University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

wlstem

A new library to aid in creating Wayland compositors

Luana Martins de Freitas Barbosa[a]

[a] Formerly known as Mateus Carmo Martins de Freitas Barbosa

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Alfredo Goldman
Co-supervisor: Nelson Posse Lago

São Paulo
December 23, 2021

Resumo

Luana Martins de Freitas Barbosa. **wlstem: *Uma nova biblioteca para auxiliar a criação de compositores Wayland***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Em sistemas operacionais Unix, o X11 tradicionalmente fornecia a base para a implementação de sistemas gráficos de janelas, permitindo a criação de gerenciadores de janelas X. Porém, com o passar dos anos, o X11 se tornou difícil de ser mantido, de modo que um novo projeto, chamado Wayland, foi criado para substituí-lo. Na arquitetura Wayland, os compositores Wayland são os responsáveis pelo gerenciamento de janelas; porém, eles também têm várias outras atribuições, que eram antes gerenciadas pelo X11. Como consequência, escrever um compositor Wayland requer uma quantidade de código consideravelmente maior do que escrever um gerenciador de janelas X. Algumas bibliotecas, entre as quais se destaca o wlroots, foram criadas para mitigar esse problema; porém, o wlroots provê apenas uma parte do código adicional que compositores Wayland precisam implementar, deixando o restante para ser implementado pelos usuários da biblioteca. Neste trabalho, nós refatoramos o código do usuário mais proeminente do wlroots, o compositor Wayland chamado sway, de modo a extrair para uma nova biblioteca, **wlstem**, partes do código do sway que não fossem relacionadas ao gerenciamento de janelas, e que poderiam, portanto, ser reutilizadas em outros compositores Wayland. No processo de criação da nova biblioteca, nós primeiramente simplificamos o código do sway, a fim de remover parte de seu código que era claramente não-reutilizável, e também para que pudéssemos trabalhar com uma base de código menor; então, nós criamos uma biblioteca que consistia em uma parte do código do sway, e progressivamente migramos código relevante do sway para a wlstem, sempre o refatorando para que a wlstem não tivesse código relacionado ao gerenciamento de janelas. Ao final deste trabalho, a wlstem lida apropriadamente com *hotplugging* de dispositivos de saída, e lida parcialmente com *hotplugging* de dispositivos de entrada. O projeto resultante deste trabalho pode ser visto em https://github.com/luamfb/wlstem.

Palavras-chave: Wayland. X11. sway. compositor. gerenciador de janelas.

Abstract

Luana Martins de Freitas Barbosa. **wlstem:** ***A new library to aid in creating Wayland compositors***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

In Unix-like operating systems, X11 traditionally provided the basis for implementing graphical windowing systems, by allowing the creation of X window managers. However, X11 has become difficult to maintain over the years. Therefore, a new project named Wayland has been devised as the modern replacement of X11. Under the Wayland architecture, Wayland compositors are responsible for window management; however, they are also responsible for several other tasks previously handled by X11. As a consequence, writing a Wayland compositor requires far more code than writing an X window manager. Some libraries, notably wlroots, have been created to address this issue, but only provide a part of the extra code Wayland compositors must implement, leaving the rest for library users to write. In this work, we refactor wlroots' most prominent user, the Wayland compositor named sway, to extract parts of its code which are unrelated to window management, and which could therefore be reused in other Wayland compositors, to a new library, **wlstem**. Wlstem provides some of the extra code Wayland compositors need and wlroots does not already provide, thus facilitating the creation of new Wayland compositors. To create this new library, we have first simplified sway's code, in order to remove some of its clearly non-reusable code, and so we could work with a smaller code base; then, we have created a library from a small part of sway's code, progressively migrated relevant code from sway to wlstem, refactoring it so wlstem would not have any window management specific code. By the end of our work, wlstem appropriately handled output device hotplugging, and partially handled input device hotplugging. The resulting project can be accessed at https://github.com/luamfb/wlstem.

Keywords: Wayland. X11. sway. compositor. window manager.

Contents

Introduction

In Unix-like operating systems, X11 has provided, for several decades, the basis for creating graphic windowing systems. However, a full windowing system using X11 also needs a separate program named X window manager. There is a wide variety of ways to arrange windows in a screen, decide which one should have focus, among several other decisions delegated to the X window manager; thus, a vast number of X window managers have been created, catering for different user interface styles and preferences.

More recently, a new project named Wayland has been created to replace X11, and the role fulfilled by X window managers belong, under Wayland, to a so-called Wayland compositor. However, Wayland compositors also accumulate several other tasks which X11 used to handle; therefore, creating a Wayland compositor from scratch requires writing far more code than creating an X window manager from scratch, hindering the creation of a richer set of user interface styles with Wayland.

Many libraries have been created to alleviate this issue, by providing code which all Wayland compositors would need to write: notably, a library named wlroots. However, wlroots also leaves out a fair amount of code Wayland compositors need to implement, and which is unrelated to window management. As such, creating a Wayland compositor with wlroots still demands writing far more code than creating an X window manager. Thus, there is need for a new library, which will complement wlroots by providing Wayland compositor code it does not provide, and therefore facilitate the creation of new Wayland compositors even further.

We have organized this text as follows: in Chapter 1, we explain in detail the concepts and terminology we consider necessary for understanding this work. In Chapter 2, we elaborate further on the goal of this work, and highlight some interesting challenges we have faced in achieving this goal. In Chapter 3, we delineate the development of wlstem, citing the changes to our codebase in a roughly chronological order. Finally, in Chapter 4, we describe the current situation of the library we have developed, and how we intend to continue improving it in the future.

Chapter 1

Background

In this chapter, we present the X Window System and its intended replacement, named Wayland, highlighting how the creation of a full windowing system requires more code with the latter than the former. We also show a specific Wayland windowing system named sway, along with wlroots, its support library, which provides most – but not all – of the extra code required of Wayland windowing systems. Finally, we take a closer look at sway's implementation, indicating to which extent the aforementioned extra code is present in it.

1.1 X Window System (X11)

The **X Window System**, more commonly known as **X11** or simply **X** [33], is a group of standards and specifications for a windowing system. It includes the main protocol, named *X Window System Protocol*, along with several extensions [34]. *Xorg* is free software that provides an implementation of X11 [36] and is, in practice, the most widely used implementation among Linux users [1].

X11 has a client-server architecture, in which GUI[1] programs are clients of a server named *X Server* [17]. The X server controls the access to the graphics device, and clients send commands to the server such as "draw a line here" or "render this text with this font there" [17]. The server is also responsible for reading input from devices such as mouse and keyboard, and sending it to clients [17], among other tasks.

There are some libraries to aid X clients in communicating with the X Server, notably Xlib [17] and the newer X C Binding (XCB) [35]. Additionally, toolkits for creating GUI programs, such as GTK+ [6] and Qt [12], provide higher-level abstractions on top of these libraries.

X11 deliberately omits some of the functionality needed for a full-fledged graphical interface: it was created to "provide mechanism, but not policy" [16]. As a few examples of what falls into "policy", consider these questions:

[1] Graphical User Interface

- Where should a newly created window be positioned in the screen? What size should it be given?
  - If there is more than one screen, in which screen(s) should the new window be displayed?
- If the mouse hovers over an unfocused window, should it receive focus?
  - If the answer is no, and the unfocused window is clicked on, should it receive focus then?
- Should a newly created window have borders or other decorations? (These may include buttons to close the window, minimize it, among other actions.)
- Should windows be allowed to overlap?
  - If so, what should be displayed in the area where windows overlap? Should the content of only one of them be shown, so that one window is "in front" of the other, or should a linear combination of the windows' contents on the area be shown instead, to emulate transparency?

A functioning windowing system must provide code to answer these questions as the situation arises. Since X11 does not provide policy, these decisions are left to a special X client, named **window manager** [16]. Over the course of X11's existence, a multitude of window managers have been created [2] to accommodate several out of all possible different policies for managing windows.

1.2 Wayland

Wayland describes itself as "a simpler replacement for X, easier to develop and maintain" [26]. Since X11's creation in the 1980s [10], much of its previously useful functionality has become obsolete: according to Wayland's FAQ[2], this includes X core fonts, and the entire X rendering API, since it allows rendering stippled lines, polygons and many other graphics primitives, none of which are used anymore [25]. According to same source, "When you're an X server there's a tremendous amount of functionality that you must support to claim to speak the X protocol, yet nobody will ever use this. [...] With Wayland we can move the X server and all its legacy technology to an optional code path." [25]

In a similar way to X11, Wayland also defines a client-server architecture, along with a protocol (whose implementation is also called Wayland) for communication between the server – named **Wayland compositor** – and its clients [26]. However, unlike X11, under Wayland there is no client or any sort of separate process that works as a window manager; instead, managing windows is one of the attributions of a Wayland compositor [24].

Among the toolkits for creating GUI programs, many already allow creating Wayland clients, including the aforementioned GTK+ [27] and Qt [13]. Also, among the libraries

[2] Frequently Asked Questions

that provide a cross-platform manner to create windows, many of those which have Linux support also allow creating Wayland clients, including SDL2 [11], GLFW [5] and winit [30]. This indicates a reasonable Wayland adoption on the client side.

On the server side, however, Wayland adoption has not progressed as much: the number of Wayland compositors listed by the Arch Linux wiki [3] is considerably behind the number of X window managers listed by the same source [2]. One of the factors that may explain this discrepancy is the extra complexity Wayland compositors have, in comparison to X window managers: the former accumulates all responsibilities of the latter (the "policy"), and also needs to supply some of the functionality previously supplied by the X Server (the "mechanism").

1.3 libweston, wlroots and sway

Weston is the reference implementation for a Wayland compositor [29]. A library named *libweston* has been created from it, as "an effort to separate the re-usable parts of Weston into a library", according to its documentation. The same source also states that "Libweston provides most of the boring and tedious bits [...] so that people who just want to write a new 'Wayland window manager' (WM) [...] can focus on the WM part" [9].

The previous description corroborates the fact that a Wayland compositor is considerably more complex than an X window manager – enough to justify creating a new library to deal with "the boring and tedious bits" of implementing a Wayland compositor. However, libweston's documentation also warns that "In current form, libweston is an amalgam of various APIs mashed together and currently it needs a large clean-up and re-organization and possibly, a split into class-specific files" [9], which indicates this library may not be a suitable solution for the problem at hand.

However, libweston is only one of a few attempts to solve this issue. Another attempt was a project named *wlc* [31]. Among its users were **sway**, a Wayland compositor compatible with the X window manager i3 [15]. In order to overcome perceived flaws in wlc, sway's main developer created a new library named **wlroots** [4]. Eventually, wlc would become officially deprecated in favor of wlroots [31].

Wlroots's README file presents it as "Pluggable, composable, unopinionated modules for building a Wayland compositor; or about 50,000 lines of code you were going to write anyway", and claims that "wlroots implements a huge variety of Wayland compositor features and implements them *right*, so you can focus on the features that make your compositor unique" [32]. It is clear from this description that, despite being created for sway, wlroots was not intended to be a mere internal component, but rather, a useful library for writing Wayland compositors.

Regarding the extra functionality a Wayland compositor must have (again, compared to an X window manager), wlroots implements most of it, but also leaves out a fair amount: in the same blog post that announced wlroots [4], its main developer wrote that "wlroots is much more powerful [than wlc], but at the cost of putting a lot more work on the shoulders of sway". This is true not only for sway, but for any compositors using wlroots: they must

implement, themselves, some of the code that could have been present in wlroots. It is the goal of this work to identify such code in sway and migrate it to a new library, so that, using this new library, Wayland compositor developers can actually focus on window management.

1.4 Basic terminology

Before we proceed, let us first introduce some basic terminology that will be used in the remaining of this text:

- **Tiling window scheme**: a window management scheme in which most windows are not allowed to overlap, and usually occupy all available screen space.
  - **Floating window**: in many tiling window schemes, some windows may actually be allowed to overlap any other one, and not necessarily take up all available screen space. Floating windows are often rendered "on top of" the other windows, in a way that hides those windows' contents where they overlap with the floating window's contents. Therefore, these windows appear to be "floating" above the rest, hence their name.
  - **Tiling window**: in a tiling window scheme where floating windows may exist, any window which is not floating is a tiling window.
- **Focus**: a window is said to have **keyboard focus** when the compositor or X server directs the input from the keyboard to this window. Similarly, a window is said to have **mouse focus** when the input from the mouse (or similar pointer device) is directed to it. When a window has both keyboard and mouse focus, it is simply said that the window has focus, or is **focused**.
- **Output device**, or simply **output**: any physical device that can serve as an electronic screen, including monitors, TVs, a mobile device's screen, etc. Therefore, we shall use "output" and "screen" interchangeably. Note that in this work, "output" does **not** refer to a process's standard output.
  - **Refresh**: the act of updating the screen contents. This operation occurs at fixed time intervals, even when the image to be displayed has not changed. The reciprocal of this time interval – that is, the amount of times a refresh occurs per time unit – is called **refresh rate**, usually expressed in Hertz (Hz).
  - **Frame**: the screen contents produced by a refresh operation. A frame is created at a refresh and exists until the next one, which produces a new frame. The new frame is considered distinct from the previous one even when there is no change to the image displayed.
    As an aside, since each refresh operation produces a new frame, refresh rates are sometimes expressed in frames per second (FPS) instead of Hertz, but these units are equivalent.
  - **Resolution**: if the screen resolution is W x H, then the screen image is a grid of pixels with W pixels of width and H pixels of height.

