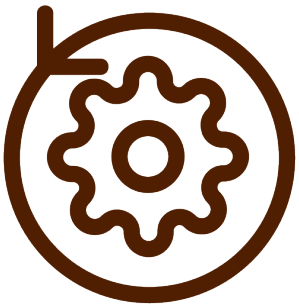
ReSet

Authors: Fabio Lenherr / Felix Tran

Project Advisor: Prof. Dr. Frieder Loch



School of Computer Science

OST Eastern Switzerland University of Applied Sciences

Abstract

In this paper, ReSet, a user-friendly settings application for Linux is developed with the Rust programming language. ReSet aims to run on every distribution and environment, providing a consistent experience for all users.

Key features include the configuration of Wi-Fi networks, Bluetooth devices, and audio devices. These features are tied together in a responsive user interface.

Future development will focus on new features such as configuring peripherals, as well as a plugin system. This will allow users to extend ReSet with their features.

Existing solutions either focus only on one functionality or are tightly coupled into an already existing environment, which brings incompatible features to other environments with it. Users of smaller, niche environments are therefore forced to either use multiple solutions at once or accept a partially incompatible solution. ReSet focuses on more than one functionality while not bundling with any environment, making it an independent application and solving the underlying issue.

In order to provide a consistent, functional and appealing design, multiple existing solutions were compared and analyzed in order to develop ReSet.

In summary, ReSet represents a significant advancement in Linux settings applications, offering a concise, cross-environment solution that prioritizes user experience through Rust's efficiency and adaptability.

Acknowledgments

We would like to express our sincerest gratitude to Prof. Dr. Frieder Loch for advising us throughout our project. His guidance and support have been invaluable to us. We are also grateful to our testers who took the time to test our product and share valuable feedback. Their input has been instrumental in helping us make improvements to our product. We would also like to acknowledge the OST for providing us with the opportunity to work on this project.

Table of Contents

1 Introduction

The Linux ecosystem is well known to be fractured, whether it is the seemingly endless amount of distributions or the various desktop environments, there will always be someone who will create something new. With this reality comes a challenge to create software that is not dependent on one singular distribution or environment.

The same lack of universality can be seen when interacting with configuration tools. Whenever a user would like to connect to a network, change their volume, or connect a Bluetooth device, they have to do this with their environment-specific tool. To a certain degree, this makes sense, users should only see settings, which they can use within their environment. Problems arise when certain environments do not provide their application or perhaps provide one without the needed functionality.

In this case, users would need to find a variety of different applications which combined offer the same functionality. However, due to the split nature of multiple applications, users end up with different levels of polish and different user interfaces. Some might argue that this is the point of these minimal environments, as they sometimes intentionally do not offer this type of software by default, but there is also a distinct lack of this type of software.

A typical example of a minimal environment is a window manager/compositor. In comparison to desktop environments, these do not offer any software other than window management: creating windows, removing windows, window positioning, etc. Any additional software needs to be installed separately, like status bars, editors, media viewers or in this case settings.

Some specific settings like monitor configuration were once universal with tools like Xrandr, but with the introduction of the Wayland display protocol, the original idea of having a universal display server was abandoned, favoring individual implementations instead. This leads to a variety of different ways to configure monitors, very few of them being compatible with each other. In this case, only a plugin system to handle individual implementations could solve this problem [3].

1.1 Objectives

ReSet will make an effort to change this situation by creating a settings application that works across these distributions and environments. This means that both major and specifically smaller environments should be able to use this application to fill the current gap. This allows users to avoid installing one tool for each task, while also using one user interface with consistent design.

ReSet should be able to handle basic audio input and output, network and Bluetooth connectivity for starters. For more advanced features or specific features, such as monitor configurations for individual compositors, ReSet should have a plugin system, that allows for modular expansion of the application. This modularity also allows for interaction with tools such as status bars or the creation of applets. This expands the functionality beyond ReSet as a user interface, and instead allows it to be a middleman between compositor and shell components.

1.2 Challenges

- **Consistency**
  As discussed in Section 1.1, multiple applications will likely not create a consistent experience, however, even with a single application, creating a consistent application with multiple technologies and use cases is not trivial.
- **Interoperability**
  The reason for environment-specific applications is the ease of integration. If one has control over the environment, it is easier to implement advanced features, however, without this control, there is a significant overhead for each feature.
- **Features**
  In order for ReSet to be a viable alternative to existing solutions, it would need to cover the majority of use cases, this means supporting as many features as possible or providing a plugin system for modular expansion.
- **Maintainability**
  With increasing scope, the overhead for developers will increase, causing potential slowdowns in development or release cycles.

The challenges specified in combination with a smaller user base for minimal environments are likely the cause for a missing universal solution. The previous solutions are functional, just not optimal.

1.3 Methodology

This project was created by first evaluating existing projects in Section 2.2 and including techniques from literature in Section 2.3. Further, technologies and potential solutions to implementations were evaluated in Section 3. With this information, the resulting application is documented in Section 4. In the end, the solution is discussed in Section 5, and potential further improvements are mentioned in Section 5.1 and Section 5.2.

2 Research

In order to develop ReSet, several existing applications, potential implementations for features and literature were analyzed.

2.1 Requirements for existing solutions

For ReSet, three different categories will be used to weigh existing projects and potential solutions.

- **Interoperability**
  This is the most important aspect, as it was the main driving factor for this work. Interoperability for ReSet is defined as the ease of use of a particular project in different environments. Important to note, is that multiple factors are considered:
  - **Amount of working features**
    The amount of features of the application that work on every environment.
  - **Amount of non-working features**
    This specifically refers to residue entries that would work on the expected environment, but not in other environments.
  - **Interoperability of toolkit**
    Depending on the toolkit, it might behave differently with specific modules missing from other environments, this would once again increase the amount of additional work needed, in order to get the expected interface.
  - **Behavior in different environments**
    Depending on the environment, applications have different ways of displaying themselves with different attributes like a minimum size. These constraints might not work well with different types of window management. A tiling window manager does not consider the minimum size of an application, as it will place the window according to its layout rule. Applications with large minimum sizes are therefore preferably avoided.
- **Ease of use**
  While functionality is important, the intention is to provide an application that is used by preference. This means that the application should be intuitive and usable without prior knowledge about the application. To achieve this, ReSet will use best practices from the GNOME Human Interface Guidelines as well as user interface literature described in Section 2.3.
- **Maintainability**
  Applications with a plethora of functionality will get large quickly. This poses a particularly hard challenge for developers to keep the project maintainable. Too many features without a well-thought-out architecture will lead to potentially faulty code.

2.2 Analysis of existing applications

In this section, several different applications are analyzed according to the requirements specified in Section 2.1. Both full settings applications provided by environments and standalone applications for individual technologies are compared.

Important to note, that for this section, each application will be evaluated with its default dark theme. For GTK applications, the libadwaita dark theme will be used. For QT applications, the breeze dark theme will be used, configured via the QT_QPA_PLATFORMTHEME=gnome environment flag. This configuration is done to keep a consistent look for all applications, without potentially compromising their look with non-native themes.

2.2.1 GNOME Control Settings

The GNOME control center [4] is as the name implies the central settings application for the GNOME desktop environment, it offers plenty of configuration, from networks to Bluetooth, to online accounts, default applications and a lot more. The application is written in C with the GTK [5] toolkit (GTK4) and follows the GNOME Human Interface Guidelines [6].

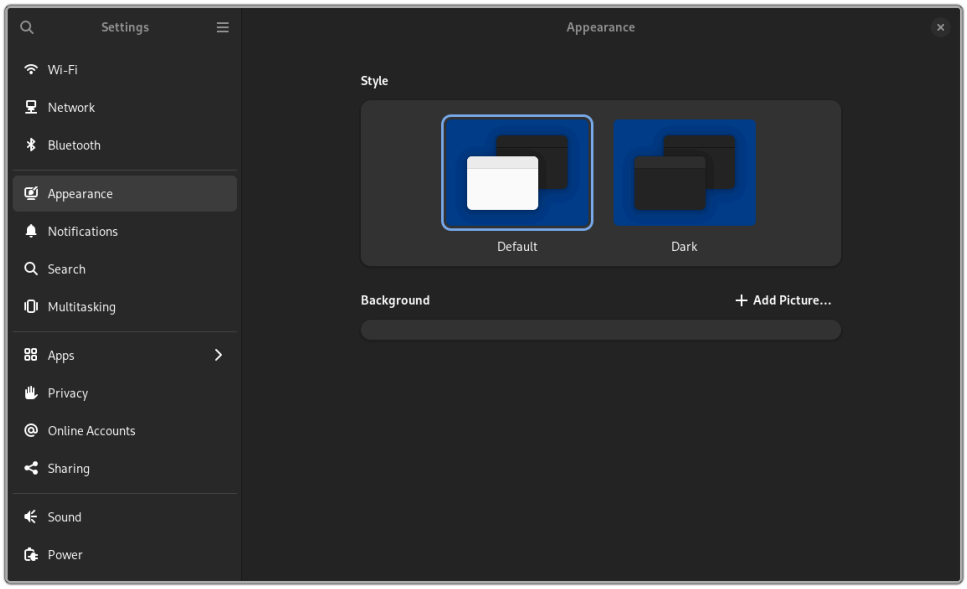


Figure 1: Appearance setting of GNOME control center

The code structure of the control center is very modular, with each tab having its folder and files. Although it is hard to immediately understand each use case of each file. Certain functionality is hard-coded with libraries, like networks, which use the NetworkManager library, while others are implemented via DBus, like monitors.

