wayland is not available.**[6] (https://www.qt.io/blog/2018/05/29/whats-new-in-qt-5-11-for-the-wayland-platform-plugin)**

On some compositors, for example **sway**, Qt applications running natively might have missing functionality. For example, **KeepassXC (https://keepassxc.org)** will be unable to minimize to tray. This can be solved by installing **qt5ct (https://archlinux.org/packages/?name=qt5ct)** and setting `QT_QPA_PLATFORMTHEME=qt5ct` before running the application.

Due to the **Incorrect sizing and bad text rendering with WebEngine using fractional scaling on Wayland (https://bugreports.qt.io/browse/QTBUG-113574)** Qt WebEngine bug, applications using Qt WebEngine, for example **Calibre (https://bugs.launchpad.net/calibre/+bug/2018658)**, may display jagged fonts. A workaround is launching the application with `QT_SCALE_FACTOR_ROUNDING_POLICY=RoundPreferFloor`. This prevents the application window being fractional scaled.

5.3 Clutter

The Clutter toolkit has a Wayland backend that allows it to run as a Wayland client. The backend is enabled in the **`clutter` (https://archlinux.org/packages/?name=clutter)** package.

To run a Clutter application on Wayland, set `CLUTTER_BACKEND=wayland`.

5.4 SDL

In **SDL3**, Wayland is used by default to communicate with the desktop compositor.

To run an SDL2 application on Wayland, set `SDL_VIDEODRIVER=wayland`. `SDL_VIDEODRIVER="wayland,x11"` allows SDL2 to use the x11 video driver instead if Wayland is not available.**[7] (https://wiki.libsdl.org/SDL2/FAQUsingSDL)**. You may also want to install **`libdecor` (https://archlinux.org/packages/?name=libdecor)** to enable window decorations (for example, on GNOME).

Refer to the **official documentation (https://wiki.libsdl.org/SDL3/README/wayland)** for more details.

5.5 GLFW

The **`glfw` (https://archlinux.org/packages/?name=glfw)** package has support for Wayland, and uses the Wayland backend if the **environment variable** `XDG_SESSION_TYPE` is set to `wayland` and the application developer has not set a specific desired backend.

See the **source code (https://github.com/glfw/glfw/blob/3.4/src/platform.c#L87-L99)** for more information.

5.6 GLEW

If the **`glew-wayland-git` (https://aur.archlinux.org/packages/glew-wayland-git/)**[AUR] package does not work with the needed GLEW-based applications, the option is to use **`glew` (https://archlinux.org/packages/?name=glew)** with Xwayland. See **FS#62713 (https://bugs.archlinux.org/task/62713)**.

5.7 EFL

Enlightenment has **complete Wayland support (https://www.enlightenment.org/about-wayland)**.

To run an EFL-based application on Wayland, set `ELM_DISPLAY=wl`.

5.8 winit

Winit is a window handling library in Rust. It will default to the Wayland backend, but it is possible to override it to Xwayland by modifying environment variables:

- Prior to version 0.29.2, set `WINIT_UNIX_BACKEND=x11`
- For version 0.29.2 and higher, unset `WAYLAND_DISPLAY`, which forces a fallback to X using the `DISPLAY` variable. **[8] (https://github.com/rust-windowing/winit/blob/baf10de95843f156b0fbad6b10c3137f1ebd4f1e/src/changelog/v0.29.md?plain=1#L134)**

5.9 Electron

Wayland support can be activated either using per-application command line flags or more globally using a configuration file. Refer to **#Configuration file** for details.

To determine which electron version the application uses, see **[10] (https://stackoverflow.com/q/50345957)**.

> **Note**
>
> In Plasma, some Electron applications can use the wrong icon (default Wayland one) for the window, while using the correct icon for the taskbar. To fix that, you can create a special application/window rule, forcing the desktop file name on such applications.

5.9.1 Environment variable

Applications using Electron 28 and higher can use the **environment variable ELECTRON_OZONE_PLATFORM_HINT (https://www.electronjs.org/docs/latest/api/environment-variables#electron_ozone_platform_hint-linux)** set to `auto` or `wayland`.

This takes lower priority than the command line flags.

5.9.2 Command line flags

Unlike on Chromium which Electron is based on, Electron applications do not enable WebRTC screen capture over PipeWire by default. Using `--enable-features=WebRTCPipeWireCapturer` is therefore recommended to avoid screen capture problems on Wayland. The capture is based on **`xdg-desktop-portal (https://archlinux.org/packages/?name=xdg-desktop-portal)`**.