  - **Output mode**: a unique combination of resolution and refresh rate. A change to one of these values means the mode has been changed, even if the other value remains unchanged.

- **Input device**, or simply **input**: any physical device that allows users to interact with the graphical systems. This includes, but is not limited to, devices such as:
  - Keyboard (physical, not on-screen);
  - Mouse, touchpad or similar pointer device;
  - Touchscreen;
  - Drawing tablet.

  Also note that in this work, "input" does **not** refer to a process's standard input.

- **Dependency**: we say a function `F1` *depends* on a function `F2` if moving the source code of `F1` to a separate library would give a linking error unless the source code of `F2` is also present in this library. (This usually happens when `F1` calls `F2` directly, or when there is a chain of function calls from `F1` that ultimately results in a call to `F2`).
  - **Dependency cycle**: a *dependency cycle* is a chain of function calls starting from `F1` that ultimately calls `F1` again.

1.5 Overview of sway's inner workings

To better grasp how much work wlroots delegates to sway, and how much of it is unrelated to window management, we now provide a general overview of sway's code, highlighting to which extent generic compositing code is intertwined with sway's particular window management scheme.

In this overview, we shall focus on functionality that may be of use to other Wayland compositors. Therefore, code which is far too specific for sway will not be covered here. (This is the case of the configuration file parser, the commands that can be passed to sway, among a few others.)

1.5.1 Sway's window management scheme

Window management is precisely the non-reusable part of a Wayland compositor, which could not have been provided by a separate library. However, in sway's current state, its window management scheme permeates most of its reusable code, so we shall examine the former first, before explaining the latter.

Sway was created to imitate i3's tiling window management scheme, which can be described as follows:

- The root (`sway_root`) has the list of all connected outputs;
- Each output (`sway_output`) has a list of workspaces;
- Each workspace (`sway_workspace`) has a list of containers;

- Each container (`sway_container`) has one of:
  - A view (`sway_view`), which represents a tiling window;
  - A vertical split with one or more other containers;
  - A horizontal split with one or more other containers;

In sway, a workspace has one out of three possible layouts: stacked, tabbed, or split. However, we will leave out the first two and concentrate on the split layout.

Container splits work by dividing the available area in the screen by $1/N$, either horizontally or vertically, where N is the number of windows in the split. Therefore, the `sway_container` type, which represents containers in sway's code, is implemented as a tree structure, where the leaves are views.

An example of how tiling windows can be arranged in i3 (which is compatible with sway) is shown in figure 1.1, along with its respective container tree in figure 1.2.

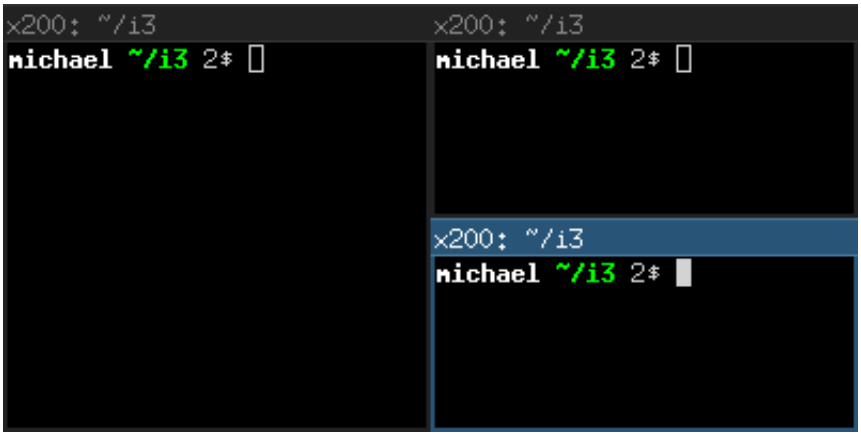

Figure 1.1: *Screenshot of terminal windows arranged in i3. The focused window has its decorations in blue. Source: [7]*

Finally, sway also allows floating windows – but we'll leave them aside for the sake of simplicity.

1.5.2 Transactions and `sway_node`

One of Wayland's goals is to make it so "every frame is perfect", and to that end, changes must be done *atomically* [18]. One way to achieve this is, instead of directly making changes that would affect a frame, to store those changes in a *pending* state, and to commit those changes – that is, copy them to the *current* state – all at once [18]. This strategy is called "transactional update".

Transactional updates are implemented for three of the types mentioned in the window hierarchy in subsection 1.5.1, namely: `sway_output`, `sway_workspace` and `sway_container`. Pending state data is stored directly as attributes of each of those types, but current state data is stored in a dedicated field, aptly named `current`. For instance, if `container` is a pointer to a valid `sway_container`, then its *pending* width is `container->width` and its current width is `container->current.width`.

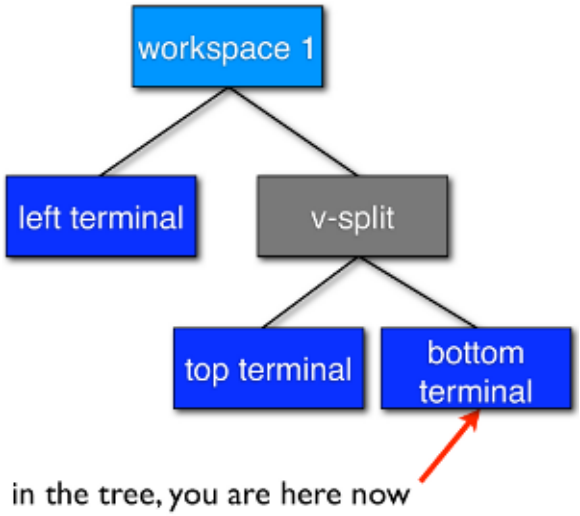

Figure 1.2: *The corresponding container tree layout, with the arrow indicating the focused window. Source: [7]*

Since each of the three aforementioned types store different data, their respective `current` fields have different types: `sway_output_state`, `sway_workspace_state` and `sway_container_state`, respectively.

Transactions are implemented for three different types – outputs, workspaces and containers – but, instead of implementing three versions of each transaction-related function that needs to know which of these types it is dealing with, the type `sway_node` is used instead. This type is a tagged union: it has a `union` field that can hold a pointer to one out of four possible types – the three aforementioned ones, along with a fourth alternative, `sway_root` – and it also has an `enum` field to tell which type the `union` holds. Each of these four types also has a field `node` of type `sway_node`, so it is possible to access the node from them and vice-versa.

Normally, when a pending state is changed, its node is marked *dirty* with the aptly named `node_set_dirty` function, and later on, all dirty nodes pending states will be committed by calling `transaction_commit_dirty`.

Finally, `sway_node` also plays a role in the focus stack, but we'll save the details for subsection 1.5.3.

The entire infrastructure of transactions and nodes is independent of window management and could, in theory, be entirely moved to a helper library, such as the one we intend to create.

1.5.3 The focus stack

Sway is compatible with i3, and therefore, its default focus policy is the same as i3's default focus policy.

By default, i3 has a focus-follows-mouse policy, meaning that "window focus follows your mouse movements as the mouse crosses window borders" [7]. Note, however, that

this does **not** imply that the focused window is always the one under the cursor: instead, this policy dictates that if the mouse moves from window `A` to window `B`, then `B` receives focus – but `A` was not necessarily the window that had focus prior to this. For example: by default, i3 gives focus to newly created windows, which do not necessarily spawn under the cursor, and that can lead to the situation shown in figure 1.3, where the cursor is to the left (circled in red), and focused window is to the right (with decorations in blue).

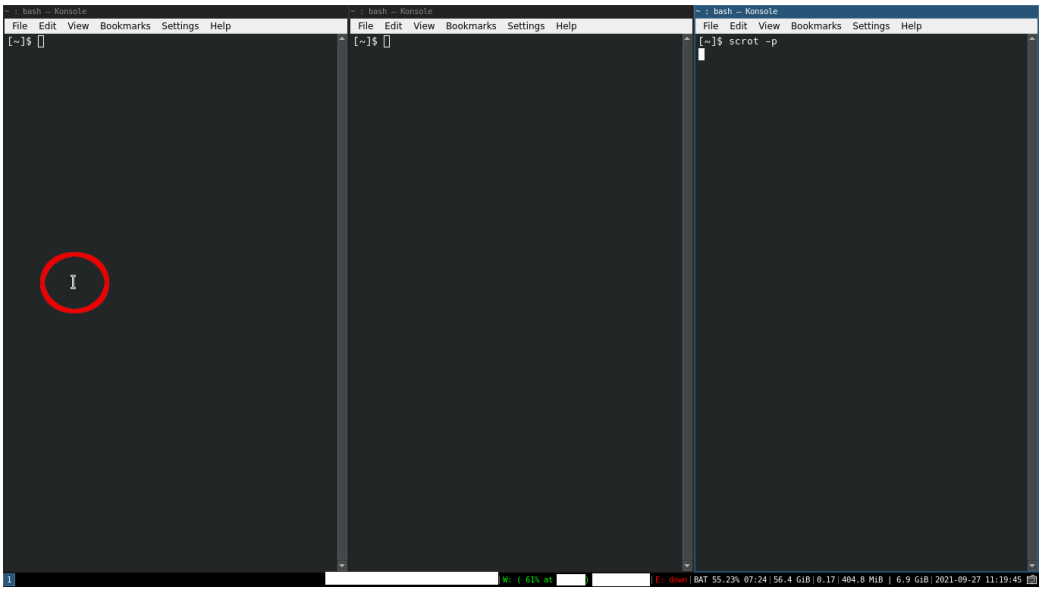

Figure 1.3: *A screenshot of three terminal windows in i3, where the cursor is circled in red and the focused window has its decorations in blue. IP addresses and network names have been censored. Source: original screenshot obtained with i3's default configuration and the command* `scrot -p`*, shown in the focused terminal.*

Another noteworthy situation happens when the currently focused window is closed. Naturally, focus has to change to another window, and again, the window chosen to receive focus at this situation is not necessarily the one under the cursor, as shown in figure 1.4, which shows the same windows of 1.3 after closing the focused terminal. The window chosen to receive focus in that situation is whichever window had focus previously, before the window that has now been closed received focus.

Note that this process can be repeated indefinitely: one can continue to close the focused window from the situation of figure 1.4, in which case focus must again be transferred to the last window to receive focus before the currently focused one, and so on until there are no windows left. Therefore, to implement this policy, sway must keep track of all windows that received focus. Since the window to receive focus most recently is the one to receive it when the currently focused window is closed, sway keeps track of the windows that receive focus in a stack: the **focus stack**. The element on top of this stack is the currently focused one; the one before it is the one to receive focus when the currently focused one is removed, and so on.

Although only windows can have focus, sway allows elements of the focus stack to be either workspaces or containers (the latter may represent windows, as mentioned in section 1.5.1). A workspace can only be on top of the focus stack (and, therefore, the

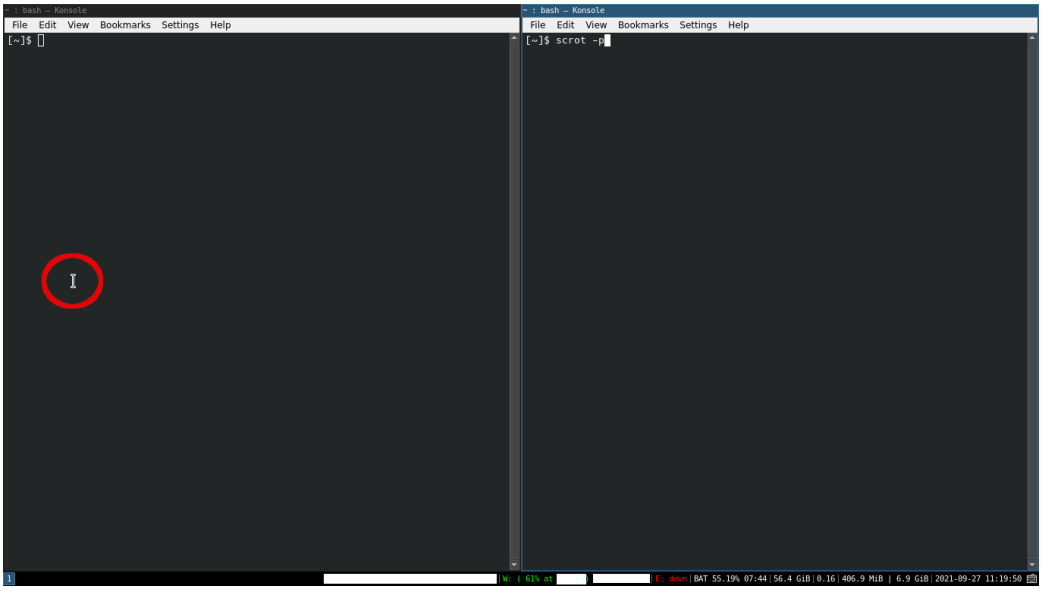

Figure 1.4: *A screenshot of two terminal windows in i3, showing the layout obtained after closing the focused terminal in figure 1.3. The cursor is circled in red and the focused window has its decorations in blue. IP addresses and network names have been censored. Source: original screenshot obtained with i3's default configuration and the command* `scrot -p`*, shown in the focused terminal.*

"focused" element) when

- it is the workspace currently being shown on screen; and
- it has no windows.