Settings are stored using dconf [7] which is a key/value system, that is optimized for reading. The form of a dconf file is a *binary* which makes it fast to read for dconf, but not readable for other systems.

GNOME control center is not supposed to be used outside the GNOME environment, especially using the Wayland protocol. Hence, not all functionality will be available on other environments.

Table 1: GNOME Control Center Requirement Fulfillment

Category	Justification	
Interoperability	Not all base features of the GNOME control center work on other environments, and GNOME exclusive features cannot be hidden.	**X**
Ease of Use	The user interface of the GNOME control center follows their own best practices [6]. It has consistent design, naming makes sense and accessibility is taken into account.	✔
Maintainability	All features of the Gnome control center are within one repository, which results in one project maintaining every feature. This can potentially cause complications with increasing size.	(✔)

2.2.2 KDE System Settings

KDE systemsettings [8] is written in C++/QML and is made with the QT toolkit [9]. It follows the KDE style of applications, featuring a very large variety of settings (on KDE), and offering other applications a way to integrate into this application via KConfig Module (KCM)[10].

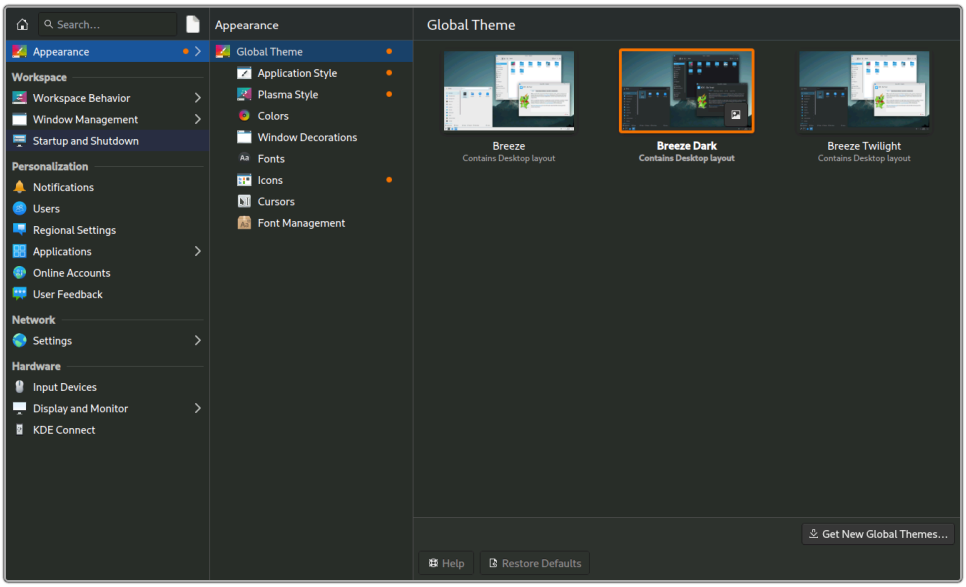


Figure 2: Appearance setting of KDE systemsettings

The program is by default very slim and does not feature any standard settings on the repository. However, Linux distributions usually ship the KDE standard modules, as KDE is the intended environment for this application. For ReSet, KDE systemsettings is still a very good resource for implementing modularity with this type of application.

Settings are stored by individual modules, which means that a lot of individual files will be written/read in order to provide all functionality.

In many cases, for KDE systemsettings it is not the application itself that makes it harder to be used on other environments, but the toolkit and the KDE specific styling of said toolkit that might not integrate well.

Table 2: KDE systemsettings Requirement Fulfillment

Category	Justification	
Interoperability	KDE systemsettings is built with modularity in mind, meaning it works purely with modules that can be built for it. This means that one could create modules for systemsettings to enable functionality in other environments. However, QT is not as well integrated into other environments as GTK, requiring users to potentially change themes for a consistent design. Additionally, the needed modules do not exist as of now.	(✓)
Ease of Use	The most common criticism of KDE systemsettings is a convoluted design. This stems from the number of settings the application can provide, alongside its heavy use of submenus that can become confusing when searching for something specific.	X
Maintainability	The modular design of KDE systemsettings allows for great maintainability on the application itself. It is however noteworthy that the configuration files created by KCM tend to fill folders with seemingly random files. This means that changing settings outside KDE systemsettings can be a challenge.	✓

2.2.3 Standalone Settings

These applications focus on one specific functionality and do not offer anything else. This means one would need to use multiple of these, in order to replicate what the other 2 discussed programs offer.

Interoperability will not be considered for these applications, as they were made for a universal use case.

Pavucontrol | Sound Application

Pavucontrol [11] or PulseAudio Volume Control is an application that handles both system-wide input/output and per-application output/input streams. It is under the umbrella of the Free Desktop project and is directly involved in PulseAudio itself. The application itself is written in C++ and GTK3.

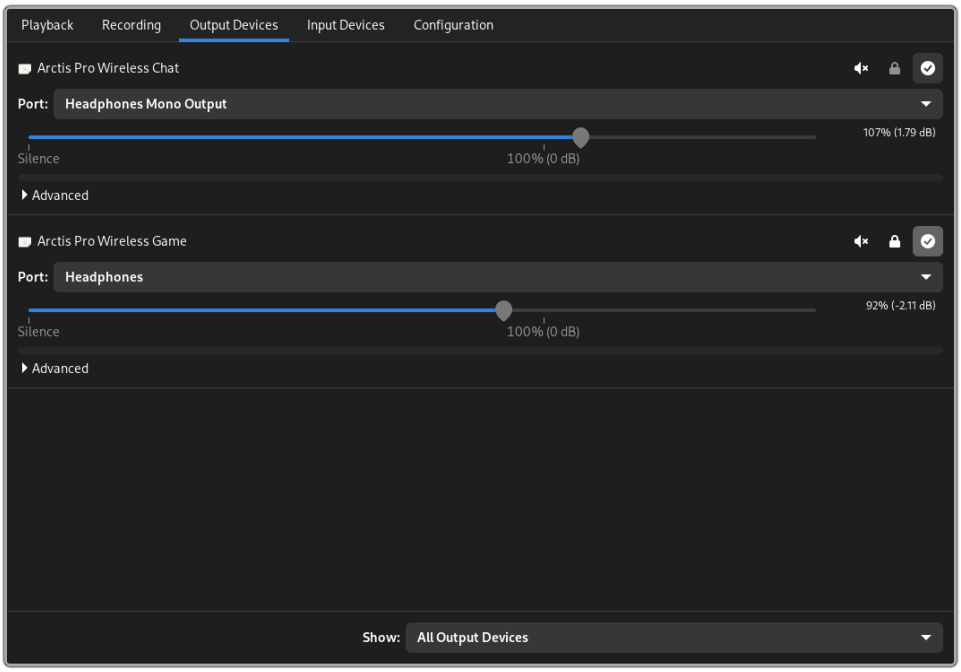


Figure 3: Output devices of pavucontrol

Pavucontrol opts for a simple but functional user interface with different functionality being grouped into their own tab. Noteworthy is the clear differentiation between playback and output devices, which separates applications using an output device from the output device itself.

Table 3: Pavucontrol Requirement Fulfillment

Category	Justification	
Ease of Use	While pavucontrol is generally made for more advanced users, it does offer a consistent design with appropriate icons/naming and integrates well into all environments.	✔
Maintainability	Pavucontrol is made with a modular codebase, which allows for easier adding of features. Note: pavucontrol is feature complete, and will likely not get more features in the future.	✔

Blueman | Bluetooth Application
Blueman [12] allows connecting and managing of Bluetooth connections, as well as some quality-of-life features like file transfer. It works great in functionality, but the buttons are not very expressive of what they will achieve. Blueman is written in Python and GTK3.

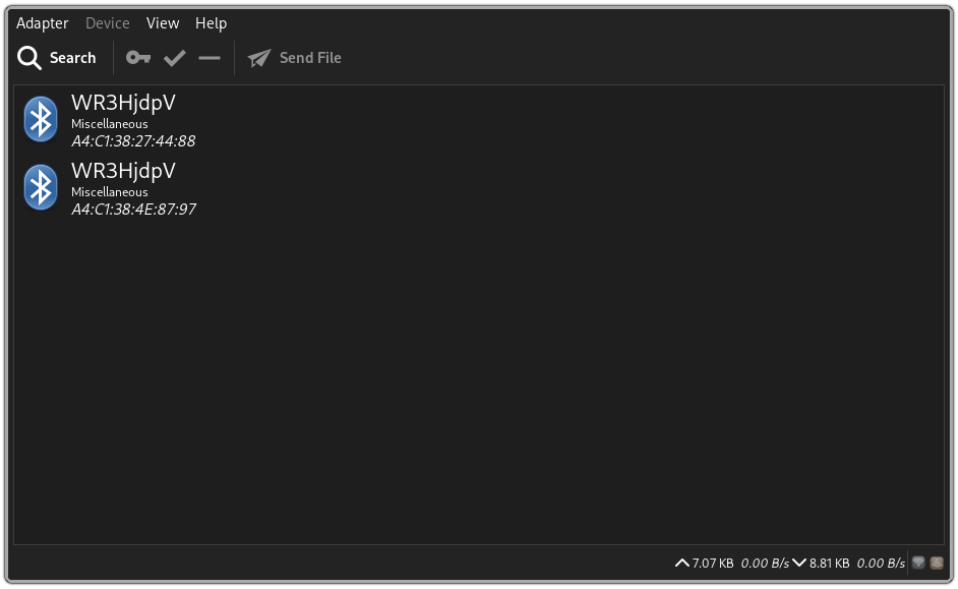


Figure 4: Main window of blueman

On top of the user interface application, Blueman also offers a tray-applet for status bars, which is started alongside Blueman. This means that one could use Bluetooth without this user interface by interacting with the applet instead.