To use **`electron (https://archlinux.org/packages/?name=electron)`**-based applications natively under Wayland when using the environment variable is not desirable or feasible, `--ozone-platform-hint=auto` can be added on Electron 20+.

A case of missing top bars can be solved by using: `--enable-features=WaylandWindowDecorations`. This will typically be necessary under **GNOME** (supported since **electron17 (https://github.com/electron/electron/pull/29618)**).

You can set these flags more permanently by means of **modifying the .desktop file** of an application and adding the flags to the end of the `Exec=` line, or more cleanly by using the below-described configuration files.

> **Note**
>
> Some packages do not forward flags to Electron, and thus will need the application developer to implement a solution.

5.9.3 Configuration file

Electron packages read `~/.config/electronXX-flags.conf` files, where *XX* is Electron version, or fallback to shared `~/.config/electron-flags.conf`, if the versioned file is not present.

Put the previously mentioned flags one per line:

```
~/.config/electron-flags.conf
```

```
--enable-features=WaylandWindowDecorations
--ozone-platform-hint=auto
```

> **Note**
>
> These configuration files only work for the Electron packages in the official repositories and packages that use them. They do not work for packages that bundle their own build of Electron such as **slack-desktop (https://aur.archlinux.org/packages/slack-desktop/)**[AUR]. Sometimes alternatives exist such as **slack-electron (https://aur.archlinux.org/packages/slack-electron/)**[AUR].

5.9.4 Older Electron versions

`electron25-flags.conf` applies only to version 25 of Electron. Older versions of Electron can be configured using their own `electron<version>-flags.conf` file.

Older versions may also require different flags, depending on the corresponding Chromium version. For example, the following flags work on Electron 13:

```
~/.config/electron13-flags.conf
```

```
--enable-features=UseOzonePlatform
--ozone-platform=wayland
```

5.10 Java

The open source implementation of the **Java** platform OpenJDK, does not yet have native support for Wayland. Until **Wakefield (https://openjdk.java.net/projects/wakefield/)**, the project that aims to implement Wayland in OpenJDK, is available, Xwayland can be used.

See **Debian:Wayland#Java Programs (supported since OpenJDK 16?)**:

> Starting with OpenJDK 16, the JRE can dynamically load GTK3 (which has Wayland support), it appears this might be supported according to this **discussion (https://stackoverflow.com/questions/39197208/java-gui-support-on-wayland)**.
> The `_JAVA_AWT_WM_NONREPARENTING` **environment variable** can be set to "1" to fix misbehavior where the application starts with a blank screen.

Since XWayland doesn't have full feature parity with Wayland, **WLToolkit (https://wiki.openjdk.org/display/wakefield/Pure+Wayland+toolkit+prototype)** can be used to fill the gaps while Wakefield isn't ready. It can be activated with `-Dawt.toolkit.name=WLToolkit`. Some programs such as the **JetBrains IDEs support it (https://blog.jetbrains.com/platform/2024/07/wayland-support-preview-in-2024-2/)**.

6 Tips and tricks

6.1 Automation

- **ydotool (https://github.com/ReimuNotMoe/ydotool)** (**`ydotool (https://archlinux.org/packages/?name=ydotool)`**) - Generic command-line automation tool (not limited to wayland). **Enable/start** the `ydotool.service` **user unit**. See **`ydotoold(8) (https://man.archlinux.org/man/ydotoold.8)`**, **`ydotool(1) (https://man.archlinux.org/man/ydotool.1)`**.
- **wtype (https://github.com/atx/wtype)** (**`wtype (https://archlinux.org/packages/?name=wtype)`**) - xdotool type for wayland. See **`wtype(1) (https://man.archlinux.org/man/wtype.1)`**.
- **keyboard (https://github.com/boppreh/keyboard)** - Python library that works on Windows and Linux with experimental OS X support. Also see the **mouse (https://github.com/boppreh/mouse)** library.
- **wlrctl (https://git.sr.ht/~brocellous/wlrctl)** (**`wlrctl (https://aur.archlinux.org/packages/wlrctl/)`**[AUR]) - A command line utility for miscellaneous wlroots extensions (supports the foreign-toplevel-management, virtual-keyboard, virtual-pointer)

6.2 Remap keyboard or mouse keys

See **Input remap utilities**.