When this situation occurs, no windows have focus.

The reason workspaces are allowed in the focus stack is the fact that sway uses the focus stack not only to determine which window has focus, but also to determine in which workspace a newly created window must be placed. This is done by function `select_workspace`, which, under sway's default configuration, chooses

- if the focus stack top is a workspace, this workspace;
- if the focus stack top is a container, this container's workspace.

Since both workspaces and containers can be part of the focus stack, it is implemented as a list of `sway_node` entries. However, a node that represents root or an output is not allowed in the focus stack.

As explained in section 1.4, window focus is determined by both keyboard focus and mouse focus, which means focus is tightly related to input devices. As we explain in section 1.5.5, a Wayland seat can, in theory, have only up to one keyboard and one pointer device, and therefore, sway keeps one focus stack per seat.

While the decision of which window receives focus is a window management one, a helper library could still provide the focus stack implementation, with helper functions to get or set focus, leaving to the compositor the decision of when and how to use them.

1.5.4 Output devices

Output devices may be plugged or unplugged at any time of the compositor process lifetime; whenever either event happens, the compositor must react accordingly.

Whenever an output device is unplugged, the compositor must reassign any pointer to the no longer existent output (or else the pointer will become dangling), and destroy any resources it used. Likewise, when an output is plugged, some bookkeeping work must be done, such as adding the new output to `sway_root`'s list of outputs, creating a new node for it, among a few more.

The work mentioned thus far is unrelated with window management; there are, however, some window management decisions to be done in terms of output insertion and removal. For instance, when an output is connected:

- Should any windows be shown on it? Which ones? In which position?
- Should the mouse cursor move to the newly plugged screen, or stay where it is?
- When a new window is created, should it be displayed on the newly plugged monitor, or on a different one?

Among other possible decisions. Similarly, when an output is disconnected:

- If there were any windows being displayed on it, should these windows be displayed in the one or more of the remaining screen(s)? If so, which ones, and in which position in those screens?

A helper library could provide the bookkeeping part of this code, leaving these decisions to the compositor.

Another interesting aspect regarding sway's implementation of output devices is the output configuration, which allows setting properties of a screen such as resolution and refresh rate. Sway's implementation has a reasonable amount of code to set or otherwise manage these properties; this code could also be present in a helper library, delegating the decision of when and how to use them to the compositor.

1.5.5 Seats and input devices

Unlike output devices, which always serve as electronic screens, input devices can be of many different types, such as keyboard, mouse, touchscreen, among other options. Conceptually, a Wayland seat – a type meant to represent a single physical seat, in which users sit to operate a computer using a keyboard, mouse, among other input devices – is capable of having up to one of each different type of input device associated with it. In practice, however, Wayland compositors often use a single seat for multiple input devices of same kind [21]. Either way, Wayland seats and input devices are closely related, not only conceptually, but in sway code as well.

In the same way as output devices, input devices may also be plugged or unplugged at any point of the compositor lifetime, and properly dealing with these events is yet another responsibility of a Wayland compositor which is, to a large extent, unrelated to window management, and could be implemented in a helper library.

1.5.6 XWayland and desktop shells

Wayland provides a backward-compatibility mechanism with X11, named XWayland. This is so that X clients can run in a Wayland compositor without being changed to speak the Wayland protocol (and therefore become Wayland clients instead). Sway has code to implement the XWayland protocol, but, as this is mostly unrelated to window management, this implementation could also be provided by a helper library.

Additionally, since Wayland is a very concise protocol, Wayland clients often need to speak one or more protocol extensions. One noteworthy kind of Wayland protocol extension is the *desktop shell*, which define important aspects of semantics of GUI windows, such as pop-ups, context menus, among others. The most prominent Wayland desktop shell is the XDG[3] shell, but some Wayland clients also use the now deprecated [28] `wl_shell`, provided by the Wayland project. Sway also needs code to deal with the desktop shell Wayland clients might use, and that code, once again, has little to do with window management decisions and could have been provided by a helper library.

1.6 The Wayland Protocol

Wayland describes an IPC[4] protocol for message-passing between two entities: a server (the Wayland compositor) and a client (any Wayland client). Messages can be either *events*, when going from server to client, or *requests*, when going from client to server [23]. Both types of messages act on *objects*, each of which has an *interface* to describe the events and requests associated with it, along with their signatures (the number and types of arguments each one needs) [20].

Every Wayland object has a 32-bit integer unique identifier (ID), which is used to reference this object in messages [23]. The object ID 0 (zero) is reserved for a null object [23], while the ID 1 is reserved for a special object, the Wayland display singleton (`wl_display`) [23][19]. For all other objects, IDs are negotiated between server and client through the `wl_display` interface.

All Wayland objects and their interfaces are described in a single XML file, `wayland.xml` [22]. In addition to the already mentioned `wl_display`, this file defines interfaces for many other objects, notably: surfaces and pixel buffers, output and input devices, and the compositor itself.

Wayland requires transmitted data to be a stream of 32-bit values with the same endianness as the one used by the host CPU [19]. Strings and arrays must be padded to 32-bits, and must begin with a 32-bit integer indicating their length [19]. It should be noted, however, that the Wayland protocol does not define the means through which data should be transmitted; Unix sockets are typically used for this purpose [23].

[3] Cross-Desktop Group

[4] Inter-Process Communication

Chapter 2

Objectives and Approach

In Chapter 1, we have presented the fundamental concepts for this work, highlighting the fact that Wayland compositors are more complex than X window managers, and that this extra complexity has prompted the creation of a few helper libraries, such as wlroots. However, as we have also highlighted in Section 1.5, wlroots still leaves much of this complexity to be dealt with by compositors themselves. Therefore, there is still room for another helper library, built on top of wlroots, that may facilitate creating Wayland compositors even further, so compositors will need to implement very little other than window management related functionalities.

Creating such a library – which we have named **wlstem** – is precisely the goal of this work. Further, since sway is wlroots's most prominent user, we have created wlstem from sway's code.

In this chapter, we present the approach we envisioned for the creation of this new library, along some challenges that had to be surpassed in this process, and the features we deem necessary or desirable for wlstem.

2.1 Creation process overview

Wlstem should be free of code that is specific to a certain window management scheme. However, it is clear from the overview presented in Section 1.5 that sway's window management code is considerably intertwined with code unrelated to window management. As such, before incorporating any part of sway's code into wlstem, we first had to separate from it any window management specific code; in most cases, this has required considerable refactoring work.

Prior to this endeavor, we have deemed useful to take an initial step of simplifying sway's implementation, by removing a part of its code which is clearly not reusable for other compositors: doing so has allowed us to work with a smaller, more manageable code base.

Then, we have taken a small part of sway's code and made it into an initial version of wlstem. This step involved moving such code to separate files when necessary, and

changing the sway build system to generate an actual library, to which the remaining of the code was linked. By the end of this step, we had a small, independent library, and the (simplified) sway program; the latter has been used throughout development to test and debug wlstem code, and also serves as a proof-of-concept implementation of a Wayland compositor using wlstem.

The rest of the creation process has been done in an iterative way, by repeating the steps roughly described below:

- Select a part of sway's remaining code that has some reusable feature;
- Refactor this code so that it is not restricted to sway's window management scheme, and does not call any code that is;
- Move this code to wlstem.

This process has been repeated as many times as possible, given the time constraints.

2.2 Challenges

In this section, we briefly show some of the challenges that had to be faced in our work of creating a new library out of sway's source code.

2.2.1 Breaking dependencies

Suppose `F` is a function we wish to move from sway to wlstem. Before doing so, `F` needs to be examined to ensure there is no window management code in it – and if there is, this code must be moved elsewhere. Therefore, it might be reasonable to move window management code to a separate function – say, `F2`.

However, simply introducing a call to `F2` in `F` would not solve the issue, since `F` would then depend on `F2` (as defined in Section 1.4), and thus migrating `F` to wlstem would result in a linking error, unless we also migrate `F2`. However, `F2` contains window management code, which is precisely what we wish to keep away from the library.

Therefore, in this scenario, we must also break the dependency between `F` and `F2`. To do so, two different approaches can be taken:

1. Make wlstem demand from users the window management specific code through a callback, and make sway provide `F2` as the callback; or
2. Make wlstem signal to that some event occurred, and make sway subscribe to the event and call `F2` when it occurs.

Wayland conveniently provides two useful types for implementing strategy 2: `wl_signal`, which allows signaling that a certain event occurred, and `wl_listener`, which allows subscribing to a certain event and calling a function when it occurs.

However, both strategies have been used in wlstem: 1 has been used when we deem it strictly necessary to perform a task at this event, but the task is window management

dependent; 2 has been used when there is no direct need for action at the given event, but the compositor may want to call window management specific code then.

2.2.2 Dependency cycles and long chains

Suppose `F1` is a sway function we wish to migrate to wlstem. Then, for every function `F` on which `F1` depends, `F` needs to be migrated to wlstem before, or at the same time that `F1` is migrated – otherwise, we would have a linking error. However, if we have a dependency cycle, the *entire* cycle must be migrated simultaneously, or else there will be a linking error.

Since cycles can be fairly long, and not all functions in it might be relevant to migrate to wlstem at a given time, it may be unwise to examine and eliminate window management code from all functions of a cycle, and only migrate them at the end of this process. Instead, it might be more reasonable to break the dependency cycle at some point, using one of the strategies outlined in Section 2.2.1. Breaking dependency cycles can also lead to clearer code, since functions become more loosely coupled.

A similar issue occurs when we wish to migrate a function `F` which calls a long chain of functions that have not yet been migrated to wlstem. As in the previous situation, instead of migrating all of the functions that `F` depends on before migrating `F` itself to wlstem, it might be more reasonable to break the dependency chain.

2.3 Automated testing

Sway does not have automated tests of any kind: all testing is done manually. This is partially explained by the fact that sway code largely relies on the events signaled by wlroots. Many of these events come from the so-called backend (corresponding to the `wlr_backend` type), and are fired on a variety of situations, such as when hardware devices are inserted or removed. There are also events associated with hardware devices themselves: a keyboard device has events for key press and release, among others; a pointer device has events for motion and clicking; and so on.

Therefore, an automated test suite for sway would require (at least) a mock wlroots backend, whose events would be emulated, along with emulated hardware devices. For instance, the mock backend could provide a function for emulating the insertion of a keyboard device, and another for emulating a key press on this device. These functions could then be called by an automated test suite.

Unfortunately, as of this writing, wlroots has no such thing as a mock backend, or mock hardware devices. Implementing those features in wlroots might pave the way for automated tests, which would greatly facilitate debugging; however, to do so would require a considerable amount of code change in wlroots. Therefore, this lies outside the scope of this work.

Since we do not have an automated test suite, to ensure no bugs have been introduced as we refactored sway code and transferred it to wlstem, we had to resort to comprehensive manual testing after nearly every change we have made. We were particularly thorough

testing situations where the code we had changed at that point was expected to be invoked: for instance, when changing code dealing with how output devices were disabled, we have connected and disconnected monitors in a wide variety of situations while the compositor was running, to ensure it would work exactly as it did prior to that code change.

2.4 Expected features

The features we believe wlstem must have are as follows:

- Wlstem must make hardware hotplugging transparent, except for window management decisions. In other words, Wayland compositors using wlstem should not need to have any code to handle hardware devices being plugged or unplugged, and this must apply for both input and output devices. However, the user may have to supply code to deal with window management decisions that arise in such situations. For instance: when an output is unplugged and there were windows being shown in it, should they be rendered at ("moved" to) a different screen from this point on? This decision is considered a window management one.
- Wlstem must abstract which protocol or desktop shell is being used by clients, so that Wayland compositors using wlstem do not need to implement code differently depending on whether the graphical program is a native Wayland client, or an X client using XWayland. Further, if the client is indeed a native Wayland client, it should not be necessary to write code differently to handle the desktop shell protocol used by the clients (such as `wl_shell`, `xdg_shell`, among others).

While non-essential, some other features are also very important:

- Wlstem should provide a focus stack implementation, along with helper functions for focus-related operations, such as setting focus to a window, get the focused window, get the previously focused window, among other possibilities. Note, however, that even if wlstem provides these helper functions, it should never decide which window gets focus.
- Wlstem should provide functions to perform common window management tasks, such as hiding or maximizing windows – however, once again, the library should only provide helper functions, and never decide when any of these tasks should be done.

