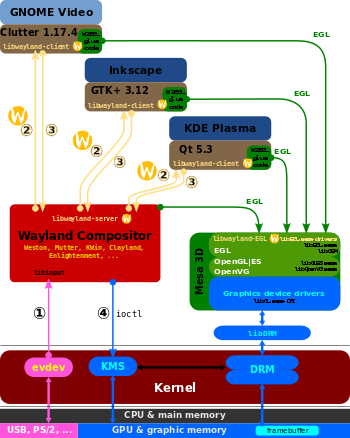
Wayland (display server protocol)

**Wayland** is a computer [protocol](https://en.wikipedia.org/wiki/Communications_protocol) that specifies the communication between a [display server](https://en.wikipedia.org/wiki/Display_server) (called a Wayland compositor]) and its clients, as well as a reference implementation of the protocol in the [C programming language](https://en.wikipedia.org/wiki/C_(programming_language))

Wayland is developed by a group of volunteers initially led by Kristian Høgsberg as a [free](https://en.wikipedia.org/wiki/Free_software) and [open](https://en.wikipedia.org/wiki/Open_source) community-driven project with the aim of replacing the [X Window System](https://en.wikipedia.org/wiki/X_Window_System) with a modern, simpler [windowing system](https://en.wikipedia.org/wiki/Windowing_system) in [Linux](https://en.wikipedia.org/wiki/Linux) and other [Unix-like](https://en.wikipedia.org/wiki/Unix-like) operating systems The project's source code is published under the terms of the [MIT License](https://en.wikipedia.org/wiki/MIT_License), a [permissive free software licence](https://en.wikipedia.org/wiki/Permissive_free_software_licence)

As part of its efforts, the Wayland project also develops a reference implementation of a Wayland compositor called [Weston](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#Weston)

**Overview**

[](https://en.wikipedia.org/wiki/File:Wayland_display_server_protocol.svg)

1. The [evdev](https://en.wikipedia.org/wiki/Evdev) module of the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel) gets an event and sends it to the [Wayland compositor](https://en.wikipedia.org/wiki/Wayland_compositor).
2. The Wayland compositor looks through its [scenegraph](https://en.wikipedia.org/wiki/Scenegraph) to determine which window should receive the event. The scenegraph corresponds to what's on screen and the Wayland compositor understands the transformations that it may have applied to the elements in the scenegraph. Thus, the Wayland compositor can pick the right window and transform the screen coordinates to window local coordinates, by applying the inverse transformations. The types of transformation that can be applied to a window is only restricted to what the compositor can do, as long as it can compute the inverse transformation for the input events.
3. As in the X case, when the client receives the event, it updates the UI in response. But in the Wayland case, the rendering happens by the client via [EGL](https://en.wikipedia.org/wiki/EGL_(API)), and the client just sends a request to the compositor to indicate the region that was updated.
4. The Wayland compositor collects damage requests from its clients and then re-composites the screen. The compositor can then directly issue an [ioctl](https://en.wikipedia.org/wiki/Ioctl) to schedule a pageflip with [KMS](https://en.wikipedia.org/wiki/KMS_(Linux_kernel)).

In recent years, Linux desktop graphics has moved from having "a pile of [rendering](https://en.wikipedia.org/wiki/Rendering_(computer_graphics)) interfaces... all talking to the [X server](https://en.wikipedia.org/wiki/Display_server), which is at the center of the universe" towards putting the Linux kernel and its components (i.e. [Direct Rendering Infrastructure (DRI)](https://en.wikipedia.org/wiki/Direct_Rendering_Infrastructure), [Direct Rendering Manager (DRM)](https://en.wikipedia.org/wiki/Direct_Rendering_Manager)) "in the middle", with "window systems like X and Wayland ... off in the corner". This will be "a much-simplified graphics system offering more flexibility and better performance".

Kristian Høgsberg could have added an [extension to X](https://en.wikipedia.org/wiki/X_Window_System_protocols_and_architecture#Extensions) as many recent projects have done, but preferred to "[push] X out of the hot path between clients and the hardware" for reasons explained in the project's FAQ:

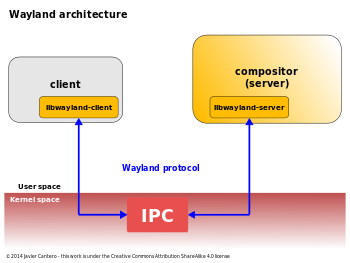
What’s different now is that a lot of infrastructure has moved from the [X server](https://en.wikipedia.org/wiki/Display_server) into the kernel (memory management, command scheduling, [mode setting](https://en.wikipedia.org/wiki/Mode-setting)) or libraries ([cairo](https://en.wikipedia.org/wiki/Cairo_(graphics)" \o "Cairo (graphics)), pixman, [freetype](https://en.wikipedia.org/wiki/Freetype), [fontconfig](https://en.wikipedia.org/wiki/Fontconfig), [pango](https://en.wikipedia.org/wiki/Pango), etc.), and there is very little left that has to happen in a central server process. ... [An X server has] a tremendous amount of functionality that you must support to claim to speak the X protocol, yet nobody will ever use this. ... This includes code tables, glyph rasterization and caching, [XLFDs](https://en.wikipedia.org/wiki/X_logical_font_description) (seriously, XLFDs!), and the entire core rendering API that lets you draw stippled lines, polygons, wide arcs and many more state-of-the-1980s style graphics primitives. For many things we've been able to keep the X.org server modern by adding extension such as [XRandR](https://en.wikipedia.org/wiki/XRandR), [XRender](https://en.wikipedia.org/wiki/XRender) and [COMPOSITE](https://en.wikipedia.org/wiki/Composite_(graphics)) ... With Wayland we can move the X server and all its legacy technology to an optional code path. Getting to a point where the X server is a compatibility option instead of the core rendering system will take a while, but we'll never get there if [we] don’t plan for it.