6.3 Screencast

See **Screen capture#Screencasting** and **Screen capture#Screencast Wayland windows with X11 applications**.

6.4 Chromium does not fully maximize

You have to enable *Use system title bar and borders* via the *chrome://settings/appearance* menu.

6.5 Persist clipboard after app close

Due to Wayland's design philosophy, clipboard data is stored in the memory of the source client. When the client closes, the clipboard data is lost. You can solve this using **wl-clip-persist (https://archlinux.org/packages/?name=wl-clip-persist)**, which runs in the background to reads the clipboard data and stores it in its own memory, separate from the source client.

6.6 Autostart wayland compositor as systemd service

> **Note**
>
> **Universal Wayland Session Manager** automatically generates systemd units for your compositors, moreover it helps you to **integrate graphical applications with systemd**

If you do not want to use a display manager or a shell, you can autostart your Wayland compositor with a **systemd** service. Adjust the `ExecStart` line with the compositor you want to use:

/etc/systemd/system/wayland-compositor.service

```
[Unit]
After=graphical.target systemd-user-sessions.service modprobe@drm.service
Conflicts=getty@tty1.service #option to disable unusable tty1

[Service]
User=username
WorkingDirectory=~

PAMName=login
TTYPath=/dev/tty1
UnsetEnvironment=TERM

StandardOutput=journal
ExecStart=/bin/labwc -s gtklock

[Install]
WantedBy=graphical.target
```

> **Note**
>
> You may need to run a screen locker such as **swaylock (https://archlinux.org/packages/?name=swaylock)** or **gtklock (https://archlinux.org/packages/?name=gtklock)** automatically through the compositor for physical security.

6.7 Use another renderer for wlroots based compositor

You can use another **wlroots renderer (https://gitlab.freedesktop.org/wlroots/wlroots/-/tree/master/render)** such as vulkan by specifying the `WLR_RENDERER` environment variable for wlroots based compositor. The list of available ones is on the **wlroots documentation (https://gitlab.freedesktop.org/wlroots/wlroots/-/blob/master/docs/env_vars.md)**.

7 Troubleshooting

7.1 Color correction

See **Backlight#Color correction**.

7.2 Slow motion, graphical glitches, and crashes

Gnome-shell users may experience display issues when they switch to Wayland from X. One of the root cause might be the `CLUTTER_PAINT=disable-clipped-redraws:disable-culling` set by yourself for Xorg-based gnome-shell. Just try to remove it from `/etc/environment` or other rc files to see if everything goes back to normal.

7.3 Remote display

- `wlroots0.18 (https://archlinux.org/packages/?name=wlroots0.18)` and `wlroots0.19 (https://archlinux.org/packages/?name=wlroots0.19)` (used by **sway**) offers a VNC backend via `wayvnc (https://archlinux.org/packages/?name=wayvnc)` since version 0.10. RDP backend has been removed **[11] (https://github.com/swaywm/wlroots/releases/tag/0.10.0)**.
- `mutter (https://archlinux.org/packages/?name=mutter)` has now remote desktop enabled at compile time, see **[12] (https://wiki.gnome.org/Projects/Mutter/RemoteDesktop)** and `gnome-remote-desktop (https://archlinux.org/packages/?name=gnome-remote-desktop)` for details.
- `krfb (https://archlinux.org/packages/?name=krfb)` offers a VNC server for `kwin (https://archlinux.org/packages/?name=kwin)`. `krfb-virtualmonitor` can be used to set up another device as an extra monitor.
- There was a merge of FreeRDP into Weston in 2013, enabled via a compile flag. The `weston (https://archlinux.org/packages/?name=weston)` package has it enabled since version 6.0.0.
- `waypipe (https://archlinux.org/packages/?name=waypipe)` (or `waypipe-git (https://aur.archlinux.org/packages/waypipe-git/)`[AUR]) is a transparent proxy for Wayland applications, with a wrapper command to run over **SSH**
  - Here is an example for launching a remote KDE kcalc under Plasma:

```
$ waypipe ssh example.local env QT_QPA_PLATFORM=wayland QT_QPA_PLATFORMTHEME=KDE dbus-launch kcalc
```

7.4 Input grabbing in games, remote desktop and VM windows

In contrast to Xorg, Wayland does not allow exclusive input device grabbing, also known as active or explicit grab (e.g. **keyboard (https://tronche.com/gui/x/xlib/input/XGrabKeyboard.html)**, **mouse (https://tronche.com/gui/x/xlib/input/XGrabPointer.html)**), instead, it depends on the Wayland compositor to pass keyboard shortcuts and confine the pointer device to the application window.