Finally, there are some other interesting features that wlstem might have:

- Wlstem could have functions to enable/disable useful wlroots protocols, such as those for screen capture, screen casting, clipboard, etc.
- Wlstem could have an API[1] to allow creating animations, so the compositor could choose to display animations at situations such as when focus is switched between windows, when workspaces are switched, among many others.

[1] Application Programming Interface

Chapter 3

Development history

In the previous chapters, we have outlined the context and goals of this work. Here, we describe our development effort in a roughly chronological order, along with the resulting changes to the codebase.

3.1 Simplifying sway and creating wlstem

Our codebase[1] has been forked from sway at commit `38571e6a0c134ea8ea5d8dc2e7c50ef37085ae83 "Log when config file is not found"`, from January 17, 2021.

The very first action taken was to remove any documentation that no longer applied to wlstem, such as `README` files and manual pages, along with other files that did not contain source code and would be no longer useful, including logo images, the desktop entry, the example configuration file, among others.

In the next subsections, we detail how we proceeded to simplify sway further and to create wlstem from it.

3.1.1 `swaybar`, `swaynag` and `swaymsg`

Sway has three auxiliary programs that, while not being a part of sway itself, are closely related to it, and are therefore provided along with sway's code: `swaybar`, `swaynag` and `swaymsg`. They are meant to mirror the functionalities of the i3 equivalent programs `i3bar`, `i3-nagbar` and `i3-msg`, respectively, and work as follows:

- `swaybar` renders a bar on an edge of the screen (by default, the bottom one), in which arbitrary text can be displayed – though it is generally used to display system information such as battery levels, network status, current workspace name, and so forth. Its i3 analogous program, `i3bar`, is shown at the bottom of figure 1.3.
- `swaymsg` is a command-line program that allows sending commands to sway: if `foo` is an internal command sway understands, one can usually run `swaymsg foo` on a

[1] https://github.com/luamfb/wlstem

command line to send `foo` to sway. (One could also bind `foo` to a key shortcut using sway's configuration file, but we'll discuss that in Section 3.1.2.) Sway commands generally either retrieve information from sway, or request sway to take a certain action. Among the latter kind is the internal command `exit`, which is the only way of cleanly exiting sway.

- `swaynag` displays a simple graphic pop-up message, which may include clickable buttons for user feedback. Under sway default configuration, when the user presses the `Ctrl-Shift-E` keybinding, this program is used to display a confirmation pop-up prior to exiting sway. (Exiting sway is done with the command `swaymsg exit`, as explained above.)

It should also be noted that sway also has an IPC[2] implementation, which is needed for some of the programs above to work.

Since none of these programs are related to what wlstem is intended to do, we have removed them all. First, we removed all uses of `swaynag`, including the one that occurred when exiting sway (which required confirmation to exit after pressing `Ctrl-Shift-E`); we then proceeded to remove swaynag's source code. After that, we have written code that, when a certain (hardcoded) special key is pressed, calls the same code path that was called when sway received the command `exit` from `swaymsg`: this way, we could cleanly exit sway without relying on `swaymsg`. Then, we proceeded to remove all `swaymsg` code. Finally, we removed the last auxiliary program, `swaybar`, along with the sway's IPC code.

3.1.2 Configuration and commands

i3 is extensively configurable through a configuration file [7], and, as a part of sway's backward-compatibility with i3, it also understands the same configuration file. As such, sway needs code to parse this file and store its contents; additionally, at several points in sway's code, the stored configuration needs to be checked so sway can actually behave as the user requested.

Additionally, sway internal commands may also be present in the configuration file, where they can be bound to key shortcuts. Therefore, sway also needs code for parsing these commands, which includes ensuring the number and kind of arguments is the same as expected by each of them.

While certainly useful for sway users, the aforementioned code configuration and command code is too specific for sway and, as such, serves no purpose for wlstem, but clutters the codebase instead.

The first step in removing this code was to stop loading the configuration file. To that end, we first removed the file (syntactic) validation; then, we removed the feature that allowed the configuration file to be reloaded while sway was running; finally, we removed the code that loaded the configuration file, leaving only the default options in use. The code to validate and parse that file was unused by then, and thus we could easily remove it.

[2] Inter-Process Communication

Then, we proceeded to remove each one of the various sway commands. We have also removed code that stored and replaced variables in the configuration file, along with code for the so-called "criteria", which made certain commands meant to be run for every window so that they would run only for the windows matching such criteria.

However, there were two sway commands that stood out as useful, and as such, they were modified rather than removed. The first is `exec`, a command such that `exec foo` makes sway run `foo` as if by running `/bin/sh -c foo` in the command line – however, its implementation used a double `fork`, such that `foo` would run as a daemon [14]. This command was turned into a separate function that did not use any code related to sway commands. The next useful command was `bind`, which allowed binding a sway internal command to a key shortcut (so that the command would be sent every time the user typed a shortcut). This was changed to bind function pointers instead of sway commands (so that the provided function would be called when a shortcut was pressed). After doing so, commands and their related types were removed entirely.

It is worth noting that the structure that was used to store sway's configuration options, `sway_config`, remained in use even after we were only using hardcoded default values. This is because, despite its name, the `sway_config` type is also used to store some of the compositor's global state. We also kept code for configuring certain options for each output or input devices, but for different reasons: it might be worth allowing users of wlstem to call such code and set these options themselves, whenever they see fit.

It is also worth noting that we have used the modified `bind` command to bind the shortcut `Ctrl+Alt+Backspace` for exiting sway, as opposed to the (single) special key that was being used up until then, and also to spawn a terminal with the shortcut `Alt+Enter`. (The terminal can then be used to spawn any other program, even a graphical one; therefore, this approach eliminates the need to implement a launcher.)

3.1.3 Window management simplifications

Wlstem should be free of window management code; however, during this code simplification step, we have found it useful to also simplify sway's window management scheme (which we have briefly explained in section 1.5.1), since doing would make it easier to separate the reusable parts of sway's code from its window management code. Additionally, the code that is not migrated to wlstem will become a reference implementation for this library, and as such, by simplifying the window management scheme we are also making our reference implementation simpler.

The very first simplification we have made in terms of window management code was to remove sway's stacked and tabbed layouts. We also removed vertical splits, so that all splits were horizontal from this point on – meaning that tiling windows would occupy the entire screen height, but only a fraction of the screen width. Then, we proceeded to remove the code for resizing tiling windows. Additionally, we have removed floating windows altogether, so that all top-level windows would be tiling windows from then on.

Then, we proceeded to eliminate some workspace properties: first, we removed the so-called "representation": a string meant to summarize information about a workspace, the containers in it, and their rough geometrical disposition in the workspace. Then, we

removed the "urgent" boolean property, which meant whether or not a window in that workspace has requested to be considered "urgent", which is a way for windows to signal that they need attention from the user.

We then removed fullscreen windows, along with hints meant to indicate when and how to display pop-ups when there is a fullscreen window.

We have also removed the sway's "output priorities" feature: as explained in section 1.5.1, each output has a list of workspaces, and thus, when a new workspace was created, an output device had to be chosen so the new workspace would belong to it. With output priorities, one could increase the chance of a certain output device to be chosen to have a specific workspace. For the sake of simplification, we have removed this feature as well.

After that, we have changed the tree-like structure of containers, so that a container could not have other containers as children, and all containers would be direct children of the workspace they were in.

Then, we have changed the `apply_horiz_layout` function, responsible for arranging containers in a horizontal layout, to ensure the containers shown in a workspace could have the same fraction of the screen width (that is, if a workspace has N windows, then each should occupy $1/N$ of the screen width), as this wasn't necessarily the case before.

Workspace simplifications

At this point, we have simplified workspaces even further, so that we could remove them altogether later on.

To that end, we first wanted to make it so we would always have a single workspace for each output. At that point, the only way to have more than one workspace was when an output device would be disconnected: then, it would have its workspace transferred to another output. Therefore, we have changed the code so that, instead of transferring the workspace to the another output, the workspaces' containers would be transferred to that output's workspace instead.

Then, we have enforced the one workspace per output rule, by aborting execution if a workspace were ever added to an output that already had an output; after thorough testing to ensure this never happened in practice, we then changed the outputs' workspace list to a single workspace.

Also to enforce the aforementioned rule, we have changed the circumstances leading to workspace destruction: we have made it so the workspace would only be destroyed when its output owner was also destroyed.

Then, we proceeded to move nearly all of the workspace's remaining fields to the output, namely: the (x, y) coordinates, width and height, and list of containers. Likewise, containers, which previously had a pointer to the workspace they belonged to, were changed to hold a pointer to an output instead.

Despite all these simplifications, workspaces were not yet removed at this point, as their nodes still played an important role in the focus stack (as we have outlined in section

1.5.3).

3.1.4 Creating wlstem

After all these simplifications, we finally sought to create an actual library from sway code. We have created wlstem from two small files: `log.c`, which contained sway logging functions, and `exec.c`, which housed the code for the adapted `exec` sway command (this adaptation has been explained in section 3.1.2). To that end, we have first moved these files to a new directory; then, we have changed sway build system so it would generate a dynamic library from these files and link the binary generated from the rest of sway code to the new library.

We have also created a README file explaining what this project was, and how to build and run it.

While wlstem has been created with a small amount of code, we would gradually move code from sway to it later on.

3.1.5 Removing workspaces

After creating wlstem, we have turned once again to workspace simplification, so we could remove workspaces from sway at last. As we have mentioned in section 3.1.3, workspaces were not yet removed at that point because their nodes were still in use in the focus stack; as such, we have turned our attention to nodes and the focus stack implementation.

As we have mentioned in section 1.5.2, the `sway_node` type can represent four possible types – `sway_root`, `sway_output`, `sway_workspace` and `sway_container` – but only the last three actually have transactions associated with them, and only the last two could be a part of the focus stack. The root node was only used by some functions that happened to receive a `sway_node` argument but needed to be passed the root sometimes. As such, the first simplification we have done was to eliminate all usage of the root node, and change the `sway_node` type so it could no longer represent `sway_root`.

At this point, we proceeded to remove workspace nodes as well, by replacing them with their output's node. This has caused crashes in sway in a variety of situations that arose when output devices were unplugged. This prompted us to improve logging in functions related to the focus stack, which, along with the core dumps generated by crashes, allowed us to diagnose the issue: when an output device was unplugged, its data would be destroyed, so its output node would have a dangling pointer. This did not occur while using workspace nodes because, when an output was unplugged, the workspace would be detached from the output (meaning it would no longer have a pointer to it), but the workspace itself would not be immediately destroyed after being detached. Our solution to this was to create a new function to iterate through the focus stack and remove any nodes belonging to a certain output, then call this function whenever an output was detached (but had not yet been destroyed).

Then, we proceeded to change any functions that still received a workspace as argument so they would receive an output instead, and removed the workspace transaction

implementation, along with the type `workspace_state` it used to use (as explained in section 1.5.2).

At this point, workspaces were only used as a field `active_workspace` in `sway_output`: an output would be considered active if and only if its active workspace was not `NULL`. As such, we replaced this field with a boolean `active` field with the same semantics (that is to say, the `active` field would be assigned `true` or `false` at the same circumstances the previous `active_workspace` field would be assigned a valid workspace or `NULL`, respectively). After this change, we proceeded to remove workspaces from sway entirely.

3.1.6 Removing example client

Before starting to migrate code from sway to wlstem, our final simplification was to remove the example Wayland client sway had: it was useless in practice, since we have been debugging with actual Wayland clients for a while at this point.

It is also worth noting there was a directory in sway named `common`, for code that was used by sway, the example client, and possibly by some of the auxiliary programs that we had already removed by then (that is, `swaybar`, `swaynag` and `swaymsg`: see section 3.1.1). We have moved all code from this directory to wlstem.

3.2 Migrating code to wlstem

At this point, we had made several simplifications to our codebase and created the new library, wlstem, from a small amount of its code. We have then proceeded to move code from sway to wlstem, as detailed in the following subsections.

Before we describe the code changes in the following subsections, it should be noted that some pseudo-namespace conventions are in use by sway, wlroots and Wayland. Namely, any type (or function) starting with the prefix:

- `wl_`
- `wlr_`
- `sway_`

Is declared (or implemented), respectively, in Wayland, wlroots, or sway.

We have followed this convention by adopting the prefix `wls_` for symbols declared in wlstem. (It should be noted, however, that some symbols migrated from sway to wlstem have retained the `sway_` prefix.)

3.2.1 Wlstem context and nodes

The very first thing we have done in this step was to create the type `wls_context`, a singleton that holds all data associated with wlstem (which we will call the “wlstem context” from now on). Along with this type, we have created the functions `wls_init` and `wls_fini`, to initialize and destroy the wlstem context, respectively. We have also created

the `wls_node_manager` type for dealing with sway nodes (which we have explained in section 1.5.2), and stored an instance of this type in `wls_context`.

The list of "dirty nodes" (also explained in section 1.5.2) was then migrated from the `sway_server` type to the new type `wls_node_manager`.