Table 4: Bluetooth Manager Requirement Fulfillment

Category	Justification	
Ease of Use	The user interface for Blueman can be rather confusing, for example: there is no obvious connect button, which might lead to a user trying to mark a device as trusted instead of connecting to it. (trusted is the ✔ button). Blueman also tends to use older icon designs.	**X**
Maintainability	Blueman is created in a modular fashion and can be considered easily maintainable.	✔

Nmtui | Network Application
Nmtui [13] is what the name suggests, it is a terminal user interface that allows users to use and edit network connections, including VPN connections. Nmtui is located in the same project as the network manager itself and is therefore also shipped as part of the network manager package. Both network manager and Nmtui are written in C. There is a specific lack of standalone user interface applications for network managers. Technically, the "network-manager-applet"[14] by GNOME exists, however, this is to be included in a system tray within status bars and does not work on its own.

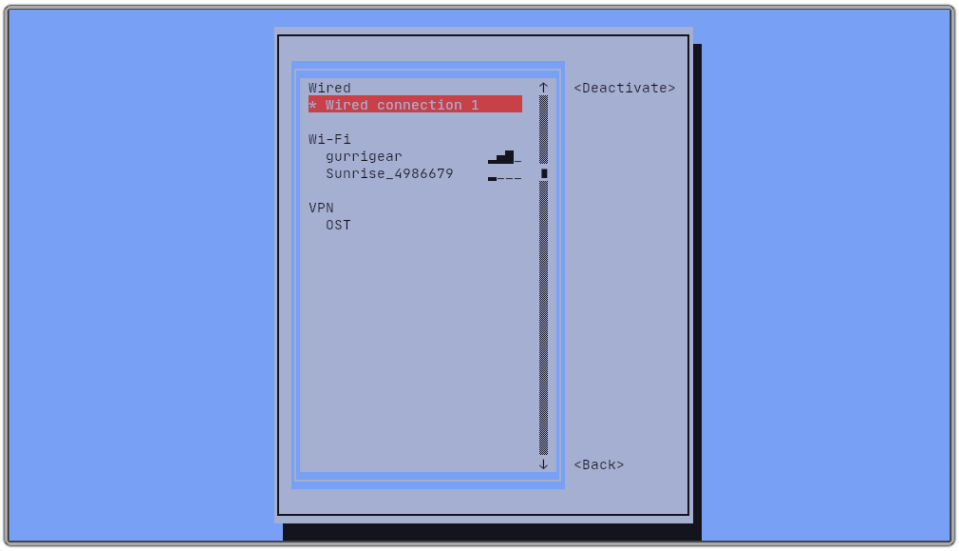


Figure 5: Wi-Fi connections in Nmtui

Alongside Nmtui, users can also choose to use Nmcli, which would be the same application but without a user interface. Nmcli can also be found within the NetworkManager repository.

Table 5: Nmtui Requirement Fulfillment

Category	Justification	
Ease of Use	Resizing the terminal breaks the appearance of the application. There is only a single theme. Users unfamiliar with terminal user interfaces might be unable to use this application. Mouse support is missing.	**X**
Maintainability	The scope of this application is small and depends fully on the parent project, it can be considered to be maintainable.	✔

2.3 User Interface Guidelines

The GNOME Human Interface Guidelines [6] are likely the most applicable for ReSet, as they are the most prominent in the Linux sphere and are directly meant to be used with GTK4, a potential user interface toolkit for ReSet. While Reset does not intend to belong to the GNOME circle, for which rather strict adherence to these guidelines is needed, ReSet will still use the best practices that are employed by these guidelines.

These best practices can also be seen in the book "Don't Make Me Think" by Steve Krug [15], or in "Designing the User Interface" by Ben Shneiderman and Catherine Plaisant [16]. In both works the authors defined rules to follow when creating user interfaces. Steve Krug specifically focused on the web, however, the vast majority of these rules can be applied in the same manner on desktop applications.

- **"Don't Make Me Think" - Steve Krug** [15]
  This means that a user should not need to "think" when using the interface, it should come as obvious where the user has to click or navigate to in order to finish their task. In the case of ReSet, this might be something like connecting to a Wi-Fi network or a Bluetooth device.
- **"Cater to universal usability" - "The eight golden rules"** [16]
  User interfaces should be created for all possible users, meaning both experts and novices should be able to use the application without issues.
  This rule depends heavily on what the possible user base is, an IDE has different users to a general messaging application.

  For example, the application Blueman introduced in Section 2.2 is used.

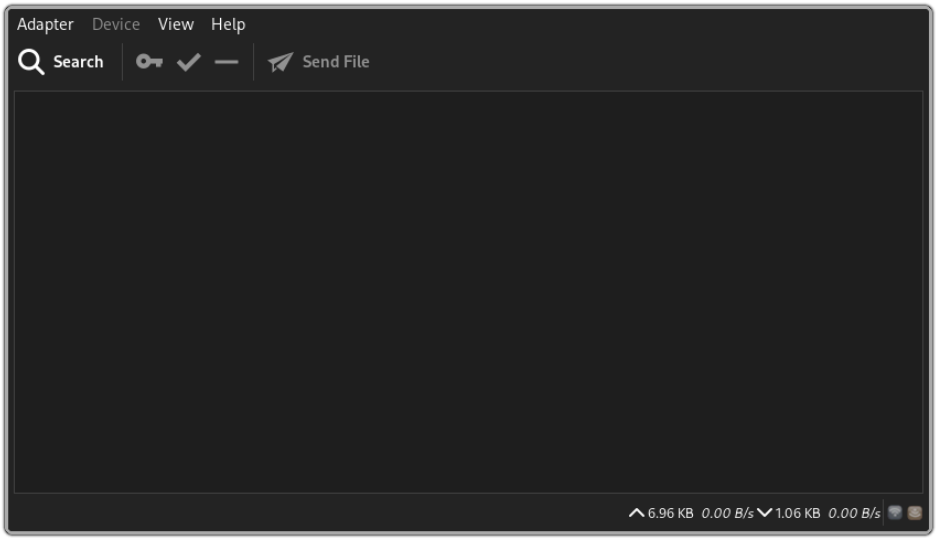


Figure 6: Blueman

When opening the application, one can see a blank page with a top bar, this may cause users to think there are no Bluetooth devices available. However, the user needs to manually search for devices as defined by the technology.

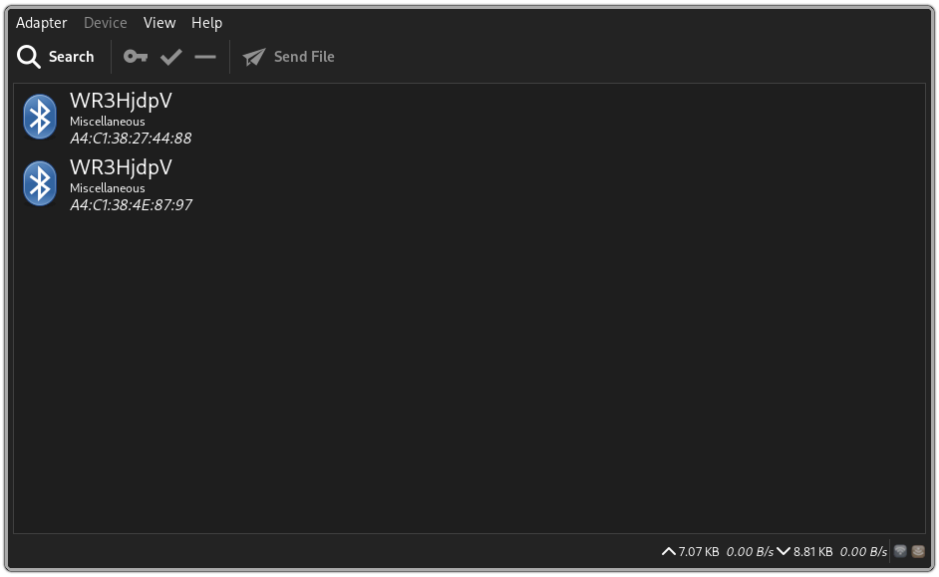


Figure 7: Blueman with scanned devices

After clicking the search button, the application proceeds to list Bluetooth devices as expected. The question now is how to pair and connect to a specific device. Users who understand Bluetooth terminology will likely proceed with the key icon, which means pairing for this application. However, there is a chance that users will first try the checkmark icon to connect to the device, which would instead mark the device as trusted.