Wayland consists of a protocol and a reference implementation named [Weston](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#Weston). The project is also developing versions of [GTK+](https://en.wikipedia.org/wiki/GTK%2B) and [Qt](https://en.wikipedia.org/wiki/Qt_(framework)) that render to Wayland instead of to X. Most applications are expected to gain support for Wayland through one of these libraries without modification to the application.

Initial versions of Wayland have not provided [network transparency](https://en.wikipedia.org/wiki/Network_transparency), though Høgsberg noted in 2010 that network transparency is possible. It was attempted as a [Google Summer of Code](https://en.wikipedia.org/wiki/Google_Summer_of_Code) project in 2011, but was not successful. Adam Jackson has envisioned providing remote access to a Wayland application by either "pixel-scraping" (like [VNC](https://en.wikipedia.org/wiki/Virtual_Network_Computing)) or getting it to send a "rendering command stream" across the network (as in [RDP](https://en.wikipedia.org/wiki/Remote_Desktop_Protocol), [SPICE](https://en.wikipedia.org/wiki/SPICE_(protocol)) or [X11](https://en.wikipedia.org/wiki/X_Window_System)). As of early 2013, Høgsberg is experimenting with network transparency using a proxy Wayland server which sends compressed images to the real compositor. In August 2017, GNOME saw the first such pixel-scraping VNC server implementation under Wayland.

**Software architecture**

**Protocol architecture**

[](https://en.wikipedia.org/wiki/File:Wayland_protocol_architecture.svg)

In the Wayland protocol architecture, a client and a compositor communicate through the Wayland protocol using the reference implementation libraries.

Wayland protocol follows a [client–server model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model) in which clients are the graphical applications requesting the display of pixel buffers on the screen, and the server (compositor) is the service provider controlling the display of these buffers.

The Wayland reference implementation has been designed as a two-layer protocol.

* A low-level layer or *wire protocol* that handles the [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication) between the two involved [processes](https://en.wikipedia.org/wiki/Process_(computing))‍—‌client and compositor‍—‌and the [marshalling](https://en.wikipedia.org/wiki/Marshalling_(computer_science)) of the data that they interchange. This layer is message-based and usually implemented using the kernel IPC services, specifically [Unix domain sockets](https://en.wikipedia.org/wiki/Unix_domain_sockets) in the case of [Linux](https://en.wikipedia.org/wiki/Linux) and [Unix-like](https://en.wikipedia.org/wiki/Unix-like) operating systems.
* A high-level layer built upon it, that handles the information that client and compositor need to exchange to implement the basic features of a [window system](https://en.wikipedia.org/wiki/Window_system). This layer is implemented as "an asynchronous object-oriented protocol".

While the low-level layer was written manually in [C language](https://en.wikipedia.org/wiki/C_(programming_language)), the high-level layer is automatically generated from a description of the elements of the protocol stored in [XML](https://en.wikipedia.org/wiki/XML) format. Every time the protocol description of this XML file changes, the C source code that implements such protocol can be regenerated to include the new changes, allowing a very flexible, extensible and error-proof protocol.

The reference implementation of Wayland protocol is split in two [libraries](https://en.wikipedia.org/wiki/Library_(computing)): a library to be used by Wayland clients called libwayland-client and a library to be used by Wayland compositors called libwayland-server.

**Protocol overview**

The Wayland protocol is described as an "asynchronous [object-oriented](https://en.wikipedia.org/wiki/Object-oriented) protocol." *Object-oriented* means that the services offered by the compositor are presented as a series of *objects* living on the same compositor. Each object implements an *interface* which has a name, a number of methods (called *requests*) as well as several associated *events*. Every request and event has zero or more arguments, each one with a name and a [data type](https://en.wikipedia.org/wiki/Data_type). The protocol is *asynchronous* in the sense that requests do not have to wait for synchronized replies or [ACKs](https://en.wikipedia.org/wiki/Acknowledgement_(data_networks)), avoiding [round-trip delay time](https://en.wikipedia.org/wiki/Round-trip_delay_time) and achieving improved performance.

The Wayland clients can make a request (a method invocation) on some object if the object's interface supports that request. The client must also supply the required data for the arguments of such request. This is the way the clients request services from the compositor. The compositor in turn sends information back to the client by causing the object to emit events (probably with arguments too). These events can be emitted by the compositor as a response to a certain request, or asynchronously, subject to the occurrence of internal events (such as one from an input device) or state changes. The error conditions are also signaled as events by the compositor. For a client to be able to make a request to an object, it first needs to tell the server the ID number it will use to identify that object. There are two types of objects in the compositor: global objects and non-global objects. Global objects are advertised by the compositor to the clients when they are created (and also when they are destroyed), while non-global objects are usually created by other objects that already exist as part of their functionality.

