ELIVE

Elive Desktop environment is based on Enlightenment. The enlightenment desktop environment has been impressing users. Bringing this mature, visually appealing environment to new Linux users is what the Elive distribution is all about.

One looks at Elive’s graceful and charming environment, one will notice it’s unlike today’s 3D virtualization. Elive can run efficiently on older system as well with a gamut of desktop applications.

Elive has its own repository with more than 400 elive specific packages some are just rebuilt with minor changes, like tweaking their default configuration files or including skin, etc. Elive is based on Debian, its package management is pristine, it’s the most stable, secure and trust worthy system available. Elive includes MP3, MP3 are heavily used by everyone whether in your MP3 player or for internet streaming.

Elive desktop environment is based on enlightenment. Enlightenment X11 is currently transitioning to Wayland. Elive is the future of graphical display layers on Linux. Primary environment is Linux. Enlightenment Library not just to make Enlightenment but other application for day to day use. Many common features have been included, such as a terminal emulator, video player, an image viewer, and even the making of an IDE.

Native Tizen application are developed using EFL is based around a screen graph from the ground up. This makes it possible to seamlessly switch from software rendering to OpenGL or any other mechanism that can be put in a render engine for evas the canvas screen graph engine. Also permits layer widgets and objects with alphorn channels from the ground up without any special coding tricks.

Tizen also has a library of its own. Main stream desktop environment for Unix-like operating system use the X windows system, and include KDE, GNOME, Xfce, and LXDE, any of which may be selected by users and are not tied exclusively to the operating system in use.

On systems running the X window system (typically Unix-family systems such as Linux, the BSDs and formal Unix distributions). But most desktop environments provide a default configuration that works with minimal user setup.

As GNOME and KDE focus on high-performance computers users of less performance or older computers often prefer alternative desktop environments, specially created for low-performance systems. Most commonly used light weight desktop environments include LxDE and XFce; they both use GTK+, which is the same underlying toolkit GNOME uses. The MATE desktop environment, a fork of GNOMES2, is comparable to xfce in its use of RAM and processor cycles, but is often considered more as an alternative to other lightweight desktop environments.

Linux desktop graphic has moved from having “a pile of rendering interface all talking to the X server, which is at the center of universe “towards putting the Linux kernel and its components’ in the middle” with “windows system Like X and Wayland off in the Conner”. This will be “a much-simplified graphic system offering, more flexibility and better performance.

Now a lot of infrastructure has moved from the X server into the kernel (memory management, command scheduling, mode setting or library and there is very little left that has to happen in a central server process. With Wayland X can be moved and all its legacy technology to an optional code path.

Wayland consists of a protocol and a reference implementation named Waston. The project is also developing versions GYK+ and Qt that render to Wayland instead of to X, most application.

Initial versions of Wayland have not provided network transparency. Adam Jackson has envisioned providing remote access to a Wayland application by either “pixel- scraping (like VNC) or getting it to send a rendering command stream “across the network as in RDP, spice or X11)

In August 2017, GNOME saw the first such pixel-scraping VNC server implementation under Wayland.

Protocol architecture

Wayland protocol follows a client- server 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 between the two involved processes-client and compositor and the marshalling of the data that they interchange. This layer is massage-based and usually implemented using the kernel IPc services, specifically Unix domain sockets in the case of Linux and Unix-like operating system.

The 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. This layer is implemented as “a synchronous object oriented protocol”.

While the low-level layer was written manually in C language, the high-level layer is automatically generated from a description of the elements of the protocol stored in 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 change, allowing a very flexible, extensible and error- proof protocol.

Protocol review

The Wayland protocol is described as an asynchronous 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, number, number of methods (called request) as well as several associated events. Every request or event has zero or more arguments, each one with a name and a data type. The protocol is asynchronous in the sense that request do not have to wait for synchronized replies or ACKs, avoiding 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 interface 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 (not only Weston). The traditional way to manipulate (maximize, minimize, full screen 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.

Rendering model

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 or OpenGL, or rely on the rendering engine of high-level widget libraries with Wayland support, such as Qt or GTk+. The client can also optionally use other specialized libraries to perform specific task, such as free type for font rendering.

The resulting buffer with the rendered window contents are store 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.

The most typical case is for the client to render directly into a video memory buffer using a hardware (GPU) accelerated API such as OpenGL Es or Vulkan. Client and compositor can share this GPU-space buffer using a special handler to reference it.

When the rendering is done, and the buffer shared, the Wayland client should instruct the compositor to present the rendered contents to the buffer on the display.

Enlighten your desktop with Elive

The enlightenment desktop environment has been impressing users. Bringing this mature, visually appealing environment to new Linux users is what the Elive distribution is all about.