In this case, the icons could be improved to represent a more technology-neutral design.

ReSet will enforce technology neutral buttons where possible, which leads to a more intuitive user interface, while still providing the user as much functionality as possible.

- **“It doesn’t matter how many times I have to click, as long as each click is a mindless, unambiguous choice.” - Steve Krug** [15]
  Each navigation should have a consistent path and a clearly defined destination, it should be clear to the user where they are right now, and where they can go from here.

  In chapter 6.4.2 “Tree-structured menus” in “Designing the user interface” [16], the authors note that no more than 3 to 4 layers should be used for pop-over menus. Later this is re-affirmed with the menu selection guidelines where “broad-narrow” is preferred over “narrow-deep”, this defines that more top level selections are preferable to many layers, as layers can confuse the user and make navigation tedious.

  Both the GNOME human interface guidelines and Steve Krug advise developers to use as few required clicks to navigate to a certain page as possible. This avoids tedious navigation where users could either get lost in navigation or simply get annoyed at the endless path.

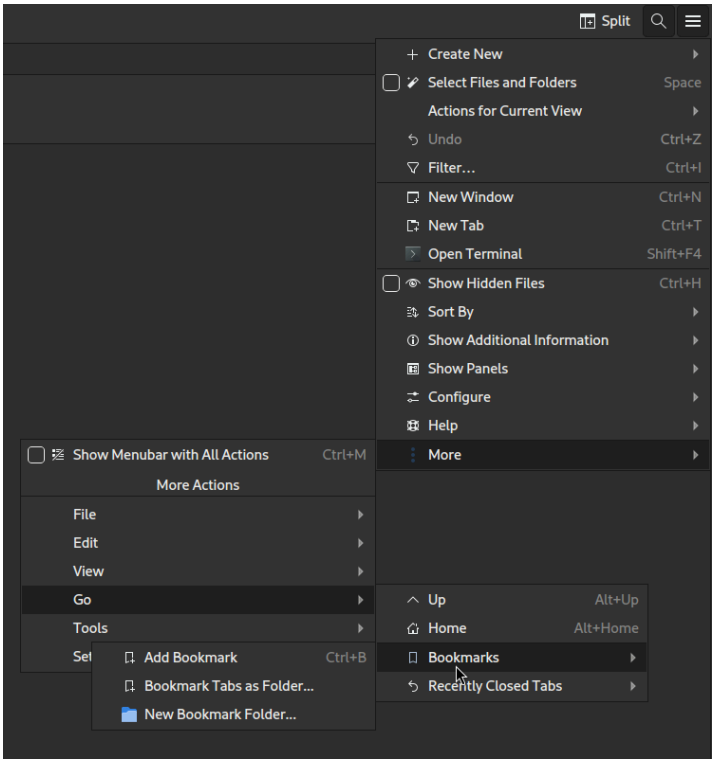


Figure 8: An extreme example of KDE hamburger menus

There is a big difference between KDE and GNOME in the context of menus, KDE uses a lot of nested submenus as can be seen in Figure 8, while the GNOME side explicitly discourages these menus citing reduced accessibility [17]. In this case, the question is about where the bookmarks are stored. The default location is in more->go->bookmarks, this is on layer 4 of a menu without search functionality and with very ambiguous navigation.

In other words, it is clear that shorter navigation is usually the best way to achieve “mindless navigation”, however, it is, as Krug mentioned, not the only factor.

ReSet will enforce this rule by avoiding context menus entirely wherever possible, and instead relying on dynamic pages to provide further functionality.

- **“Get rid of half the words on each page, then get rid of half of what is left.” - Steve Krug** [15]
  This defines unnecessary information on a page or application. Everything that the user does not care about should be omitted. One should note however that this does not imply the removal or omitting of *features*, instead only showing users a certain feature when they need it.

  This is convergent with the previous rule which ReSet will also handle with dynamic pages, meaning advanced functionality will be provided with a button that leads to a page transition.
- **“Reduce short-term memory load” - “The eight golden rules”** [16]
  Humans have a limited capacity for information, this includes what people see on an application. The authors therefore propose to collapse complex user interfaces like multi-page displays into one.

  The downside of this approach can be a too simple application, meaning the users might be forced to use more than one tool for the task. For example, compared to KDE applications, GNOME is often considered to be simpler, but also less configurable.

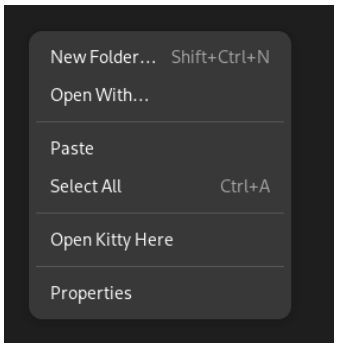


Figure 9: Context menu in Nautilus (GNOME file manager)

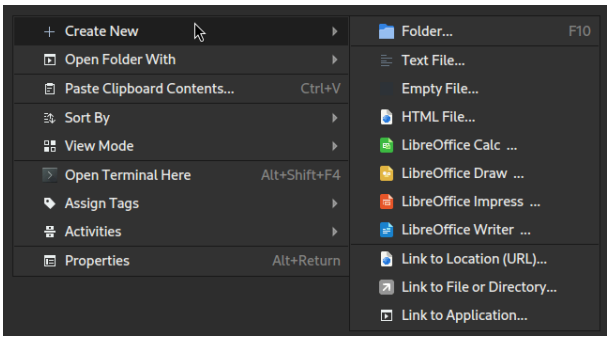


Figure 10: Context menu in Dolphin (KDE file manager)

Here the KDE application is more powerful, offering a variety of files to create, including links and shortcuts, while the GNOME experience only offers a new folder and anything else needs to be done with a terminal. For Nautilus, one can also create templates within the Linux template folder which will then be available in the pop-over menu as well, but this is not an obvious feature [18], [19].

ReSet aims to provide a middle ground between these two approaches.

- **“Support internal locus of control” - “The eight golden rules”** [16]
  The rule refers to clear omission of unnecessary data for experts, and giving them a short, clear, and usually customizable path to their end goal. In user interfaces this is usually done with keyboard shortcuts as seen in Figure 11.

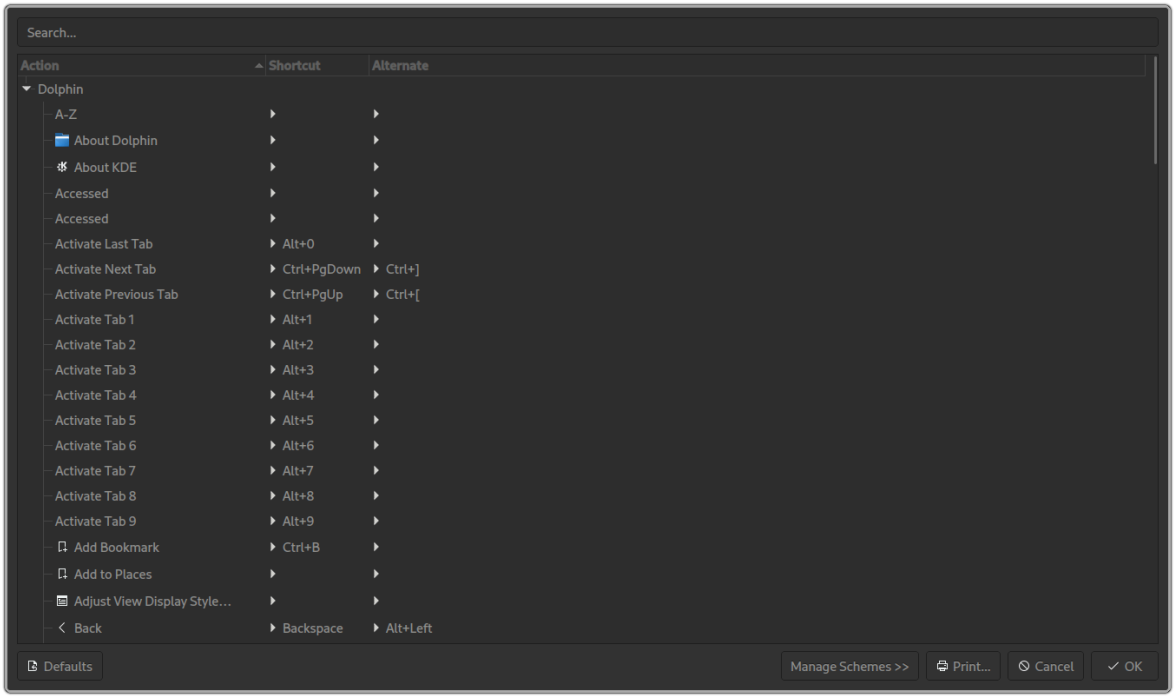


Figure 11: Shortcuts menu in Dolphin

The Dolphin version is much more complex, this is because you can customize any shortcut in Dolphin using this interface. For Nautilus shortcuts can only be configured using scripts that extend the functionality of Nautilus [20], according to a GTK developer, this is no longer possible on a toolkit level [21].