The interfaces and their requests and events are the core elements that define the Wayland protocol. Each version of the protocol includes a set of interfaces, along with their requests and events, which are expected to be in any Wayland compositor. Optionally, a Wayland compositor may define and implement its own interfaces that support new requests and events, thereby extending functionality beyond the core protocol. To facilitate changes to the protocol, each interface contains a "version number" attribute in addition to its name; this attribute allows for distinguishing variants of the same interface. Each Wayland compositor exposes not only what interfaces are available, but also the supported versions of those interfaces.

**Wayland core interfaces**

The interfaces of the current version of Wayland protocol are defined in the file protocol/wayland.xml of the Wayland source code. This is an [XML](https://en.wikipedia.org/wiki/XML) file that lists the existing interfaces in the current version, along with their requests, events and other attributes. This set of interfaces is the minimum required to be implemented by any Wayland compositor.

Some of the most basic interfaces of the Wayland protocol are:

* *wl\_display* – the core global object, a special object to encapsulate the Wayland protocol itself
* *wl\_registry* – the global registry object, in which the compositor registers all the global objects that it wants to be available to all clients
* *wl\_compositor* – an object that represents the compositor, and is in charge of combining the different surfaces into one output
* *wl\_surface* – an object representing a rectangular area on the screen, defined by a location, size and pixel content
* *wl\_buffer* – an object that, when attached to a *wl\_surface* object, provides its displayable content
* *wl\_output* – an object representing the displayable area of a screen
* *wl\_pointer*, *wl\_keyboard*, *wl\_touch* – objects representing different input devices like [pointers](https://en.wikipedia.org/wiki/Pointer_(graphical_user_interfaces)) or [keyboards](https://en.wikipedia.org/wiki/Computer_keyboard)
* *wl\_seat* – an object representing a seat (a set of input/output devices) in [multiseat configurations](https://en.wikipedia.org/wiki/Multiseat_configuration)

A typical Wayland client session starts by opening a connection to the compositor using the *wl\_display* object. This is a special local object that represents the connection and does not live within the server. By using its interface the client can request the *wl\_registry* global object from the compositor, where all the global object names live, and bind those that the client is interested in. Usually the client binds at least a *wl\_compositor* object from where it will request one or more *wl\_surface* objects to show the application output on the display.

**Wayland extension interfaces**

A Wayland compositor can define and export its own additional interfaces. This feature is used to extend the protocol beyond the basic functionality provided by the core interfaces, and has become the standard way to implement Wayland protocol extensions. Certain compositors can choose to add custom interfaces to provide specialized or unique features. The Wayland reference compositor, Weston, used them to implement new experimental interfaces as a testbed for new concepts and ideas, some of which later became part of the core protocol (such as *wl\_subsurface* interface added in Wayland 1.4).

**Extension protocols to the core protocol**

**XDG-Shell protocol**

XDG-Shell protocol (see [freedesktop.org](https://en.wikipedia.org/wiki/Freedesktop.org) for XDG) is an extended way to manage surfaces under Wayland compositors (not only Weston). The traditional way to manipulate (maximize, minimize, fullscreen, etc.) surfaces is to use the wl\_shell\_\*() functions, which are part of the core Wayland protocol and live in libwayland-client. An implementation of the xdg-shell protocol, on the contrary, is supposed to be provided by the Wayland compositor. So, you will find the xdg-shell-client-protocol.h header in the Weston source tree. Each Wayland compositor is supposed to provide its own implementation.

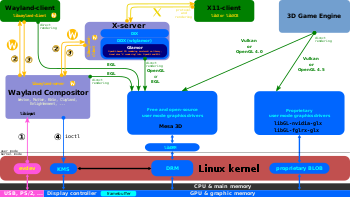
As of June 2014[[update]](https://en.wikipedia.org/w/index.php?title=Wayland_(display_server_protocol)&action=edit), XDG-Shell protocol was not versioned and still prone to changes.

xdg\_shell is a protocol aimed to substitute wl\_shell in the long term, but will not be part of the Wayland core protocol. It starts as a non-stable API, aimed to be used as a development place at first, and once features are defined as required by several desktop shells, it can be finally made stable. It provides mainly two new interfaces: xdg\_surface and xdg\_popup. The xdg\_surface interface implements a desktop-style window that can be moved, resized, maximized, etc.; it provides a request for creating child/parent relationship. The xdg\_popup interface implements a desktop-style popup/menu; an xdg\_popup is always transient for another surface, and also has implicit grab.

**IVI-Shell protocol**

IVI-Shell is an extension to the Wayland core protocol, targeting [in-vehicle infotainment](https://en.wikipedia.org/wiki/In-vehicle_infotainment) (IVI) devices.

**Rendering model**

[](https://en.wikipedia.org/wiki/File:The_Linux_Graphics_Stack_and_glamor.svg)

[Wayland compositor](https://en.wikipedia.org/wiki/Wayland_compositor) and its clients use [EGL](https://en.wikipedia.org/wiki/EGL_(API)) to draw directly into the [framebuffer](https://en.wikipedia.org/wiki/Framebuffer); [X.Org Server](https://en.wikipedia.org/wiki/X.Org_Server) with [XWayland](https://en.wikipedia.org/wiki/XWayland) and [Glamor](https://en.wikipedia.org/wiki/Glamor_(software)).