We then proceeded to migrate the type `sway_node` itself (both headers and implementation) to wlstem, renaming it as `wls_transaction_node` due to its role in the transaction implementation.

Finally, we have migrated the `new_node` event (which, as the name implies, is fired when a node is created) from the type `sway_root` to `wls_node_manager`.

3.2.2 Output-related types, `server_wm` and `wls_server`

In sway, all screens are arranged in a bi-dimensional coordinate space, named output layout. When a change was made to this output layout, an event would be fired, and sway had two different handlers for this same event: `handle_output_layout_change` and `output_layout_handle_change`. We have unified these handlers by turning the latter into a new function, named `arrange_output_layout`, which was called by the former. The new function had only window management specific code, while the remaining handler had none. Later on, we would make it so that, instead of having `handle_output_layout_change` call `arrange_output_layout` directly, it would emit an event, to which a new type would listen and call `arrange_output_layout`. This new type, `server_wm`, was meant to deal with explicitly window management dependent tasks and was implemented on sway, because we do not want any such code in wlstem.

We have then migrated to wlstem the sway type `sway_root`, which implemented the root in the window management hierarchy, as explained in section 1.5.1. We would later move the output manager (a field of type `wlr_output_manager_v1`, from wlroots) to `sway_root`, and rename `sway_root` as `wls_output_manager` since it concentrated all output manager functionalities by then.

Then, we have migrated the most widely used fields of the type `sway_server` – namely, the Wayland-related `wl_display` and `wl_event_loop`, and the wlroots-related field `backend` (of type `wlr_backend`) – to a new type, `wls_server`. We have done this instead of migrating `sway_server` directly because `sway_server` had a fairly large amount of fields, such that it would not be viable to migrate it in its entirety. Also, we have added an instance of the new type to `wls_context`, and changed the functions that accessed these fields from `sway_server` so they would use `wls_server` instead.

Finally, we have migrated to wlstem the type `output_config`, and moved the list of output configurations to `wls_context`. This type stores several configurable parameters for monitors – notably, resolution and refresh rate.

3.2.3 Input code and `sway_debug`

The code related to output devices also depended on some of the code related to input devices, which we therefore proceeded to migrate to wlstem, after a few organizational

code changes (which are not worth mentioning). Simultaneously, we have migrated some input-related code on which output devices did not depend, but which we also wished to have in wlstem.

The first input-related code we migrated to wlstem was sway's implementation of input methods. Input methods allow creating input method editors (IMEs), which can map the characters typed on the keyboard, usually letters of the Latin alphabet, into a multitude of other characters, which are usually not present in most keyboards. As such, IMEs are particularly useful for writing in some Asian languages such as Chinese and Japanese, since these languages have tens of thousands of different characters (and therefore it would not be viable to create a keyboard with all possible characters). One such IME is `mozc`, shown in figure 3.1.

Figure 3.1: *Screenshot of the input method editor `mozc` in a Linux terminal application, after pressing the keys K-I-K-A-N, then Space. The IME displays a list for selecting one out of several possible Japanese words matching the pressed keys. Source: own screenshot.*

Wlroots provides two types to implement input methods: one for an input method manager (`wlr_input_method_manager_v2`) and one for a text input method (`wlr_text_input_manager_v3`). The type `sway_server` had a pointer to an instance of each of these types – both of which we have migrated to a new wlstem type, `wls_input_method_manager`, and we have added an instance of this new type to `wls_context`.

Afterwards, we have moved `sway_server`'s field `tablet_v2`, of type `wlr_tablet_manager_v2`, to the wlstem context. This wlroots type manages graphics tablets (also known as drawing tablets): input devices that allow users to draw with a designated stylus. (Graphics tablets should not be confused with mobile tablets, which are usually equipped with entire

operating systems, and can be seen as complete computing devices). Along with the tablet manager, we have migrated several tablet-related functions to wlstem.

Also, in two straightforward but noteworthy changes, we have also migrated to wlstem the list of seats (moved from `sway_input_manager` to `wls_context`) and the list of transactions (moved from `sway_server` to `wls_node_manager`).

We have also migrated code related to the "idle" and "idle inhibit" features. Some compositors might wish to monitor the user session to take a certain action if user input is not detected (that is, the system is "idle") within a fixed amount of time, such as locking the screen or suspending the computer if no user input occurs in 10 minutes, for instance. Wlroots provides the type `wlr_idle` to help implementing this feature. However, it might be desirable to temporarily disable (inhibit) this feature in some situations, such as while a video is being played. To this end, sway provides the type `sway_idle_inhibit_manager_v1`. Both types were migrated to a new wlstem type, `wls_misc_protocols`, and we have added an instance of this new type into the wlstem context.

Also, as part of the effort to migrate transactions to wlstem, we have moved the type `sway_debug` to wlstem, renaming it as `wls_debug`. (However, we did not migrate transactions themselves to wlstem at this point.) The type `sway_debug` held information as to how sway should behave in several aspects that may help debugging, such as: whether to increase verbosity of transaction logging, how damage tracking should (or should not) be visually indicated, and so on. This type was related to transactions because it dictated the transaction timeout as well.

Finally, we have migrated the cursor-related function `cursor_rebase_all`, which does a "cursor rebase" (that is, warps the cursor to a sensible default location) in all output devices – and as such, some output-related code depended on it.

3.2.4 Damage tracking and `for_each` functions

Sway uses a technique called *damage tracking* [8] to detect which regions of the screen(s) actually need to be re-rendered (usually due to changes in the region's screen contents), instead of re-rendering the entire screen indiscriminately for all screens. This is implemented through several functions meant to mark a part of the screen as "damaged", meaning it needs to be re-rendered. However, there are situations in which an entire screen does need to be re-rendered: sway does this by marking the entire screen as damaged, with the function `output_damage_whole`. We have migrated this function to wlstem, along with the function `output_destroy` (which, as the name suggests, frees all resources used by an output device). Later on, we would migrate all other damage tracking functions, which were scattered throughout several files in sway, to a dedicated file `damage.c` in wlstem.

Sway also makes use of several "for each" functions, named under the general structure `FOO_for_each_BAR`, which receive a pointer to a function meant to operate on type `BAR`, and call this function for each `BAR` belonging to `FOO`. For instance, the function `output_for_each_surface` receives a function meant to operate on surfaces, and calls this function for each surface shown in a certain output. These functions were also scattered

throughout many files in sway, but we have migrated them to a dedicated file `foreach.c` in wlstem.

3.2.5 Migrating window borders back

At this point, we realized we had inadvertently introduced some window management code into wlstem – namely, code related to window borders and titles – which we then either removed or migrated *back* from wlstem to sway.

First, we have removed the `border_thickness` field (which stored how many pixels a window border should have) from both `sway_container` and `sway_container_state`: whenever those fields were being used, we used instead the `sway_config` field of same name.

Then, we have removed the boolean fields `border_top`, `border_left`, `border_right` and `border_bottom`, from both `sway_container` and `sway_container_state`. The first field was never used, and so we have removed it right away. The other three indicated whether or not a window should draw a border to its left, right or bottom, respectively; however, the fields were initially set to true and their values were never changed afterwards, so we have simply removed them by hardcoding their values as true.

We then proceeded to remove title-related fields (other than the title itself) from `sway_container`: this type had a field `formatted_title` to store the title after it had been formatted with Pango markup, and also three textures, `title_focused`, `title_unfocused` and `title_urgent`, to be used when a window was focused, unfocused, or when it had been marked urgent, respectively. We consider that the choice of Pango markup for formatting the title, as well as how many title textures and when to use them, are all window management decisions, and as such, we ought to remove this code from wlstem. Therefore, we have moved all these fields to a new type, `window_title`, and have decided to add to `sway_container` a field `data` of type `void*` to store user-defined data, which we have used to store an instance of `window_title`, which would be created as soon as the window was created, thanks to the signal `new_window` that we have introduced to notify when a window was created.

Finally, we have made a change to the function `container_discover_outputs`, which was not yet on wlstem, but which we have migrated to wlstem immediately after making this change. This function would, in some circumstances, move a container (that is, a window) to a different output, and when doing so, it checked both output's `scale` field: if there was any change between the scale of the output on which the window previously was, and the scale of the output to which the window has moved, the title textures had to be recreated. To that end, this function would then call `container_update_title_textures`; however, we did not wish this function call on wlstem, since decisions about title textures are considered window management ones. Therefore, instead of calling this function, we have made it so `container_discover_outputs` would emit a signal `scale_change` to inform library users of this event. Then, we have made it so sway would listen to this event and update title textures appropriately when it occurred.

3.2.6 Seatops, input configuration and transactions

Sway's "seatops", whose name is an abbreviated form of "seat operations", were instances of the type `sway_seatop_impl`, which contained several function pointers meant to be called at (usually input-related) events, such as button press, cursor movement, among a few others. The rationale for seatops is that the same event might need to be dealt with differently, depending on the circumstances: for instance, while a window was being resized, the cursor movement had to change the size of a window, whereas if no resize was ongoing, there would be no need for doing so. For each such circumstance where certain events had to be dealt with in a different way, sway had a different instance of `sway_seatop_impl`, with different function pointers each, along with a field of type `void*` to store relevant data for each seatop.

Originally, sway had several seat operations, but as we simplified sway, many seatops were removed, such that at this point there were only two different seatops: the default seatop, aptly named "default", and the seatop named "down", used while the cursor was being pressed down. Since we only had two seatops, we introduced a boolean field `cursor_pressed` to tell whether or not the cursor was being pressed down, and changed the implementation of all functions that previously used seatops, so that, instead of calling the function pointer stored by the seatop instance, they would check this boolean field and call one out of two possible functions. We have done this for every event to which `sway_seatop_impl` had a function pointer, and removed the corresponding function pointer from this type, until it became nearly empty: the only function pointer we have kept was "rebase", which was meant to implement the "cursor rebase" that we have mentioned in section 3.2.3.

Then, we have proceeded to migrate to wlstem the type that held configurable parameters for input devices: `input_config`, which we have moved from sway header `sway_config.h` to its own header in wlstem.

At this point, transactions were nearly ready to be migrated to wlstem. The only obstacle to this was a call to `input_manager_current_seat`, which was used to determine whether or not a container was focused in its *current* state (as opposed to its pending state; see section 1.5.2). This information would then be used to determine whether the window should be rendered with its focused or unfocused title texture (as we have mentioned in section 3.2.5).

Since `input_manager_current_seat` depended on a very large amount of functions, and even had a few dependency cycles (a situation explained in section 2.2.2), we have decided it would be easier to comment out the call to this function, and change the code to always render windows with the unfocused title texture. It should be stressed that this solution, while not ideal, have only resulted in cosmetic changes in windows' titles. Finally, after this change, we have migrated transactions themselves to wlstem.

3.2.7 Output and containers

At this point, we have analyzed the remaining output-related code in sway, to separate window management code from it and migrate the remaining code to wlstem.

We began this by migrating to wlstem the function `output_begin_destroy`, which did the necessary preparations before calling `output_destroy` (explained in section 3.2.4).

Then, we have refactored the function `handle_commit`, a handler for the `commit` output event, which occurs when a new frame is ready. This function was entirely made of window management code; however, it was used by `handle_new_output`, which we wanted to migrate to wlstem. To address this, we have moved most of `handle_commit` code to a new function, `handle_output_commit`, and changed `handle_commit` so it would simply call the new function with the appropriate arguments. Then, we have made it so `handle_output_commit` would be a callback to be received when initializing wlstem, so that `handle_commit` could be migrated to wlstem later on while not having any window management code, and while still handling the `commit` event, through the user-provided callback. (Note that we could have made it so `handle_commit` would do nothing but re-emit the signal, and let users subscribe to it; however, an user might forget to do so, and if that happened, no changes would ever happen to the contents displayed in the screen. Therefore, we have instead opted to demand a callback from the user.)

Another function used by `handle_new_output` that still had some window management code was `handle_mode`, which would call the window management functions `arrange_layers` and `arrange_output` when there was a change to the output mode (explained in section 1.4). We have made it so this function would only emit a new signal, `output_mode_changed`, and made sway listen to this signal and call these functions as necessary.

Then, after migrating some output-related helper functions to wlstem, we have proceeded to refactor `output_render` to split window management code into separate functions. Since most of the rendering code is window management dependent, we have migrated it into two separate functions: `output_render_overlay` and `output_render_non_overlay`. Afterwards, we have made it so these two functions are also callbacks passed to wlstem during its initialization, in the same way we have done with `handle_output_commit`. We then migrated `output_render` to wlstem.

Afterwards, we have migrated `output_create` to wlstem (as the name implies, this function is meant to properly allocate and initialize an output device). Then, we have changed `output_evacuate`, a function that transfers containers from an output that is going to be disabled to another one, so that the choice for the output that will receive ("absorb") the containers is done by a separate function `choose_absorber_output`. Then, we have made it so this function, too, is passed as a callback to wlstem, because we consider that choice to be a window management decision. After that, we have migrated `output_evacuate` to wlstem, along with several functions related to it.