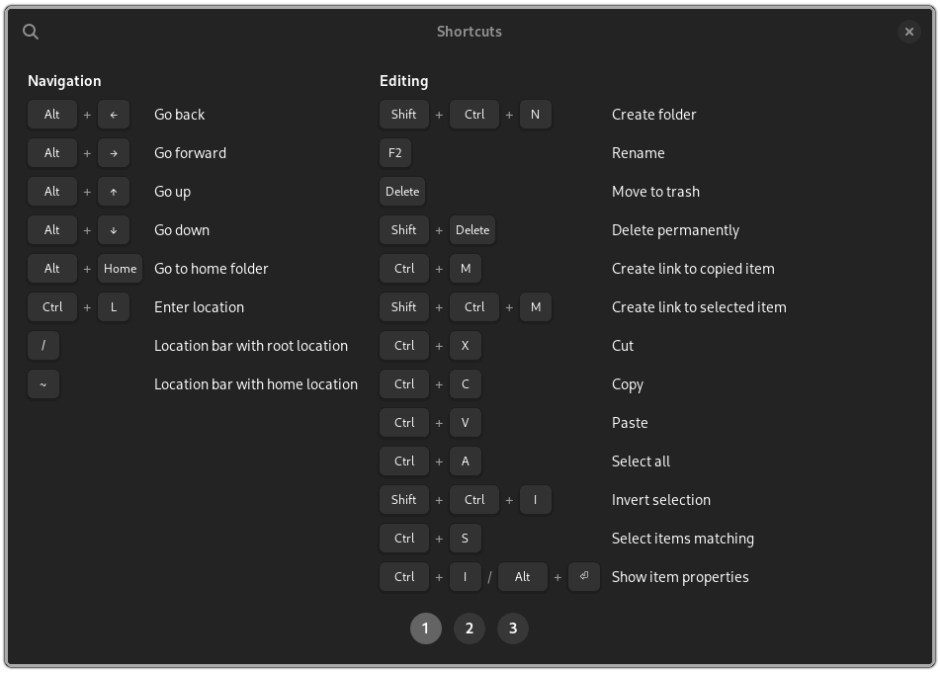


Figure 12: Shortcuts menu in Nautilus

- **“Strive for consistency” - “The eight golden rules”** [16]
  No matter how one creates a user interface, the very first thing one should thrive for is to make it consistent. This includes practices like ensuring items have the same appearance no matter where they are placed or that buttons with similar functionality and have the same labeling. In “Designing the user interface” [16], the authors empathized this in chapter 2.4.3 with a clear example:

Consistent	Inconsistent
delete/insert table	delete/insert table
delete/insert column	remove/add column
delete/insert row	destroy/create row
delete/insert border	erase/draw border

  For ReSet, this rule will be used to ensure that a similar user interface flow as well as similar toolkit modules are employed to guarantee consistency.

2.4 Persistent Settings

In “The Pragmatic Programmer” [22] by David Thomas and Andrew Hunt, the authors mentioned the importance of text file configuration that is human-readable and can be put under version control. For ReSet the importance comes from the fact that the configuration of various categories needs persistent storage, and ReSet also needs to handle this.

The already mentioned GNOME control center handles this via a database, contrary to the chapter in the aforementioned book. This is done for increased speed in loading and storing configuration data. However, it does come with the downside of needing a program to interact with the stored data, as you cannot otherwise access it.

KDE systemsettings in contrast work with text files as Thomas and Hunt advocate, but their files are sometimes unorganized, meaning it is often hard to understand where a certain setting might be if you would like to change something, perhaps due to a bug in the graphical user interface or because you just like to make these changes manually.

For ReSet, using small and structured text files over databases ensures version control compatibility, environment-agnostic usage, as well as application agnostic usage and configuration.

2.5 Plugin System

As read on NullDeref [23], there are multiple ways to create a potential plugin system for ReSet:

- **WASM**
  Web assembly is a very flexible way of creating plugins, as it is based on something that will work on every device that has a WASM target. However, WASM is not something known to anyone involved in this project, which is why it is not covered further.
- **Scripting Languages**
  Languages like Lua have succeeded integrating in multiple fields such as game programming and even the Neovim editor [24]. In both cases, it expands the potential functionality by giving developers a fully functional programming language while still keeping the original system with a more performant system programming language.
- **IPC**
  With inter-process communication, one will have a lot of overhead when talking about a plugin system, however, it is a lot easier to write. For this project, IPC can be considered as a potential solution, should the other alternatives not be viable.
- **Dynamic Loading**
  This refers to dynamic libraries that are loaded at runtime. These dynamic libraries are the most performant way to create a plugin system without outright moving towards changing the code and recompiling. However, it forces ReSet to offer a "stable" ABI which can be done directly over the C programming language, or indirectly with the abi_stable crate for the Rust programming language.

  The project "anyrun" [25] by Kirottu serves as a perfect example of a small but powerful example of the abi_stable crate. Anyrun is a so-called application launcher, with each plugin being able to fill the launcher queries.

```rust
use abi_stable::std_types::{RString, RVec, ROption};
use anyrun_plugin::*;
#[init]
fn init(config_dir: RString) {
  // ...
}

#[info]
fn info() -> PluginInfo {
  PluginInfo {
    // ...
  }
}

#[get_matches]
fn get_matches(input: RString) -> RVec<Match> {
  // ...
}

#[handler]
fn handler(selection: Match) -> HandleResult {
  HandleResult::Close
}
```

Listing 1: Example code for an anyrun plugin, available at [25]

With just 4 functions, an anyrun plugin can be created which will fill a dynamic library functions to be loaded at runtime.

3 Implementation Evaluation

In this section, the initial plan for ReSet is created. This includes evaluating technologies for ReSet and an initial design for the user interface.

3.1 Technologies

Technologies are evaluated using a value table, which defines a score between 0 and 10 for each category over each tool. Each category is also given a constant weight, in order to evaluate which tool will be chosen.
The weights are as follows: low -> 1, medium -> 2, high -> 3

The following categories are evaluated for programming languages:

- **Familiarity** | weight: medium
  Indicates how familiar the developers are with a certain tool. More familiarity means an easier development process without surprises. Note that familiarity is the subjective relative experience compared to other languages and does not indicate a particular level of skill.
- **Developer Experience** | weight: low
  This encompasses the entire development cycle, meaning toolchain, LSPs, formatters, code coverage tools and more. With a modern developer experience, you can guarantee functionality without prolonged setup phases.
- **Ecosystem** | weight: high
  Ecosystem is defined as the amount and the quality of packages and libraries that are available for this language. For ReSet, this can be a crucial category, as many different types of services are used, hence the need for good integration with them.
  Important: The ecosystem is highly dependent on the Linux desktop, which is not always favorable for all tools, for example: .NET MAUI, a very popular user interface toolkit is not usable, as it does not run on the Linux desktop [26].
- **Runtime speed** | weight: low
  Runtime speed is likely only a concern for the daemon, and even in this case, it is unlikely to be too slow with any modern programming language.
- **Resource usage** | weight: medium
  Many computers have enough RAM by now, however, ReSet intends to work on any distribution, including lightweight distributions meant for older or lower-end systems, therefore RAM usage should be a concern, especially for the daemon.
- **Development speed** | weight: medium
  ReSet is limited in time scope, therefore tools with decent development speed should be considered. Note that this includes the time needed for debugging and potential problems, such as undefined behavior or dynamic type issues.

Special Requirement: All tools used in this project **must be published under an open-source license**, as ReSet will be published under the GPL-3.0 license.

Table 6: Programming Languages

Category	Python	TypeScript	C#	C++	Rust	Weight
Familiarity	8	6	7	6	6	*2
Developer experience	7	9	7	3	10	*1
Ecosystem	6	5	5	10	9	*3
Runtime speed	5	5	8	10	10	*1
Resource usage	5	5	8	10	10	*2
Development speed	9	9	8	5	5	*2
Total	74	69	76	85	89	

Programming Language | ReSet is written in Rust.
Rust was chosen for its speed, low memory usage, memory-safe design and robust ecosystem. While Rust is more complex to write than languages such as JavaScript, it comes with a significantly reduced memory cost and with the addition of a static type system [27]–[29].

Compared to other system programming languages, Rust comes with a modern ecosystem out of the box, providing a formatter, a compiler, an LSP server, a code-checking tool and a package manager in one (Rustup) [30]. This allows for a more streamlined developer experience and standardizes features, which in return makes more complex tasks like cross-compilation a lot easier. For example, Rust allows adding a so-called "target triple", which is a specific platform. Using this triple, it is possible to build the project with "`cargo build --target x86_64-linux-unknown-gnu #or other platform`". Similarly, adding packages is also just one command "`cargo add gtk`", which is comparable to npm while still offering C/C++ runtime speed.

Another consideration for this language is the technology stack. For many other languages, only a specific set of tools allows for a full IDE workflow, including debugger, LSP and more. With Rust, this is not the case, as it either provides said tool or uses a well-established open-source tool for each task [31], [30]. This avoids cases like the official and proprietary C# debugger, which only works with official Microsoft tools [32], [33], or the C++ problem of having multiple compilers with different feature sets [34]. Hence, both languages described will have different experiences on different platforms and editors/IDEs.