The Wayland protocol does not include a rendering API. Instead, Wayland follows a *direct rendering* model, in which the client must render the window contents to a buffer shareable with the compositor. For that purpose, the client can choose to do all the rendering by itself, use a rendering library like [Cairo](https://en.wikipedia.org/wiki/Cairo_(graphics)) or [OpenGL](https://en.wikipedia.org/wiki/OpenGL), or rely on the rendering engine of high-level widget libraries with Wayland support, such as [Qt](https://en.wikipedia.org/wiki/Qt_(software)) or [GTK+](https://en.wikipedia.org/wiki/GTK%2B). The client can also optionally use other specialized libraries to perform specific tasks, such as [Freetype](https://en.wikipedia.org/wiki/Freetype) for [font rendering](https://en.wikipedia.org/wiki/Font_rasterization).

The resulting buffer with the rendered window contents are stored in a *wl\_buffer* object. The internal type of this object is implementation dependent. The only requirement is that the content data must be shareable between the client and the compositor. If the client uses a software (CPU) renderer and the result is stored in the [system memory](https://en.wikipedia.org/wiki/Main_memory), then client and compositor can use [shared memory](https://en.wikipedia.org/wiki/Shared_memory) to implement the buffer communication without extra copies. The Wayland protocol already provides natively this kind of shared memory buffers through *wl\_shm* and *wl\_shm\_pool* interfaces. The drawback of this method is that the compositor may need to do additional work (usually to copy the shared data to the GPU) to display it, which leads to slower graphics performance.

The most typical case is for the client to render directly into a [video memory](https://en.wikipedia.org/wiki/Video_memory) buffer using a hardware (GPU) accelerated API such as [OpenGL](https://en.wikipedia.org/wiki/OpenGL), [OpenGL ES](https://en.wikipedia.org/wiki/OpenGL_ES) or [Vulkan](https://en.wikipedia.org/wiki/Vulkan_(API)). Client and compositor can share this GPU-space buffer using a special handler to reference it. This method allows the compositor to avoid additional copies of data to the GPU, resulting in faster graphics performance than using shm buffers, and therefore the preferred one. The compositor can further optimize the composition of the final scene to show on the display by using the same hardware acceleration API as the client.

When the rendering is done, and the buffer shared, the Wayland client should instruct the compositor to present the rendered contents of the buffer on the display. For this purpose, the client binds the buffer object that stores the rendered contents to the surface object, and sends a "commit" request to the surface, transferring the effective control of the buffer to the compositor. Then, the client waits for the compositor to release the buffer (signaled by an event) if it wants to reuse the buffer to render another frame, or it can use another buffer to render the new frame and, when the rendering is finished, to bind this new buffer to the surface and commit its contents. The procedure used for rendering, including the number of buffers involved and their management, is entirely under the client control.

**Comparison with other window systems**

**Differences between Wayland and X**

There are several differences between Wayland and X in regard to performance, code maintainability, and security:

Architecture

The [composition manager](https://en.wikipedia.org/wiki/Compositing_window_manager) is a separate, additional feature in X, while Wayland merges display server and compositor as a single function. Also, it incorporates some of the tasks of the [window manager](https://en.wikipedia.org/wiki/Window_manager), which in X is a separate client-side process.

Composition

Compositing is optional in X, but mandatory in Wayland. Compositing in X is "active"; that is, the compositor must fetch all pixel data, which introduces latency. In Wayland, compositing is "passive", which means the compositor receives pixel data directly from clients.

Rendering

The X server itself is able to perform rendering, although it can also be instructed to display a rendered window sent by a client. In contrast, Wayland does not expose any API for rendering, but delegates to clients such tasks (including the rendering of fonts, widgets, etc.). Window decorations can be rendered on the client side (e.g., by a graphics toolkit) or on the server side (by the compositor).

Security

Wayland isolates the input and output of every window, achieving confidentiality, integrity and availability in both cases; the original X design lacks these important security features, although some extensions have been developed trying to mitigate it. Also, with the vast majority of the code running in the client, less code needs to run with *root* privileges, improving security.

Inter-process communication

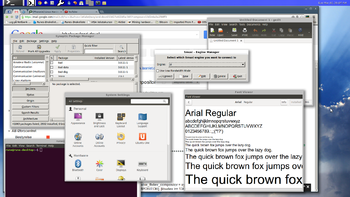
The X server provides a basic communication method between X clients, later extended by [ICCCM](https://en.wikipedia.org/wiki/ICCCM) conventions. This X client-to-client communication is used by window managers and also to implement [X sessions](https://en.wikipedia.org/wiki/X_session_manager), [selections and drag-and-drop](https://en.wikipedia.org/wiki/X_Window_selection), and other features. Wayland core protocol does not support communication between wayland clients at all, and the corresponding functionality (if needed) should be implemented by the [desktop environments](https://en.wikipedia.org/wiki/Desktop_environments) (like KDE or GNOME), or by a third party (for example, by using native [IPC](https://en.wikipedia.org/wiki/Inter-process_communication) of the underlying operating system).

Networking

The X Window System is an [architecture](https://en.wikipedia.org/wiki/X_Window_System_protocols_and_architecture) that was designed at its core to run over a network. Wayland does not offer network transparency by itself;

however, a compositor can implement any [remote desktop protocol](https://en.wikipedia.org/wiki/List_of_remote_desktop_protocols) to achieve remote displaying. In addition, there is research into Wayland image streaming and compression that would provide remote frame buffer access similar to that of [VNC](https://en.wikipedia.org/wiki/VNC).