At this point, we have finally migrated `handle_new_output` to wlstem, along with all functions it uses. We have also migrated all of the remaining output-related functions that were not involved in window management, and similarly for container-related functions. Finally, we proceeded to rename "container" as "window" in the all of wlstem code, to avoid confusion.

Chapter 4

Conclusion and future work

Wlstem has been successfully created and already implements some reusable code for Wayland compositors; however, as of this writing, wlstem still does not have all features we deem necessary for it. We have described these must-have features in section 2.4, but, to summarize, we believe wlstem must:

- Make hardware hotplugging transparent for output and input devices;
- Make clients' desktop shell and usage of XWayland transparent.

We consider that, currently, wlstem deals satisfactorily with hotplugging of output devices, but only partially so for input devices. Additionally, wlstem does not yet deal with XWayland, nor does it make desktop shells transparent. This is not due to a fault in wlstem, but rather, because we have not yet migrated enough code from sway to wlstem to implement these features.

As such, we intend to continue migrating code from sway to wlstem, while also refactoring code to make sure window management decisions are never made by our library, as we have done so far. We hope that, in doing so, we will achieve completion of the aforementioned must-have features which wlstem does not yet adequately implement.

It should also be noted that we intend to undo the changes we have done to window titles, which we have described in Section 3.2.6.

Finally, after achieving all wlstem must-have features, we shall write a small tutorial, or some similar form of documentation, so that new users who wish to create a Wayland compositor using our library will know where to start.

Appendix A

Full commit history

To give a complete view of this project's evolution, we present the full commit history, obtained from the command `git log '--pretty=format:%h %as %s'`. (Long commit messages have been manually formatted.)