The following categories are evaluated for UI toolkits:

- **Familiarity** | weight: low
  Indicates how familiar the developers are with a certain tool. More familiarity means an easier development process without surprises. Note that familiarity is the subjective relative experience compared to other toolkits and does not indicate a particular level of skill.
- **Language Integration** | weight: medium
  This defines how well the chosen language will integrate with the UI toolkit, in other words, certain toolkits might get a zero score here, signifying incompatibility.
- **Documentation** | weight: medium
  Defines how well the toolkit is documented. This will be important when going beyond the typical "Hello World" for UI programs. A well-documented toolkit can reduce the development time by magnitudes.
- **Features** | weight: low
  ReSet does not need many features, however, ReSet does require first-class support for the Linux desktop.

Table 7: UI Toolkits

Category	GTK	Iced	QT	Weight
Familiarity	**6**	**4**	**0**	***1**
Language Integration	**7**	**10**	**6**	***2**
Documentation	**8**	**4**	**8**	***2**
Features	**8**	**6**	**10**	***1**
Total	**44**	**38**	**38**	

UI Toolkit | ReSet uses GTK4 as its UI toolkit.
GTK (GNOME) toolkit, formerly Gimp Toolkit) is a well-established, LGPLv2.1+ licensed, cross-platform UI toolkit that has seen decades of usage and improvements [5]. While the library itself is written in C, it does offer stable language bindings for a large set of languages, including Rust via gtk-rs [35]. Compared to native Rust libraries, it offers a more robust set of defined widgets, themes and tools. Specifically the toolkit "iced" was considered, however, it currently lacks documentation and needs several library implementations in order to fit with ReSet's requirements [36].

The last consideration is QT, it is a cross-platform toolkit that uses its own form of JavaScript (QML) to draw windows [9], [37]. QT is a well-known toolkit which also features Rust bindings, however, it is completely unknown to the project team members, making it a suboptimal choice.
For QT, there is also the consideration of integration mentioned in Section 2.2.

3.2 Architecture

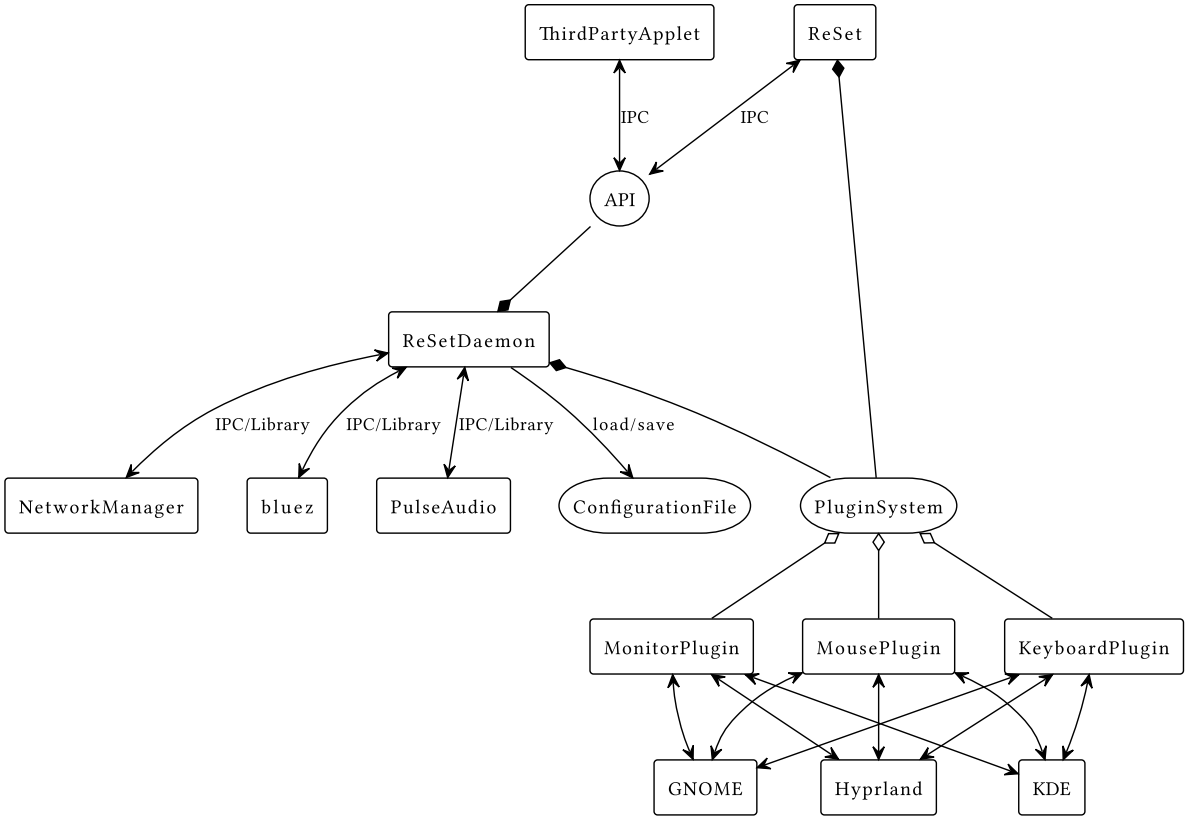


Figure 13: Architecture of ReSet

3.2.1 Description

Hyprland, GNOME, KDE | Various desktop environments or window managers/compositors.
PulseAudio | The de-facto default Linux sound server.
bluez | The standard Linux Bluetooth stack.
NetworkManager | A very popular Linux network stack.
ReSet | The graphic user interface for ReSet.
ReSet-Daemon | The backend process for ReSet that handles functionality.

Base functionality is provided using well-known Linux system services such as PulseAudio [38] for audio, NetworkManager [13] for network and bluez [39] for Bluetooth. These services will be either integrated IPC APIs or libraries, depending on the service. Additional systems will be integrated with a plugin-system which allows for integration of specific environments.

Interaction with the daemon can either be done with the user interface application provided by ReSet or with IPC interaction from another source, this is explicitly done to allow third-party applications to also use ReSet.

The plugins are made via dynamic libraries, which are loaded by both the daemon and the user interface application. The daemon part handles the functionality of the plugin, while the user interface application needs a way to display the data to the user in a suitable manner.

The example environments are only a selection, a plugin could be made for any environment.

3.3 Domain Model

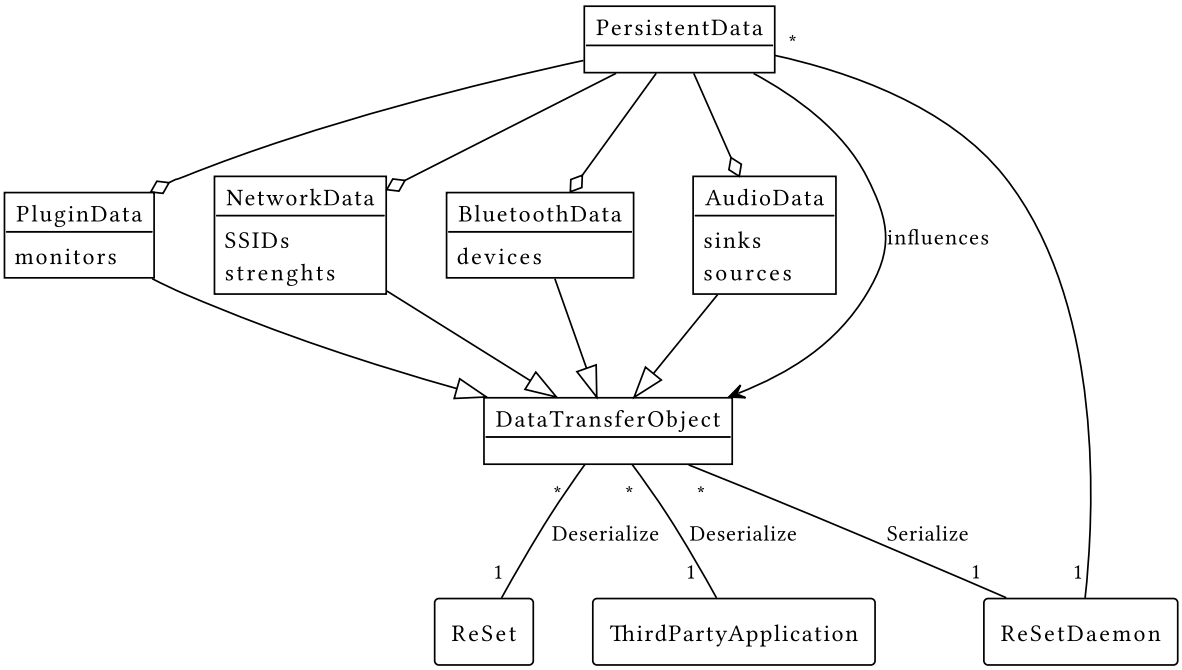


Figure 14: Domain Model of ReSet

3.3.1 Description

Persistent Data | A subset of the service data that will be saved for later use (persistence).
Plugin Data | Data created for a specific plugin, needs to be provided by the plugin developer.
ReSet | The graphic user interface for ReSet.
ReSet-Daemon | The backend process for ReSet that handles functionality.
Data Transfer Object | Serialized data that will be sent to ReSet or third-party applications.