**Compatibility with X**

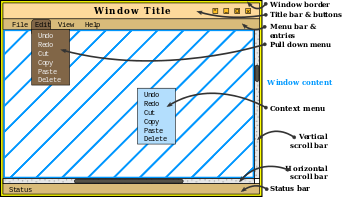
[](https://en.wikipedia.org/wiki/File:XWayland.png)

A screenshot showing x wayland

[*XWayland*](https://en.wikipedia.org/wiki/XWayland) is an [X Server](https://en.wikipedia.org/wiki/X_Window_System) running as a Wayland client, thus capable of displaying native X11 client applications in a Wayland compositor environment.This is similar to the way [XQuartz](https://en.wikipedia.org/wiki/XQuartz) runs X applications in [macOS](https://en.wikipedia.org/wiki/MacOS)’s native windowing system. The goal of XWayland is to facilitate the transition from X Window System to Wayland environments, providing a way to run unported applications in the meantime. XWayland was mainlined into [X.Org Server](https://en.wikipedia.org/wiki/X.Org_Server) version 1.16.

Widget toolkits such as [Qt 5](https://en.wikipedia.org/wiki/Qt_(software)) and [GTK+ 3](https://en.wikipedia.org/wiki/GTK%2B) can switch their graphical back-end at run time, allowing users to choose at [load time](https://en.wikipedia.org/wiki/Load_time) whether they want to run the application over X or over Wayland. Qt 5 provides the -platform command-line option to that effect, whereas GTK+ 3 lets users select the desired [GDK](https://en.wikipedia.org/wiki/GDK) back-end by setting the GDK\_BACKEND [Unix environment variable](https://en.wikipedia.org/wiki/Environment_variable).

**Wayland compositors**

[](https://en.wikipedia.org/wiki/File:Window_(windowing_system).svg)

Typical elements of a [window](https://en.wikipedia.org/wiki/Window_(computing)). Neither Wayland nor X11 specifies what software is responsible for rendering the [window decoration](https://en.wikipedia.org/wiki/Window_decoration). Weston requires that they are drawn by the client, but [KWin](https://en.wikipedia.org/wiki/KWin) will implement server-side decoration.

*Main article:* [*List of Wayland compositors*](https://en.wikipedia.org/wiki/List_of_display_servers#Wayland)

[Display servers](https://en.wikipedia.org/wiki/Display_server) that implement the Wayland display server protocol are also called *Wayland compositors* because they additionally perform the task of a [compositing window manager](https://en.wikipedia.org/wiki/Compositing_window_manager).

* [Weston](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#Weston) – the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of a Wayland compositor; Weston implements client-side decoration
* Lipstick – mobile [graphical shell](https://en.wikipedia.org/wiki/Graphical_shell) framework which implements Wayland compositor. It is used in [Sailfish OS](https://en.wikipedia.org/wiki/Sailfish_OS), [Nemo Mobile](https://en.wikipedia.org/wiki/Mer_(software_distribution)#Nemo_Mobile) and [AsteroidOS](https://en.wikipedia.org/wiki/AsteroidOS).
* [Enlightenment](https://en.wikipedia.org/wiki/Enlightenment_(window_manager)) has full Wayland support since version 0.20.
* [KWin](https://en.wikipedia.org/wiki/KWin) had incomplete Wayland support in April 2013.
* [Mutter](https://en.wikipedia.org/wiki/Mutter_(software)) maintains a separate branch for the integration of Wayland for GNOME 3.9 (in September 2013).
* [Clayland](https://en.wikipedia.org/wiki/Clutter_(software)) is a simple example Wayland compositor using Clutter.
* [Westeros](https://en.wikipedia.org/w/index.php?title=Westeros_(software)&action=edit&redlink=1) is a Wayland compositor library that allows applications to create their own Wayland displays, which allows nesting and embedding of third party applications.
* **Weston**

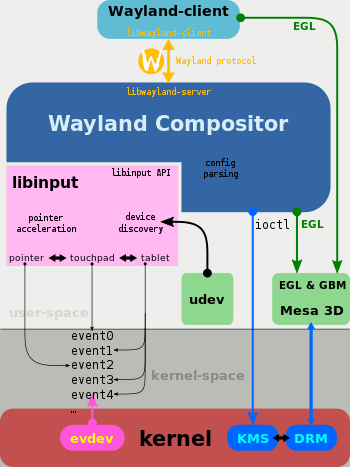
Weston is the reference implementation of a Wayland compositor also developed by the Wayland project. It is written in [C](https://en.wikipedia.org/wiki/C_(programming_language)) and published under the [MIT License](https://en.wikipedia.org/wiki/MIT_License). Weston only has official support for the [Linux](https://en.wikipedia.org/wiki/Linux) operating system due to its dependence on certain features of the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel), such as [kernel mode-setting](https://en.wikipedia.org/wiki/KMS_driver), [Graphics Execution Manager](https://en.wikipedia.org/wiki/Graphics_Execution_Manager) (GEM), and [udev](https://en.wikipedia.org/wiki/Udev), which have not yet been implemented in other Unix-like operating systems. When running on Linux, handling of the input hardware relies on [evdev](https://en.wikipedia.org/wiki/Evdev), while the handling of buffers relies on [Generic Buffer Management](https://en.wikipedia.org/wiki/Generic_Buffer_Management) (GBM). However, in 2013 a prototype port of Weston to [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD) was announced.

Weston relies on GEM to share application buffers between the compositor and applications. It contains a plugin system of "shells" for common desktop features like docks and panels. Clients are responsible for the drawing of their window borders and their decorations. For rendering, Weston can use [OpenGL ES](https://en.wikipedia.org/wiki/OpenGL_ES) or the pixman library to do [software rendering](https://en.wikipedia.org/wiki/Software_rendering). The full OpenGL implementation is not used, because on most current systems, installing the full OpenGL libraries would also install [GLX](https://en.wikipedia.org/wiki/GLX) and other [X Window System](https://en.wikipedia.org/wiki/X_Window_System) support libraries as dependencies.