Note that the `git log` command lists the most recent commits first.

```
f27a77e 2021-11-29 Move view.c out of tree/ directory
9b3ddbc 2021-11-04 Rename container as window
3732cd9 2021-11-04 Rename container.h -> window.h
e050611 2021-11-01 Merge desktop/output.c and tree/output.c into a single file
410bd06 2021-11-01 Move arrange_* function implementations to separate file
fbf4f71 2021-11-01 Move arrange_* function prototypes to separate header
84ba1d6 2021-11-01 Move prototypes of functions at window_title.c to
                   window_title.h
ddd102d 2021-11-01 Rename tree/container.c as window_title.c
049c74a 2021-11-01 Migrate several functions from tree/container.c to wlstem
c2fc911 2021-11-01 Move container_begin_destroy() from tree/container.c to
                   container.c
a652eb0 2021-10-31 Migrate handle_output_layout_change() to wlstem
c8468fb 2021-10-31 handle_output_layout_change(): Remove unused sway_server
                   pointer
673411b 2021-10-31 Migrate useful output-related functions to wlstem
ae855f0 2021-10-31 Rename output's field ‘tiling‘ as ‘windows‘
f290af3 2021-10-31 Move new_output listener from sway_server to
                   wls_output_manager
1bb697d 2021-10-31 Migrate handle_new_output and the handlers it uses to wlstem
8b6a626 2021-10-31 Migrate output_disable() to wlstem
1b70538 2021-10-31 Turn choose_absorber_output into a wlstem callback
75031f0 2021-10-31 output_evacuate(): move choice of fallback output to new
                   function
8642754 2021-10-31 Migrate output_create() to wlstem
836571c 2021-10-31 handle_new_output(): Remove unused pointer to sway_server
42846a3 2021-10-31 Migrate output_render to wlstem
```

```
893decd 2021-10-31 Pass output_render_{overlay,non_overlay} as callbacks in
                   wls_init
29566d8 2021-10-31 output_render(): Refactor most actual rendering into seperate
                   functions
7bda730 2021-10-30 Rename 'render_output' with a more descriptive name
b36b7fc 2021-10-30 Move render_rect and premultiply_alpha to wlstem
6699520 2021-10-30 Move scissor_output() and render_texture() to wlstem
340b678 2021-10-29 Migrate surfaces to wlstem
0839325 2021-10-29 Make new signal/listener pair for output mode change
996e46f 2021-10-29 Rename tree/arrange.c as wm.c
ffcb187 2021-10-29 Remove incorrect comment
d844599 2021-10-29 Turn handle_output_commit into a user-supplied callback
5424c1c 2021-10-29 Rename sway_arrange.h -> server_wm.h
f5e0063 2021-10-29 desktop/output.c: Make handle_commit a wrapper for
                   handle_output_commit
c46c72f 2021-10-29 Migrate output_begin_destroy() to wlstem
016cf6c 2021-10-29 sway_output: Remove unused pointer to sway_server
d90c099 2021-10-28 Migrate transactions to wlstem
bf2f7d8 2021-10-28 HACK: comment out input_manager_current_seat call
e0d9775 2021-10-28 Move input_config definition to new header in wlstem
4480e7e 2021-10-27 Move wls_output_layout_for_each_output() to foreach.c
fe63056 2021-10-27 Remove code from handle_start_drag / handle_request_start_drag
e3c96f6 2021-10-27 Rename tiling_container_at and container_at_linear
571a968 2021-10-27 Move 'surface_at_view' to new file at.c
e03421a 2021-10-27 Move ‘current_seat‘ from sway_config to wlstem context
ecd4b6f 2021-10-27 Remove code to render seatop
377f778 2021-10-27 Remove ‘unref‘ function pointer from seatop_impl
c4c434f 2021-10-27 Remove ‘tablet_tool_tip‘ function pointer from seatop_impl
c3fdbea 2021-10-27 Remove ‘tablet_tool_motion‘ function pointer from seatop_impl
a5be233 2021-10-27 Remove ‘pointer_axis‘ function pointer from seatop_impl
53d2cca 2021-10-27 Remove ‘pointer_motion‘ function pointer from seatop_impl
7088918 2021-10-27 Remove ‘button‘ function pointer from seatop_impl
0be83a3 2021-10-27 Remove ‘end‘ function pointer from seatop_impl
cd11e37 2021-10-26 Make a boolean field to tell which seatop "mode" we're in
e32f6cc 2021-10-26 seatop_allows_set_cursor(): Hardcode return value
02aeebc 2021-10-26 seatop_tablet_tool_motion(): Remove dead code
8b575cf 2021-10-20 Migrate several view and container related functions to wlstem
76f034c 2021-10-20 Don't update title textures directly at
                   container_discover_outputs
785306f 2021-10-20 Move (most) title data away from sway_container
79f2e06 2021-10-20 Create new_window signal
5164611 2021-10-19 Hardcode containers' border_left/right/bottom fields as true
bb4678d 2021-10-19 Remove unused ‘border_top‘ field from container and
                   container_state
8c58e00 2021-10-19 Remove border thickness info from container and
                   container_state
```

```
7a7837b 2021-10-17 container.c: Emit destroy signal when container is destroyed
ec49d51 2021-10-17 view_autoconfigure: Remove dead code
89ad063 2021-10-16 Migrate damage functions to damage.{c,h} in wlstem
81b4188 2021-10-16 Move functions *_for_each_* to foreach.{c,h}
03cf1bf 2021-10-16 Move sway_layers header to wlstem
5567539 2021-10-16 Migrate sway_view's header to wlstem
0bd42b9 2021-10-16 Migrate output_damage_whole and output_destroy to wlstem
8faf6a0 2021-10-16 Migrate cursor_rebase_all() to wlstem
17a9e01 2021-10-16 Migrate transaction_timeout_ms to wls_debug
fec36df 2021-10-16 Give 'txn_timeout_ms' the clearer name
                   'transaction_timeout_ms'
44496e9 2021-10-16 Migrate sway_debug to wlstem context, as wls_debug
2b17976 2021-10-16 Migrate idle_inhibit_v1 and some view/seat functions
6012efe 2021-10-15 Don't include "sway_server.h" in sway_idle_inhibit_v1.h
7cbbf87 2021-10-14 Migrate transactions list to wlstem's node_manager
6b0f2f3 2021-10-13 Migrate seats list to wlstem context
cbe74ab 2021-10-13 Migrate most of tablet-related functions to wlstem
e061e31 2021-10-13 Migrate cursor header to wlstem
9f0b91b 2021-10-13 Migrate tablet_v2 field from sway_server to wlstem context
1c2e824 2021-10-13 Migrate text input / input methods to wlstem
2ccb104 2021-10-10 Move headers to wlstem: seat, input_manager, tablet,
                   text_input
391f952 2021-10-10 Don't include sway_config.h in sway_input-manager.h
62c22d3 2021-10-10 Migrate bool_option to wlstem (util.h)
9ea4194 2021-10-10 sway_seat.h: Remove unneeded #include for sway_config.h
6b48fa7 2021-10-10 Don't include sway_server.h at sway_input-manager.h
5374a01 2021-10-05 input_manager_create(): Receive wls_server instead of
                   sway_server
e22d9e1 2021-10-03 Unify enums for seat_config default/enabled/disabled options
ee54b33 2021-10-02 Rename wls_server.h as server.h
d3be6dc 2021-10-02 Rename headers sway/<foo>.h as sway_<foo>.h
912de44 2021-10-02 Remove unused header pool-buffer.h
bc7f1ad 2021-10-02 Flatten sway's include directory tree
625ba32 2021-08-30 Rename sway_root -> wls_output_manager
2ad48bb 2021-08-30 Migrate output_manager from sway_server to wlstem
f1d8fe1 2021-08-30 Migrate output_config to wlstem
503e4ce 2021-08-30 output_config: Remove unused functions
a15eb15 2021-08-30 Temporarily move some calls away from output_enable()
8433921 2021-08-30 Set seat focus at output_connected handler instead of
                   output_enable
8478039 2021-08-30 Move seize_containers_from_noop_output call to output_connect
                   handler
76cb698 2021-08-29 config/ouptut.c: Refuse to disable an enabled output
2c2143d 2021-08-29 Use signals instead of calling arrange_root at
                   output_(enable,disable)
3436311 2021-08-19 Migrate output_configs list to wlstem
```

```
544ed43 2021-08-19 output, output_config: Remove calls to
                   input_manager_config_all...
3f04e8e 2021-08-19 Create type server_wm and wire it to output_layout_changed
b49cfd6 2021-08-18 Make wls_server and migrate some of sway_server to it
0e66a30 2021-08-16 Migrate sway_root to wlstem
b757453 2021-08-16 Temporarily move root_for_each_container from output.c to
                   root.c
d063573 2021-08-16 root.c: Remove unused #includes
a6b2aa3 2021-08-15 Unify callbacks for output_layout's change event
610054b 2021-08-14 Move new_node events from sway_root to wls_node_manager
8c66644 2021-08-14 Rename sway_node -> wls_transaction_node
0133071 2021-08-13 Migrate sway_node to wlstem
5089775 2021-08-13 Migrate dirty_nodes to separate type and create wls_context
0e43ada 2021-08-13 Implicitly declare sway_view on output.h
9c9dc05 2021-08-13 Move common/ files to wlstem/
0a7b38a 2021-08-13 Remove pool-buffer example client
a51c409 2021-08-13 wlstem: Add to wlstem the same dependencies sway has
eb3eed0 2021-08-13 Branch out output_config code from sway_config.h to a new
                   header
83adf67 2021-08-13 Rename sway/config.h as sway/sway_config.h
7c54eb5 2021-08-11 Move input_config_tool from sway/config.h to input/tablet.h
ac4a8c6 2021-08-11 tree/container.c: Remove unused #includes
2a89c1b 2021-08-11 Move sway_container code that relies on sway_seat to separate
                   file
29dafab 2021-08-09 Remove workspaces
a1ff262 2021-08-09 Remove unused #includes to "sway/tree/workspace.h"
cf69bf2 2021-08-09 Replace ‘active_workspace‘ with boolean ‘active‘ field
e1a9e51 2021-08-09 output.c: Inline workspace_detach() at output_evacuate()
0028e86 2021-08-09 Hardcode workspace_is_visible() calls as ‘true‘
8cecbad 2021-08-09 Hardcode config's focus on activation field behavior
5febea6 2021-08-09 Remove sway_workspace_state
f280a83 2021-08-09 render_workspace,render_output: Hardcode focused field in
                   parent data
b2a0e5d 2021-08-09 transaction.c: Remove unused workspace_state
47bbf8f 2021-08-09 Remove unused function output_for_each_workspace
bd3a87a 2021-08-09 output_seize_containers_...: Receive sway_output directly
5025df3 2021-08-09 Replace workspace_for_each_container with
                   output_for_each_container
fbfd6c1 2021-08-09 Inline workspace_destroy
04e7e9f 2021-08-09 Remove unused function root_for_each_workspace
e2eda29 2021-08-09 Replace workspace_is_empty with output_has_containers
6092c77 2021-08-09 Replace workspace_add_tiling with output_add_container
ec77fda 2021-08-08 Remove output node from all focus stacks when it's
                   disconnected
ff20ec5 2021-08-04 node_init: Check for invalid node types
1d68fc1 2021-07-18 seat_set_raw_focus(): Dump focus stack in log file
```

```
ea08d66 2021-07-17 select_output: Return only active outputs
c51b26b 2021-07-16 Don’t use workspaces’ nodes
ef92561 2021-07-06 container_at: Receive output instead of workspace
77fd057 2021-07-06 Manually inline output_get_active_workspace()
f8d79ae 2021-07-06 Simplify output_get_active_workspace
8736426 2021-07-04 Remove root node
4b2bd0c 2021-07-04 Make output node’s parent be itself
a6eb9c4 2021-07-04 Migrate list.{c,h} to wlstem
e7c5a08 2021-06-30 Move wlstem headers to separate directory
5baa717 2021-06-23 README.md: Fix line breaks
d1b5890 2021-06-23 README.md: Fix line breaks and capitalization
cc92997 2021-06-23 Add README.md
1b7e4a7 2021-06-14 Create new library wlstem from exec.c and log.c
b7f26d9 2021-06-14 exec.c: Remove unnecessary #include’s
9c30620 2021-04-11 Replace calls to seat_get_focus_inactive() using root node
0507796 2021-04-11 Remove unused function arrange_node()
2c099e0 2021-03-21 Create function root_for_each_output()
e28c21a 2021-03-21 Rename node_is_container_or_ws() to a more meaningful name
d500e5c 2021-03-20 Make container hold a reference to an output instead of a
                   workspace
1418ca5 2021-03-20 Refactor (almost inline) function output_add_workspace()
73b09d1 2021-03-15 output_evacuate(): call workspace_detach() after seizing
                   containers
1c3145a 2021-03-15 Check for NULL before accessing workspace’s output
54e1f2e 2021-03-15 Transfer container list (‘tiling‘) from workspace to output
a6dba6e 2021-03-14 seat.{c,h}: Remove seat’s workspace field
a296125 2021-03-14 Replace seat_get_focused_workspace() with
                   seat_get_focused_output()
6077959 2021-03-14 seat.{c,h}: Inline seat_get_last_known_workspace()
cb29888 2021-03-07 Remove x,y,width,height fields from workspace
f42cc16 2021-03-07 Inline remaining arrange_workspace() call
34ac5c9 2021-03-07 Replace arrange_workspace() calls with arrange_output() calls
80087ab 2021-03-07 Replace workspace_get_box() with output_get_render_box()
4bd01ba 2021-03-07 Store workspace’s (x,y) coords in output
9f822d6 2021-03-07 Inline render_containers()
fe35ab7 2021-03-07 Remove unused function opposite_direction()
3280bb5 2021-03-07 Don’t detach or destroy workspace at
                   output_seize_containers_...
d03a471 2021-03-07 output.c: Don’t detach noop_output’s workspace
4094c3f 2021-03-07 node.{c,h}: Remove unused functions
4efad34 2021-03-07 output.c: Don’t destroy empty workspaces
67671b7 2021-03-07 output.c: Refuse to add workspaces tied to a different output
9fe5e2b 2021-02-28 Remove workspace_consider_destroy() and its calls
13b9574 2021-02-27 Remove *_find_* functions
b1f2aac 2021-02-27 Remove workspace name
645d90b 2021-02-27 seat.{c,h}: Remove prev_workspace_name
```

```
43ec10c 2021-02-27 Refuse to create workspace for NULL output
154aad5 2021-02-27 workspace.c: Remove dead if statement
83e2ef5 2021-02-27 Remove empty function workspace_name_from_binding() and its
                   calls
61900c2 2021-02-27 Rename restore_workspaces() as
                   seize_containers_from_noop_output()
129945e 2021-02-27 Remove empty function output_sort_workspaces() and its calls
9510f8f 2021-02-27 Change output's workspace list for a single workspace pointer
d9842f7 2021-02-27 output.c: Enforce one workspace per output
f02cde7 2021-02-27 Remove useless loop at restore_workspaces()
47af4a5 2021-02-27 Remove unused sway_container fields
941ff84 2021-02-27 apply_horiz_layout(): use same width fraction for all children
7915f3f 2021-02-27 output.c: Transfer containers instead of whole workspace
8e0437d 2021-02-26 container.h: Remove unimplemented function prototype
8c04e72 2021-02-24 Remove containers' parent
ebc725c 2021-02-24 container.{c,h}: Remove unused functions
f0eae06 2021-02-24 Remove empty function container_for_each_child() and related
                   code
f2d2d62 2021-02-24 Remove containers' children
cd355de 2021-02-24 Remove container_reap_empty() and its calls
c1da9bd 2021-02-24 Remove functions related to container representation
0a3f999 2021-02-24 Remove container_find_child() and its calls
1fff819 2021-02-24 node_get_children(): Don't use containers' children
d878647 2021-02-24 Always check for NULL at node_get_children()'s return value
fdb1023 2021-02-24 container.{c,h}: Remove unused functions
77feb67 2021-02-18 workspace.{c,h}: Remove unused functions
3bf120a 2021-02-16 workspace.{c,h}: Remove output_priority
28b59ca 2021-02-16 Remove workspace_output_add_priority()
61ff7af 2021-02-16 Remove workspace_output_get_highest_available()
8bd11bd 2021-02-16 Make workspace_output_get_highest_available() always return
                   NULL
e1bbe07 2021-02-15 Remove unused function workspace_output_raise_priority()
0157137 2021-02-15 Remove unused workspace_by_name() and related functions
a7e4d29 2021-02-15 Hardcode workspace_name_from_binding() as no-op
6b93054 2021-02-15 Remove workspaces list from sway_output_state
7dad11b 2021-02-15 Remove focused_inactive title textures and border colors
f4992de 2021-02-15 Remove unused function seat_get_focus_inactive_tiling()
99bb665 2021-02-15 Remove focused_inactive_child from workspaces and containers
507a321 2021-02-14 Remove unused transient containers / popup_during_fullscreen
b21c677 2021-02-14 Remove fullscreen views
f549e75 2021-02-14 Remove workspace's urgent property
f08e162 2021-02-14 Remove workspace's representation
da9e29c 2021-02-14 Remove fullscreen_global container
058723f 2021-02-11 Remove PID -> workspace mappings
3f456ee 2021-02-10 config.{c,h}: Remove workspace_config type and related
                   functions
```

```
44e70b0 2021-02-10 config.{c,h}: Remove unused function
                   workspace_output_cmp_workspace()
90b5a58 2021-02-10 workspace.h: Remove unused function workspace_find_config()
9603700 2021-02-10 workspace_valid_on_output(): Remove dead code
3f00ad0 2021-02-10 render.c: Remove render_floating()
793e719 2021-02-10 config: Remove floating container-related fields
99f60f7 2021-02-10 seat.{c,h}: Remove seat_get_focus_inactive_floating()
ba4fa7e 2021-02-10 Remove floating-related code from views
39bb25f 2021-02-10 Remove floating containers list from workspaces
a87550e 2021-02-10 Remove floating containers
285e015 2021-02-10 Remove scratchpad
2028f5a 2021-02-10 Remove sticky containers
9b5d607 2021-02-10 Remove sway_container_layout type
972aa67 2021-02-10 Remove unused function output_get_default_layout()
edcc86b 2021-02-10 Remove layout fields from workspaces
7a05490 2021-02-10 arrange_children(): remove unneeded layout parameter
e4a0d75 2021-02-10 No longer change cursor to resizing cursor icon
81a7c17 2021-02-10 container_build_representation(): Remove unused parameter
5636c99 2021-02-10 container.c: Remove check for container layout
2e5dae8 2021-02-10 Remove comment about container layout which no longer applies
2be6fde 2021-02-10 Remove tiling resize
f5b1339 2021-02-10 Remove container_current_parent_layout() and its calls
b9653c1 2021-02-10 seatop_resize_tiling.c: Remove vertical splits
4107cc5 2021-02-10 Remove squash functions
5749983 2021-02-10 Refactor most switch..case's testing for container layout
5928c7d 2021-02-08 Replace node_get_layout() with node_is_container_or_ws()
00563f8 2021-02-08 Remove vertical, tabbed and stacked layout
47694e9 2021-02-06 Make all borders behave as if they were B_NORMAL
7aba56f 2021-02-06 Hardcode config's border and floating_border as B_NORMAL
14dd415 2021-02-06 config: Remove unused field 'focus_wrapping'
b4ffc23 2021-02-06 config: remove modes list
bad9d1b 2021-02-06 Remove binding flags
2a0cde3 2021-02-06 Remove mouse and switch bindings
2b7d4ad 2021-02-06 Hardcode config's default_orientation and default_layout as
                   L_NONE
8d40fca 2021-02-06 config: Remove unused fields dragging_key and resizing_key
d541cff 2021-02-06 Disable floating modifier
773e122 2021-02-06 Disable moving/resizing floating views
ebaaf0d 2021-02-06 Remove container marks
20f93d8 2021-02-06 Hardcode config->show_marks as true
d3ebb45 2021-02-06 Disable tiling drag feature
35d0b91 2021-02-06 Remove gaps
6be609d 2021-02-06 Remove ability to hide edge borders
7e41185 2021-02-06 config: remove hide_lone_tab
96c9263 2021-02-06 Hardcode config->hide_lone_tab as false
8bf4615 2021-02-06 config: remove smart_gaps field
```

```
3f869e2 2021-02-06 Hardcode config->smart_gaps as false
30133d1 2021-02-06 Remove float_maximum* config fields
c9531d2 2021-02-06 Replace config's handler_context with current_seat
764304c 2021-02-02 Remove workspace_configs and associated functions
657d734 2021-02-02 config: Remove current_config* fields
a150300 2021-02-02 Remove seat's deferred_bindings and run_deferred_bindings()
ca36414 2021-02-02 Remove empty function `run_deferred_commands`
a7eb608 2021-02-02 config.h: Remove unused type `sway_variable`
66d2443 2021-01-31 Change termination keybinding to Ctrl+Alt+Backspace
27ae9d9 2021-01-30 Move files away from commands folder
729dc43 2021-01-30 Remove commands.c
b5706a5 2021-01-30 Receive keys directly in bind command
262f6a2 2021-01-30 Receive key modifiers directly in bind command
6195666 2021-01-30 bind.c: Remove switch binding commands
ee73b92 2021-01-30 bind.c: Remove group parsing from cmd_bindsym_or_bindcode()
e0c9484 2021-01-30 commands.h: Remove unused types
6f1c7bf 2021-01-29 commands.{c,h}: Remove unused functions
873f591 2021-01-29 Log command's errors instead of returning them
a67ea89 2021-01-29 bind.c: Remove flags from bind commands
29fa49c 2021-01-29 workspace.c: use mock command at workspace_name_from_binding()
8a38993 2021-01-29 switch.c: Use binding callback instead of command
77b7311 2021-01-29 Rework terminal spawning with new bind command
e6fd527 2021-01-29 bind.c: use callbacks instead of command strings
48f7b9c 2021-01-29 Replace command string with callback at bindings
ca4258f 2021-01-28 Refactor exec into a separate function
3d1d625 2021-01-28 config.c: Move keybinding insertion to separate function
dcb7cc4 2021-01-28 sway: Remove JSON dependency
c620aac 2021-01-28 Remove cmd_results_to_json()
1f2f2cf 2021-01-28 Remove command `swap`
2557b2f 2021-01-28 Remove container_swap() declaration and calls
4ccda96 2021-01-28 Remove `resize` command
f4aacaf 2021-01-28 meson.build: Remove swaybg-related option gdk_pixbuf
d43c857 2021-01-28 Remove file background-image
3292dfd 2021-01-28 meson_options: Remove swaybg-related option
4b14ef2 2021-01-28 Remove i3's IPC protocol implementation
8f6e9c3 2021-01-28 sway: stop sending i3 protocol's IPC events
00743a3 2021-01-28 Remove unused `swaybg`-related fields
1706735 2021-01-28 config: Remove unused flag `failed`
6260ada 2021-01-28 config: Remove unused command queue
baac009 2021-01-28 config: Remove unused floating_scroll_* fields
b7dd45d 2021-01-27 Remove criteria and all code using it
c56ec65 2021-01-27 Remove CT_COMMAND criteria
ae417b2 2021-01-27 Remove ASSIGN_OUTPUT criteria
7a75c1d 2021-01-27 Remove ASSIGN_WORKSPACE criteria
851143d 2021-01-27 Remove command `workspace`
6b131ff 2021-01-27 Move free_workspace_config() to config.c
```

```
5bdacde 2021-01-27 Remove ASSIGN_WORKSPACE_NUMBER criteria
a5b1066 2021-01-27 Remove no_focus criteria
57c14d9 2021-01-27 commands.h: Remove unused commands
50356bf 2021-01-27 Remove IPC client code
e0b6d57 2021-01-27 main.c: Don’t run as IPC client
58b7e6a 2021-01-27 Remove variable list from config
c3f9cd0 2021-01-27 Remove command ‘set‘ and free_sway_variable()
6e14235 2021-01-27 Remove do_var_replacement() and its calls
651ff32 2021-01-26 Remove ‘seat‘ command and its remaining subcmds
10e310d 2021-01-26 Remove input command and its remaining subcmds
b45a460 2021-01-26 Remove input command ‘tool_mode‘
1ffeec8 2021-01-26 Remove input commands map to/from region
d64fbc6 2021-01-26 Remove input command ‘events‘
caceb57 2021-01-26 Remove ‘output‘ command and its subcommands
36b7553 2021-01-26 Remove command ‘layout‘
f5d2735 2021-01-26 Remove command ‘kill‘
a598b76 2021-01-26 Remove command ‘for_window‘
2374a71 2021-01-26 Remove command ‘focus‘
65d6f6b 2021-01-26 Remove command ‘mode‘
9e1e207 2021-01-26 Remove command ‘move‘
d68d44d 2021-01-26 Remove command ‘scratchpad‘
7963e66 2021-01-26 Remove command ‘inhibit_idle‘
f0337cf 2021-01-26 Remove command ‘rename‘
0d80360 2021-01-26 Remove split commands
3eb5ce3 2021-01-26 Remove command ‘sticky‘
e6e00e3 2021-01-26 Remove command ‘gaps‘
cc8c6cc 2021-01-26 Remove command ‘nop‘
ad1546c 2021-01-26 Remove client commands
1ae7d70 2021-01-26 Remove command ‘no_focus‘
f730ec4 2021-01-26 Remove command ‘shortcuts_inhibitor‘
09141f0 2021-01-26 Remove command ‘fullscreen‘
14b622f 2021-01-24 Remove option-setting commands
e0a271c 2021-01-24 Remove commands related to floating windows
0d723c2 2021-01-24 Remove command ‘floating‘
e7ee694 2021-01-24 Remove command ‘exit‘
0a5fbfb 2021-01-24 Remove command ‘create_output‘
30b461c 2021-01-24 Remove command ‘assign‘
d7b2e0e 2021-01-24 Remove command ‘urgent‘
066fa55 2021-01-24 Remove border-related commands
4bf4f02 2021-01-23 Remove titlebar-related commands
89afb2a 2021-01-23 Remove mark-related commands
96ac549 2021-01-23 Remove command focus_follow_mouse
282d678 2021-01-23 Remove config_handlers (config-time commands)
0c0565c 2021-01-23 Remove command ‘xwayland‘
d68744e 2021-01-23 Remove command workspace_layout
f72f93f 2021-01-23 Remove command default_orientation
```

```
4e2747c 2021-01-23 Remove command swaybg_command
86f38fd 2021-01-23 Add shortcut to invoke terminal (Alt+Enter)
7aefca2 2021-01-23 Remove ws_auto_back_and_forth functionality
c8bdf6e 2021-01-23 config: Remove ‘include‘ command
ac7c1ad 2021-01-23 config: Remove unused functions
6bdb25d 2021-01-23 Don’t access removed field ‘config->reading‘
4a6f89a 2021-01-23 load_main_config(): load default config only
b86855a 2021-01-23 sway: Remove "reload config" feature
3011301 2021-01-23 config.c: Remove commented-out code
0af8656 2021-01-23 Remove custom config file name
6a24d15 2021-01-23 sway: Remove config file validation
4121670 2021-01-23 Remove sway dir to ‘mod_sway‘ to avoid confusion
0da45a0 2021-01-22 meson_options.txt: Remove unused options
6636a7d 2021-01-22 Remove system tray-related code
ecf47fa 2021-01-22 main.c: no longer load swaybar
a90516f 2021-01-22 reload.c: no longer reload swaybar
b7e8f67 2021-01-22 Remove code for swaybar visibility
cf1f284 2021-01-22 IPC: Remove swaybar-related functionality
4d9d3b6 2021-01-22 Remove swaybar-related config code
8a1c256 2021-01-22 Remove swaybar-related commands
baeb245 2021-01-22 meson.build: Remove subproject swaybar
ccb97a7 2021-01-22 Remove swaybar-related files
5b44d4b 2021-01-20 Remove swaymsg
f9cbbfb 2021-01-20 sway: Add hardcoded keybinding to exit
9ce2714 2021-01-19 Add tags file to .gitignore
6c3eaae 2021-01-19 sway: Stop using swaynag
9bad715 2021-01-19 Remove swaynag-related files
aa98530 2021-01-19 Remove CONTRIBUTING.md
4ca9c56 2021-01-18 Remove example configuration file
c330095 2021-01-18 Remove manpages
97ea8cf 2021-01-18 Add myself to LICENSE
dfd8add 2021-01-18 Remove auxiliary sway programs and scripts
ff25aca 2021-01-18 Remove desktop entry
ac118d5 2021-01-18 Remove shell completions
5ed3cf8 2021-01-18 Remove assets
27c4892 2021-01-18 Remove READMEs
e9aba14 2021-01-18 Fork sway
```