This change in input grabbing breaks current applications' behavior, meaning:

- Hotkey combinations and modifiers will be caught by the compositor and will not be sent to remote desktop and virtual machine windows.
- The mouse pointer will not be restricted to the application's window which might cause a parallax effect where the location of the mouse pointer inside the window of the virtual machine or remote desktop is displaced from the host's mouse pointer.

Wayland solves this by adding protocol extensions for Wayland and Xwayland. Support for these extensions is needed to be added to the Wayland compositors. In the case of native Wayland clients, the used widget toolkits (e.g GTK, Qt) needs to support these extensions or the applications themselves if no widget toolkit is being used. In the case of Xorg applications, no changes in the applications or widget toolkits are needed as the Xwayland support is enough.

These extensions are already included in **`wayland-protocols (https://archlinux.org/packages/?name=wayland-protocols)`**, and supported by **`xorg-xwayland (https://archlinux.org/packages/?name=xorg-xwayland)`**.

The related extensions are:

- **Xwayland keyboard grabbing protocol (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/main/unstable/xwayland-keyboard-grab/xwayland-keyboard-grab-unstable-v1.xml)**
- **Compositor shortcuts inhibit protocol (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/main/unstable/keyboard-shortcuts-inhibit/keyboard-shortcuts-inhibit-unstable-v1.xml)**
- **Relative pointer protocol (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/main/unstable/relative-pointer/relative-pointer-unstable-v1.xml)**
- **Pointer constraints protocol (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/main/unstable/pointer-constraints/pointer-constraints-unstable-v1.xml)**

Supporting Wayland compositors:

- Mutter, **GNOME**'s compositor **since release 3.28 (https://bugzilla.gnome.org/show_bug.cgi?id=783342)**
- wlroots supports relative-pointer and pointer-constraints
- Kwin
  - **KDE#X11 shortcuts conflict on Wayland**
  - **Keyboard shortcuts inhibit (https://invent.kde.org/plasma/kwin/-/blob/master/src/wayland/keyboard_shortcuts_inhibit_v1_interface.cpp)**

Supporting widget toolkits:

- GTK since release 3.22.18.

7.5 GTK themes not working

See **https://github.com/swaywm/sway/wiki/GTK-3-settings-on-Wayland**.

7.6 Avoid loading NVIDIA modules

Add `__EGL_VENDOR_LIBRARY_FILENAMES=/usr/share/glvnd/egl_vendor.d/50_mesa.json` as **environment variable** before launching a Wayland compositor like **sway**.

7.7 Magnifying/surface scaling

Screen magnifying is not solved yet, a pull request was merged mid-2022 **providing the protocol wp-surface-scale (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/145)**.

7.8 Wayland lag/stuttering since kernel 6.11.2 (AMD)

Until this issue is patched in future kernel releases, a workaround is to add `amdgpu.dcdebugmask=0x400` to the cmdline.

See: **https://community.frame.work/t/wayland-lag-stuttering-since-kernel-6-11-2/59422**

8 See also

- **Wayland documentation online (https://wayland.freedesktop.org/docs/html/)**
- **Official repository (https://gitlab.freedesktop.org/wayland)**
- **Fedora:How to debug Wayland problems**
- **We are Wayland now! (https://wearewaylandnow.com/)** - An updated version of "Are we Wayland yet?"
- **Awesome Wayland projects (https://awesomeopensource.com/projects/wayland)**
- **Cursor themes**
- **Arch Linux forum discussion (https://bbs.archlinux.org/viewtopic.php?id=107499)**
- **i3 Migration Guide - Common X11 apps used on i3 with Wayland alternatives (https://github.com/swaywm/sway/wiki/i3-Migration-Guide#common-x11-apps-used-on-i3-with-wayland-alternatives)**
- **Wayland Explorer - A better way to read Wayland documentation (https://wayland.app/protocols/)**
- **How can I tell if an application is using XWayland (https://askubuntu.com/questions/1393618/how-can-i-tell-if-an-application-is-using-xwayland)**