*Maynard* is a [graphical shell](https://en.wikipedia.org/wiki/Graphical_shell) and has been written as a plugin for Weston, just as the [GNOME Shell](https://en.wikipedia.org/wiki/GNOME_Shell) has been written as a plugin to [Mutter](https://en.wikipedia.org/wiki/Mutter_(software)).

A remote access interface for Weston was proposed in October 2013 by a [RealVNC](https://en.wikipedia.org/wiki/RealVNC) employee.

**libinput**

[](https://en.wikipedia.org/wiki/File:Libinput_for_Wayland_compositors.svg)

libinput was created to consolidate the input stack across multiple Wayland compositors.

Libinput handles input devices for multiple Wayland compositors and also provides a generic [X.Org Server](https://en.wikipedia.org/wiki/X.Org_Server) input driver. It aims to provide one implementation for multiple Wayland compositors with a common way to handle input events while minimizing the amount of custom input code compositors need to include. libinput provides device detection (via [udev](https://en.wikipedia.org/wiki/Udev)), device handling, input device event processing and abstraction.

Version 1.0 of libinput followed version 0.21, and included support for tablets, button sets and touchpad gestures. This version will maintain stable API/ABI.

As GNOME/GTK+ and [KDE Frameworks 5](https://en.wikipedia.org/wiki/KDE_Frameworks_5) have mainlined the required changes, Fedora 22 will replace X.Org's evdev and Synaptics drivers with libinput.

The Weston code for handling input devices (keyboards, pointers, touch screens, etc.) was split into its own separate library, called *libinput*, for which support was first merged in Weston 1.5.

With version 1.16, the [X.Org Server](https://en.wikipedia.org/wiki/X.Org_Server) obtained support for the [libinput](https://en.wikipedia.org/wiki/Libinput) library in form of a wrapper called xf86-input-libinput.

**Wayland Security Module**

Wayland Security Module is a proposition that resembles the [Linux Security Module](https://en.wikipedia.org/wiki/Linux_Security_Module) interface found in the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel).

Some applications (especially the ones related to [accessibility](https://en.wikipedia.org/wiki/Accessibility)) require privileged capabilities that should work across different Wayland compositors. Currently, applications under Wayland are generally unable to perform any sensitive tasks such as taking screenshots or injecting input events. Wayland developers are actively looking for feasible ways to handle privileged clients securely and then designing privileged interfaces for them.

Wayland Security Module is a way to delegate security decisions within the compositor to a centralized security decision engine.

**Adoption**

As explained in the "Software architecture" section above, the Wayland protocol is designed to be simple so that additional protocols and interfaces need to be defined and implemented to achieve a holistic windowing system. As of July 2014, these additional interfaces are actively being worked on. So, while the toolkits already fully support Wayland, the developers of the [raphical gshells](https://en.wikipedia.org/wiki/Graphical_shell) are cooperating with the Wayland developers creating the necessary additional interfaces.

**Desktop Linux distributions**

In general, out of the box support for a full desktop running Wayland in major Linux distributions is still in early stages. Most Linux distributions that ship version 3.20 or newer of the Gnome desktop environment do support manually installing a Wayland session, and in case of Gnome 3.22 or newer might default to Wayland. Notable early adopters that provide Wayland out of the box are:

* [Fedora](https://en.wikipedia.org/wiki/Fedora_Linux) starting with version 25 uses Wayland for the default GNOME 3.22 desktop session, with [X.Org](https://en.wikipedia.org/wiki/X.Org_Server) as a fallback if the graphics driver cannot support Wayland. Fedora 25 carries patches that make the Wayland session work with NVidia drivers as well.
* RebeccaBlackOS is a [live USB](https://en.wikipedia.org/wiki/Live_USB) Debian-based Linux distribution that allows a convenient way to try out a real Wayland desktop without having to make any modifications to the main operating system of the computer. It is being used since as early as 2012 to showcase Wayland.
* **Toolkit support**

Toolkits supporting Wayland include the following:

* [Clutter](https://en.wikipedia.org/wiki/Clutter_(software)) has complete Wayland support.
* [EFL](https://en.wikipedia.org/wiki/Enlightenment_Foundation_Libraries) has complete Wayland support, except for selection.
* [GTK+](https://en.wikipedia.org/wiki/GTK%2B) 3.20 has complete Wayland support.
* [Qt 5](https://en.wikipedia.org/wiki/Qt_5) has complete Wayland support.
* [SDL](https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer) support for Wayland debuted with the 2.0.2 release and was enabled by default since version 2.0.4.
* [GLFW](https://en.wikipedia.org/wiki/GLFW) 3.2 has Wayland support.
* [FreeGLUT](https://en.wikipedia.org/wiki/FreeGLUT) has initial Wayland support.

**Desktop environments**

Desktop environments in process of being ported from X to Wayland include [GNOME](https://en.wikipedia.org/wiki/GNOME), [KDE Plasma 5](https://en.wikipedia.org/wiki/KDE_Plasma_5) and [Enlightenment](https://en.wikipedia.org/wiki/Enlightenment_(software)). The [Hawaii desktop environment](https://en.wikipedia.org/w/index.php?title=Hawaii_(desktop_environment)&action=edit&redlink=1) is a desktop environment that exclusively supports Wayland.