Each functionality defines their own DTOs that will be sent to the user interface application, which will then be able to utilize the data.

Plugins will also handle their own data structures, the daemon will only send the data to the user interface application, which then in return calls another function from the plugin, in order to display the data to the user. Additionally, should a plugin want to store data to persistence, it will have to provide the data to store in a serializable manner (PersistentData object).

Persistent data is deserialized by the daemon, this data will then be merged with the runtime data by the adjacent service. This would make functionality like setting up custom key mappings at startup possible.

3.4 UI Design

3.4.1 UI Build Toolkit

There are multiple ways to build GTK user interfaces. The first one is to use the GTK library itself to build the UI directly in the code. The next two tools are UI builders like Glade [40] and Cambalache [41]. These tools will also be evaluated using a value table with a score range between 0 and 10 for each category. Each category is also given a weight to express impact.
The weights are as follows: low -> 1, medium -> 2, high -> 3

The following categories are evaluated:

- **Functionality** | weight: high
  Describes how many features the tool has to help with development. More functionality means that it is easier and faster to implement certain features. But too many features will increase complexity which is a detriment to ease of use.
- **Ease of use** | weight: high
  Indicates how intuitive the tools are. Better ease of use means the learning curve is lower. This also means a lowered complexity which makes it easier to understand what each feature does. It also includes how easy it is to find information about a tool on the internet. A lack thereof means a lot of research on the tools itself just to implement basic features.
- **Collaboration** | weight: medium
  Describes how well the tool can be used in a team. This includes version control, how the tool can be used by multiple people at the same time and how easy merge conflicts can be solved.
- **Debugging** | weight: low
  Tools that help developers find problems and bugs in the UI. This includes error messages and warnings and their quality.
- **Updates** | weight: medium
  ReSet uses the latest version of GTK, which means that the tools should also be updated regularly to support the latest features.

Table 8: UI Builder tools

Category	Code UI	Glade	Cambalache	Weight
Functionality	**4**	**1**	**8**	***3**
Ease of Use	**2**	**8**	**6**	***3**
Collaboration	**9**	**2**	**2**	***2**
Debugging	**5**	**5**	**5**	***1**
Updates	**10**	**0**	**9**	***2**
Total	**61**	**36**	**69**	

ReSet will use Cambalache to build the UI, as it provides an application in which the UI can be built with drag and drop. This makes it easy to instantly see how the UI looks. Glade does provide the same

functionality because Cambalache is heavily inspired by it, but it does not support GTK4 and is no longer maintained. While a code created user interface has impressive scores in the Collaboration and Updates categories, it is harder to use because there is no UI builder.

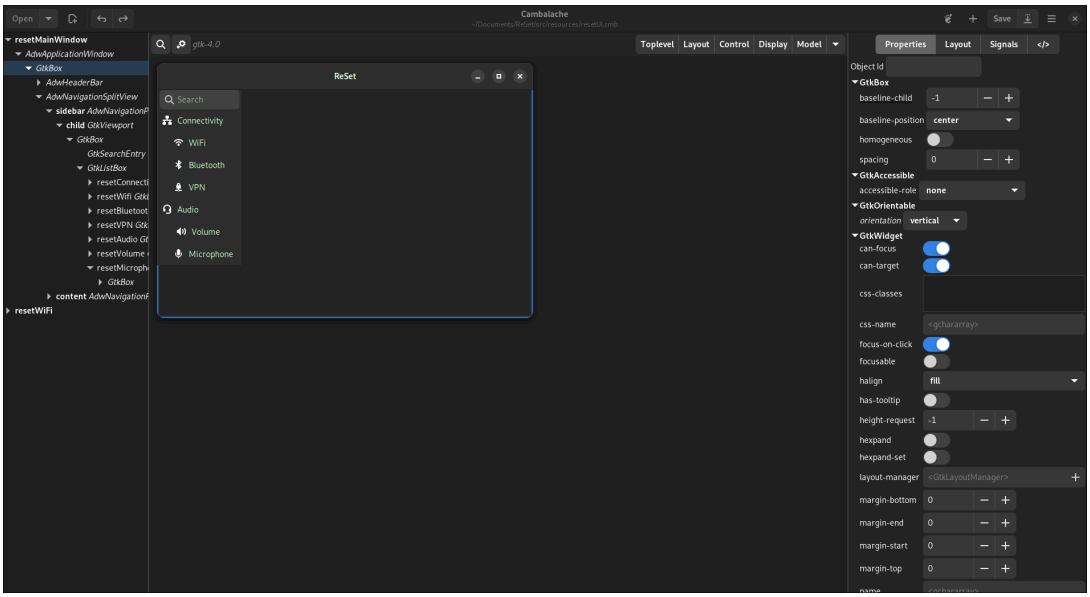


Figure 15: Early UI of ReSet in Cambalache

Libadwaita

In addition to GTK, ReSet will also use a library called libadwaita [42]. Libadwaita is a library that extends the features of GTK. It is developed by the GNOME project and thus follows the GNOME Human Interface Guidelines as well. It provides a lot of beautiful UI widgets that make the UI feel modern.

3.4.2 UI Mock

The initial UI mocks have been made in a generic PDF tool. The goal of these mocks is to get a general idea of how the UI should look like and to figure out a good user experience.

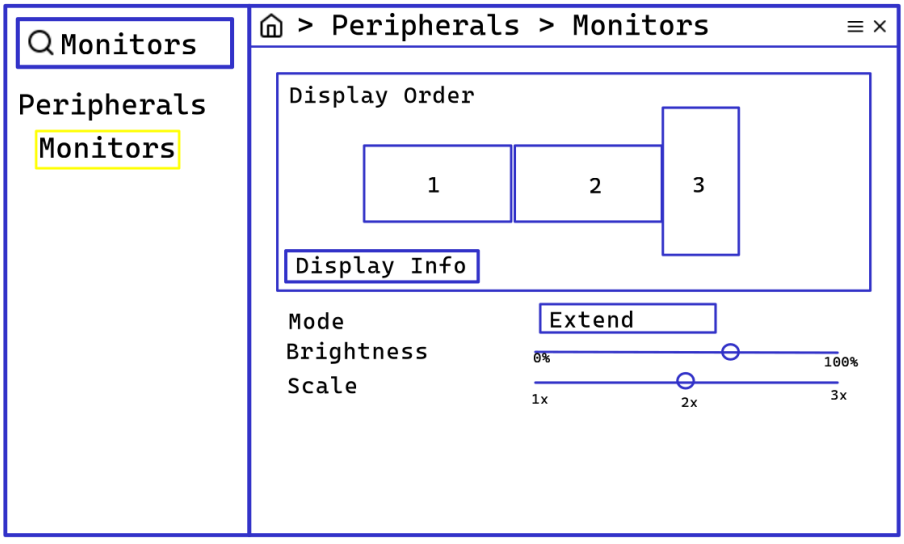


Figure 16: UI mock of monitor setting

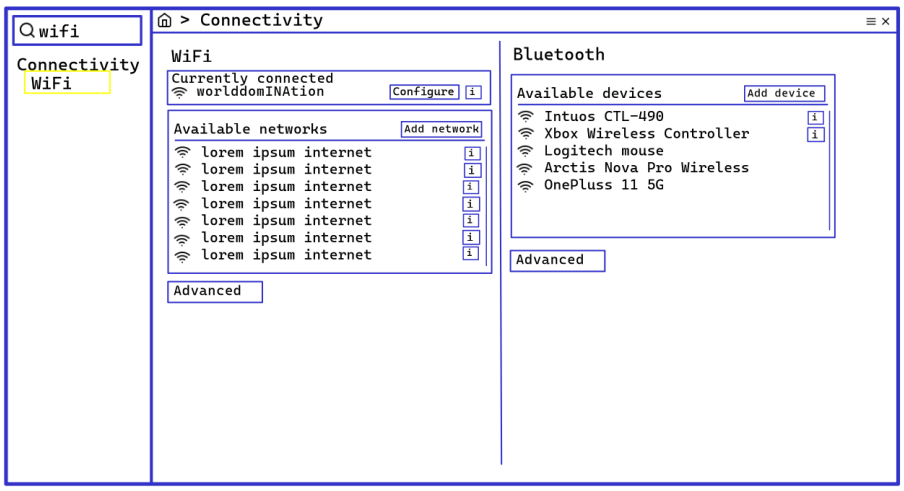


Figure 17: UI mock behavior on wide monitor screen

UI structure

The settings app can be divided into two parts. The left part is a sidebar that contains a list of all settings and the right side displays the actual setting where the user can make changes. While this is not a standard, it is a very common layout structure as seen in Section 2.2 and can also be found on other operating systems like Windows and macOS. The list of settings contains multiple categories like Connectivity, Sound and more, with each of them containing more subcategories.

Because a settings app contains a lot of configuration options and customization features, having a search bar is a necessity. To make it as smooth as possible, the list of setting categories is updated for every character typed to only show relevant options. This has already been implemented in JetBrains Rider for example. In Figure 18, searching for specific settings reduces the list to a handful of matching entries that may contain the setting the user is looking for.