References

[1] *Arch Linux wiki page on Xorg.* "Xorg (commonly referred to as simply X) is the most popular display server among Linux users." URL: https://wiki.archlinux.org/index.php?title=Xorg&oldid=692772 (visited on 09/03/2021) (cit. on p. 3).

[2] *Arch Linux Wiki: (Non-exhaustive) List of window managers.* URL: https://wiki.archlinux.org/index.php?title=Window_manager&oldid=691882#List_of_window_managers (visited on 09/03/2021) (cit. on pp. 4, 5).

[3] *Arch Linux Wiki: List of Wayland compositors.* URL: https://wiki.archlinux.org/index.php?title=Wayland&oldid=690868#Compositors (visited on 09/05/2021) (cit. on p. 5).

[4] *Drew DeVault's blog: The future of Wayland, and sway's role in it (Archived).* "However, wlc has some limitations, ones that sway has been hitting more and more often [...] To address these limitations, we've been working very hard on a replacement for wlc called wlroots." URL: http://web.archive.org/web/20180606181450/https://drewdevault.com/2017/10/09/Future-of-sway.html (visited on 09/03/2021) (cit. on p. 5).

[5] *GLFW FAQ, Section 1.4 - What platforms are supported by GLFW?* "Support for Wayland is available but not yet feature complete." URL: https://www.glfw.org/faq.html#14---what-platforms-are-supported-by-glfw (visited on 09/11/2021) (cit. on p. 5).

[6] *GTK+ project's main page.* URL: https://www.gtk.org/ (visited on 09/11/2021) (cit. on p. 3).

[7] *i3's user guide (archived).* URL: http://web.archive.org/web/20210822214659/https://i3wm.org/docs/userguide.html (visited on 09/12/2021) (cit. on pp. 8, 9, 20).

[8] *Introduction to damage tracking (archived).* URL: https://web.archive.org/web/20210312135541/https://emersion.fr/blog/2019/intro-to-damage-tracking/ (visited on 12/19/2021) (cit. on p. 27).

[9] *libweston documentation.* URL: https://wayland.pages.freedesktop.org/weston/toc/libweston.html (visited on 09/03/2021) (cit. on p. 5).

[10] *LINFO: The X Window System: A Brief Introduction.* "X was born in May 1984 [...] X11 was released in September 1987." The Linux Information Project. URL: http://www.linfo.org/x.html (visited on 09/05/2021) (cit. on p. 4).

[11] *phoronix: SDL 2.0.16 Released With Better Wayland Support, PipeWire Integration.* "SDL 2.0.16 is a big update with much better native Wayland support". URL: https://www.phoronix.com/scan.php?page=news_item&px=SDL-2.0.16-Released (visited on 09/05/2021) (cit. on p. 5).

[12] *Qt project's main page.* URL: https://www.qt.io/ (visited on 09/11/2021) (cit. on p. 3).

[13] *QtWayland main wiki page*. URL: https://wiki.qt.io/QtWayland (visited on 09/03/2021) (cit. on p. 4).

[14] *StackOverflow: What is the reason for performing a double fork when creating a daemon? (archived)*. URL: https://web.archive.org/web/20211009010142/https://stackoverflow.com/questions/881388/what-is-the-reason-for-performing-a-double-fork-when-creating-a-daemon (visited on 12/22/2021) (cit. on p. 21).

[15] *sway project's README*. URL: https://github.com/swaywm/sway/blob/b345d6e5d2f175b0be5c5f2a445291611104c98e/README.md (visited on 09/03/2021) (cit. on p. 5).

[16] *TLDP: window managers*. The Linux Documentation Project. URL: https://tldp.org/HOWTO/XWindow-Overview-HOWTO/window-managers.html (visited on 09/03/2021) (cit. on pp. 3, 4).

[17] *TLDP: X architecture overview*. The Linux Documentation Project. URL: https://tldp.org/HOWTO/XWindow-Overview-HOWTO/arch-overview.html (visited on 09/03/2021) (cit. on p. 3).

[18] *Wayland Book: Design Patterns: Atomicity (archived)*. URL: http://web.archive.org/web/20210909192422/https://wayland-book.com/protocol-design/design-patterns.html#atomicity (visited on 09/13/2021) (cit. on p. 8).

[19] *Wayland Book: Globals & the registry (archived)*. URL: https://web.archive.org/web/20210310230817/https://wayland-book.com/registry.html (visited on 12/23/2021) (cit. on p. 13).

[20] *Wayland Book: Interfaces, requests, and events (archived)*. URL: https://web.archive.org/web/20210310230506/https://wayland-book.com/protocol-design/interfaces-reqs-events.html (visited on 12/23/2021) (cit. on p. 13).

[21] *Wayland Book: Seats: Handling input (archived)*. URL: https://web.archive.org/web/20210310230211/https://wayland-book.com/seat.html (visited on 12/20/2021) (cit. on p. 12).

[22] *Wayland Book: The high-level protocol (archived)*. URL: https://web.archive.org/web/20210310230434/https://wayland-book.com/protocol-design/high-level.html (visited on 12/23/2021) (cit. on p. 13).

[23] *Wayland Book: Wire protocol basics (archived)*. URL: https://web.archive.org/web/20210310230340/https://wayland-book.com/protocol-design/wire-protocol.html (visited on 12/23/2021) (cit. on p. 13).

[24] *Wayland FAQ: How can I replace Wayland's Window Manager?* "The Wayland architecture integrates the display server, window manager and compositor into one process." URL: https://wayland.freedesktop.org/faq.html#heading_toc_j_11 (visited on 09/03/2021) (cit. on p. 4).

[25] *Wayland FAQ: What is wrong with X?* Free Desktop. URL: https://wayland.freedesktop.org/faq.html#heading_toc_j_6 (visited on 09/03/2021) (cit. on p. 4).

[26] *Wayland home page*. "Wayland is intended as a simpler replacement for X, easier to develop and maintain. [...] Wayland is a protocol for a compositor to talk to its clients as well as a C library implementation of that protocol." Free Desktop. URL: https://wayland.freedesktop.org (visited on 09/05/2021) (cit. on p. 4).

[27] *Wayland support status in GTK+*. URL: https://wiki.gnome.org/Initiatives/Wayland/GTK%2B (visited on 09/03/2021) (cit. on p. 4).

[28] *Wayland: Appendix A. Wayland Protocol Specification (archived)*. URL: https://web.archive.org/web/20210310230348/https://wayland.freedesktop.org/docs/html/apa.html (visited on 12/20/2021) (cit. on p. 13).

[29] *Weston documentation home page*. "Weston is the reference implementation of a Wayland compositor, as well as a useful environment in and of itself." Free Desktop. URL: https://wayland.pages.freedesktop.org/weston/index.html (visited on 09/03/2021) (cit. on p. 5).

[30] *winit scope: FEATURES.md*. "It supports the main graphical platforms: [...] Unix [...] via Wayland". URL: https://github.com/rust-windowing/winit/blob/657b4fd59eb8d4f89eef8388e3346e7901ed1fa6/FEATURES.md (visited on 09/11/2021) (cit. on p. 5).

[31] *wlc project's README*. "wlc is officially deprecated. Interested users are encouraged to use wlroots instead." URL: https://github.com/Cloudef/wlc/blob/6e2c49bf5789115bf33873f3e0ebe603c8e7725d/README.rst (visited on 09/06/2021) (cit. on p. 5).

[32] *wlroots project's README*. URL: https://github.com/swaywm/wlroots/blob/c8d97e27916e1a1df6d103fa957a56829af6add3/README.md (visited on 09/03/2021) (cit. on p. 5).

[33] *X Window System manual*. The Open Group. URL: https://www.x.org/releases/X11R7.7/doc/man/man7/X.7.xhtml (visited on 09/03/2021) (cit. on p. 3).

[34] *X Window System standards manual*. The Open Group. URL: https://www.x.org/releases/X11R7.7/doc/man/man7/Standards.7.xhtml (visited on 09/03/2021) (cit. on p. 3).

[35] *XCB project page (Archived)*. "The X protocol C-language Binding (XCB) is a replacement for Xlib featuring a small footprint, latency hiding, direct access to the protocol, improved threading support, and extensibility." Free Desktop. URL: http://web.archive.org/web/20210420003248/https://xcb.freedesktop.org/ (visited on 09/11/2021) (cit. on p. 3).

[36] *Xorg home page*. The Open Group. URL: https://www.x.org/wiki (visited on 09/03/2021) (cit. on p. 3).