In November 2015, Enlightenment e20 was announced with "full Wayland support". GNOME 3.20 was the first version "to have a full Wayland session". GNOME 3.22 now defaults to Wayland. GNOME 3.24 will ship the NVidia patches developed for Fedora 25.

Wayland support for KDE Plasma was delayed until the release of Plasma 5, though previously [KWin](https://en.wikipedia.org/wiki/KWin) 4.11 got an experimental Wayland support. The version 5.4 of Plasma was the first with a Wayland session.

**Other software**

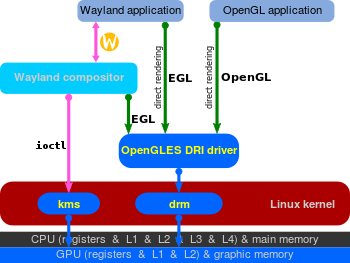
Other software supporting Wayland includes the following:

* [Intelligent Input Bus](https://en.wikipedia.org/wiki/Intelligent_Input_Bus) is working on Wayland support, it could be ready for Fedora 22.
* [RealVNC](https://en.wikipedia.org/wiki/RealVNC) published a Wayland developer preview in July 2014.
* [Maliit](https://en.wikipedia.org/wiki/Maliit) is an [input method](https://en.wikipedia.org/wiki/Input_method) framework that runs under Wayland.
* [kmscon](https://en.wikipedia.org/wiki/Kmscon) supports Wayland with wlterm.
* [Mesa](https://en.wikipedia.org/wiki/Mesa_(computer_graphics)) has Wayland support integrated.
* [Eclipse](https://en.wikipedia.org/wiki/Eclipse_(software)) was made to run on Wayland during a [GSoC](https://en.wikipedia.org/wiki/Google_Summer_of_Code)-Project in 2014.
* The [Vulkan WSI](https://en.wikipedia.org/wiki/Vulkan_WSI) (Window System Interface) is a set of API calls serve a similar purpose as EGL does for OpenGL ES or GLX for OpenGL. Vulkan WSI includes support for Wayland from day one: VK\_USE\_PLATFORM\_WAYLAND\_KHR. Vulkan clients can run on unmodified Wayland servers, including Weston, GENIVI LayerManager, Mutter / GNOME Shell, Enlightenment, and more. The WSI allows applications to discover the different GPUs on the system, and display the results of GPU rendering to a window system.

**Mobile and embedded hardware**

Mobile and embedded hardware supporting Wayland includes the following:

* [GENIVI Alliance](https://en.wikipedia.org/wiki/GENIVI_Alliance): The GENIVI automotive industry consortium for [in-vehicle infotainment](https://en.wikipedia.org/wiki/In-vehicle_infotainment) (IVI) supports Wayland.
* [Raspberry Pi](https://en.wikipedia.org/wiki/Raspberry_Pi): The [Raspberry Pi Foundation](https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation) in collaboration with [Collabora](https://en.wikipedia.org/wiki/Collabora) released [Maynard](https://en.wikipedia.org/wiki/Maynard_(software)) and work on improving performance and memory consumption, but do not expect to be able to replace X11 as the default display server until later in 2013.
* [Jolla](https://en.wikipedia.org/wiki/Jolla): Smartphones from Jolla use Wayland. It is also used as standard when Linux [Sailfish OS](https://en.wikipedia.org/wiki/Sailfish_OS) is used with hardware from other vendors or when it is installed into Android devices by users.
* [Tizen](https://en.wikipedia.org/wiki/Tizen): Tizen up to 2.x supports Wayland in [in-vehicle infotainment](https://en.wikipedia.org/wiki/In-vehicle_infotainment) (IVI) setups and from 3.0 onward defaults to Wayland.
* **History**

[](https://en.wikipedia.org/wiki/File:Linux_graphics_drivers_DRI_Wayland.svg)

**Wayland** uses [direct rendering](https://en.wikipedia.org/wiki/Direct_Rendering_Manager) over [EGL](https://en.wikipedia.org/wiki/EGL_(API)).

Kristian Høgsberg, a [Linux](https://en.wikipedia.org/wiki/Linux) graphics and [X.Org](https://en.wikipedia.org/wiki/X.Org_Server) [developer](https://en.wikipedia.org/wiki/Software_developer) who previously worked on [AIGLX](https://en.wikipedia.org/wiki/AIGLX) and [DRI2](https://en.wikipedia.org/wiki/Direct_Rendering_Infrastructure), started Wayland as a spare-time project in 2008 while working for [Red Hat](https://en.wikipedia.org/wiki/Red_Hat). His stated goal was a system in which "every frame is perfect, by which I mean that applications will be able to control the rendering enough that we'll never see tearing, lag, redrawing or flicker." Høgsberg was driving through the town of [Wayland, Massachusetts](https://en.wikipedia.org/wiki/Wayland,_Massachusetts) when the underlying concepts "crystallized", hence the name.

In October 2010, Wayland became a [freedesktop.org](https://en.wikipedia.org/wiki/Freedesktop.org) project.As part of the migration the prior [Google Group](https://en.wikipedia.org/wiki/Google_Groups) was replaced by the *wayland-devel* mailing list as the project's central point of discussion and development.

The Wayland client and server libraries were initially released under the [MIT License](https://en.wikipedia.org/wiki/MIT_License), while the reference compositor Weston and some example clients used the [GNU General Public License version 2](https://en.wikipedia.org/wiki/GNU_General_Public_License#Version_2). Later all the GPL code [was relicensed](https://en.wikipedia.org/wiki/Software_relicensing) under the MIT license "to make it easier to move code between the reference implementation and the actual libraries". In 2015 it was discovered that the license text used by Wayland was a slightly different and older version of the MIT license, and the license text was updated to the current version used by the [X.Org project](https://en.wikipedia.org/wiki/X.Org_Server) (known as [MIT Expat License](https://en.wikipedia.org/wiki/Expat_License)).

Wayland works with all [Mesa-compatible](https://en.wikipedia.org/wiki/Mesa_(computer_graphics)) drivers with [DRI2](https://en.wikipedia.org/wiki/DRI2) support as well as [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) drivers via the [Hybris project](https://en.wikipedia.org/wiki/Hybris_(software)).

The developers of Wayland are largely current [X.Org Server](https://en.wikipedia.org/wiki/X.Org_Server) developers

I wrote the document out the search of what I got From Wikipedia, the free encyclopedia

  (Redirected from [Wayland compositor](https://en.wikipedia.org/w/index.php?title=Wayland_compositor&redirect=no))

## References

* 1. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-1) initial%20commit
  2. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-2) [lists.freedesktop.org/archives/wayland-devel/2017-August/034748.html](https://lists.freedesktop.org/archives/wayland-devel/2017-August/034748.html)
  3. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-3) [lists.freedesktop.org/archives/wayland-devel/2017-August/034749.html](https://lists.freedesktop.org/archives/wayland-devel/2017-August/034749.html)
  4. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-4) [lists.freedesktop.org/archives/wayland-devel/2017-August/subject.html](https://lists.freedesktop.org/archives/wayland-devel/2017-August/subject.html)
  5. ^ [Jump up to: ***a***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-License_5-0) [***b***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-License_5-1) [***c***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-License_5-2) Harrington, Bryce (15 September 2015). [*"[ANNOUNCE] wayland 1.8.93"*](http://lists.freedesktop.org/archives/wayland-devel/2015-September/024226.html). [*freedesktop.org*](https://en.wikipedia.org/wiki/Freedesktop.org) (Mailing list).
  6. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-wllicense_6-0) [*"wayland/wayland: root/COPYING"*](http://cgit.freedesktop.org/wayland/wayland/tree/COPYING). freedesktop.org*. Retrieved 14 June 2015*.
  7. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Larabel_2015_license_7-0) Larabel, Michael (10 June 2015). [*"Wayland's MIT License To Be Updated/Corrected"*](https://www.phoronix.com/scan.php?page=news_item&px=Wayland-License-MIT-Update). Phoronix.com*. Retrieved 17 April 2016*.
  8. ^ [Jump up to: ***a***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_homepage_8-0) [***b***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_homepage_8-1) [***c***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_homepage_8-2) [*"Wayland"*](https://wayland.freedesktop.org/). Wayland project*. Retrieved 9 May 2016*.
  9. ^ [Jump up to: ***a***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_FAQ_9-0) [***b***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_FAQ_9-1) [***c***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_FAQ_9-2) [***d***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Wayland_FAQ_9-3) [*"Wayland FAQ"*](http://wayland.freedesktop.org/faq.html). Wayland project*. Retrieved 9 May 2016*.
  10. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Corbet_2010_10-0) Corbet, Jonathan (5 November 2010). [*"LPC: Life after X"*](https://lwn.net/Articles/413335/). LWN.net*. Retrieved 17 April 2016*.
  11. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-H.C3.B8gsberg_remotewl_11-0) Høgsberg, Kristian (9 November 2010). [*"Network transparency argument"*](http://lists.freedesktop.org/archives/wayland-devel/2010-November/000097.html). Wayland isn't a remote rendering API like X, but that doesn't exclude network transparency. Clients render into a shared buffer and then have to tell the compositor (...) what they changed. The compositor can then send the new pixels in that region out over the network. The Wayland protocol is already violently asynchronous, so it should be able to handle a bit of network lag gracefully. Remote fullscreen video viewing or gaming isn't going to work well, [but] I don't know any other display system that handles that well and transparently.
  12. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Larabel_2011_remote_12-0) Larabel, Michael (18 August 2011). [*"Remote Wayland Server Project: Does It Work Yet?"*](https://www.phoronix.com/scan.php?page=news_item&px=OTgxMQ). Phoronix.com*. Retrieved 17 April 2016*.
  13. [**Jump up ^**](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Jackson_remotewl_13-0) Jackson, Adam (9 November 2010). [*"[Re:] Ubuntu moving towards Wayland"*](http://lists.fedoraproject.org/pipermail/devel/2010-November/145306.html).
  14. ^ [Jump up to: ***a***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Stone_2013_LCA_14-0) [***b***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Stone_2013_LCA_14-1) Stone, Daniel (28 January 2013). [*The real story behind Wayland and X*](http://mirror.linux.org.au/pub/linux.conf.au/2013/mp4/The_real_story_behind_Wayland_and_X.mp4) (Speech). linux.conf.au 2013. Canberra*. Retrieved 13 April 2016*.
  15. ^ [Jump up to: ***a***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Willis_2013_15-0) [***b***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Willis_2013_15-1) [***c***](https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)#cite_ref-Willis_2013_15-2) Willis, Nathan (13 February 2013). [*"LCA: The ways of Wayland"*](https://lwn.net/Articles/536862/). LWN.net*.*