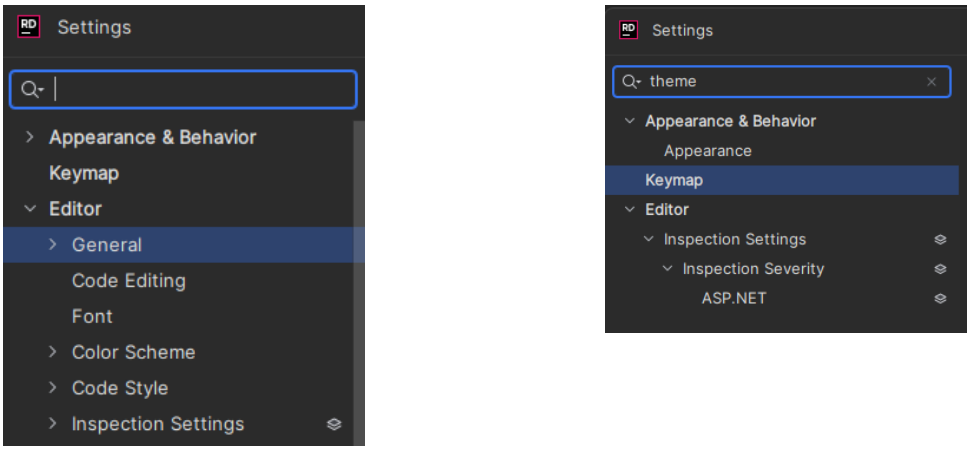


Figure 18: JetBrains Rider setting search bar

The right side displays the settings that can be configured. It is dynamically adding new settings to the screen if there is enough space to facilitate it. For example, if the user clicks on a category like Connectivity, it will show the Wi-Fi settings and if there is enough space, Bluetooth and VPN settings are also displayed next to it in a flowbox layout. But if the user clicks on the Wi-Fi setting, only the Wi-Fi setting is visible because it is clear that the other settings are irrelevant. This is especially useful on bigger monitor sizes, because there is generally a lot of unused space in the settings applications mentioned in Section 2.2, especially if the window is displayed in full screen.

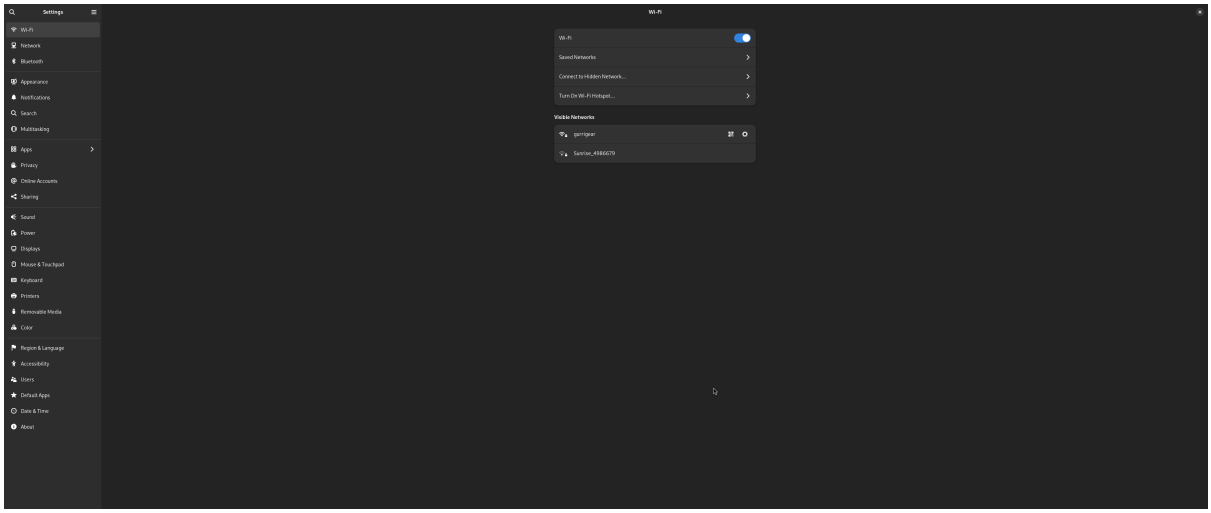


Figure 19: Full-screen Gnome control center settings window on ultra-wide monitor

On top of that, the settings are structured in a hierarchical order, which allows ReSet to have a breadcrumb menu similar to file paths. This hierarchical order allows users to navigate using the Back button.

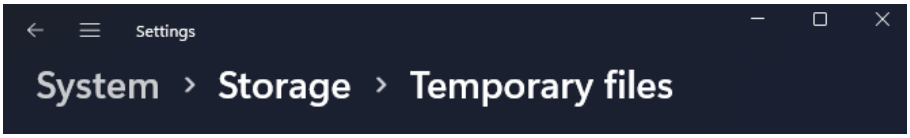


Figure 20: Windows 11 breadcrumb menu

In Figure 21, the first UI mock that follows all the mentioned ideas using Cambalache is visualized.

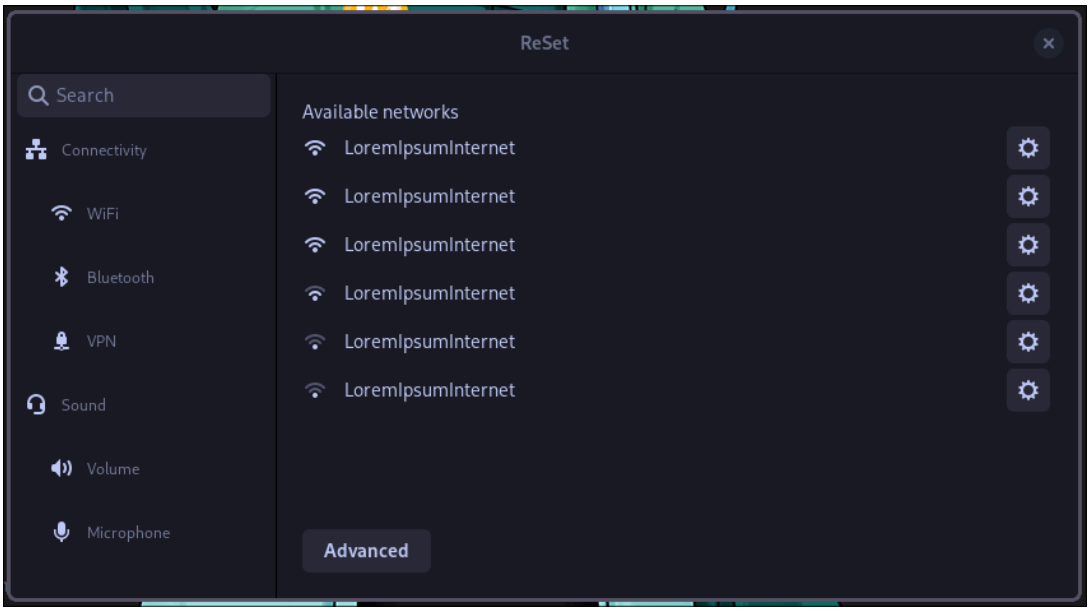


Figure 21: UI mock of Wi-Fi setting

4 Results

In this section, the results of ReSet are discussed.

4.1 DBus

As explained in Section 3.2, inter-process communication is handled via DBus, in this section, the DBus usage is elaborated.

The API for ReSet can be found in Section 11.8 or on docs.rs [43] for an updated version.

4.1.1 DBus Types

In order to properly understand the DBus API, the following table will provide information about DBus types and how they correspond to the regular Rust types.

Table 9: DBus Type Table

DBus Type	Rust Type	Generic Type
y	u8	unsigned byte
b	bool	boolean
n	i16	signed 16bit integer
q	u16	unsigned 16bit integer
i	i32	signed 32bit integer
u	u32	unsigned 32bit integer
x	i64	signed 64bit integer
t	u64	unsigned 64bit integer
d	f64	64bit floating point number
s	String/&str	String
o	Path<'static>	DBus object path
a	Vec<T>	DBus Array

4.1.2 Daemon and Application

DBus first registers a service-providing application with a name. This name will then either be available for the user session in the operating system or the entire system, depending on the session the application requests. ReSet will only request a session namespace, but it will communicate to sytem namespaces such as the org.freedesktop.DBus namespace.

On this namespace, the application can register objects that will be responsible for providing functionality to potential clients. Each object does this by using interfaces that the application can define, this means that you can implement both generic interfaces for each object to implement, or create a specific interface for a specific object [44].

In Listing 2, an example client is shown which calls a function on the org.Xetibo.ReSet namespace.

```rust
// spawn a new thread to not block the GUI thread
thread::spawn(|| {
    // create a temporary connection for DBus
    let conn = Connection::new_session().unwrap();
    let proxy = conn.with_proxy(
        "org.Xetibo.ReSet.Daemon",  // The DBus name to target
        "/org/Xetibo/ReSet/Daemon", // The DBus object path for the daemon
        Duration::from_millis(100),
    );
    // The return type of this DBus method
    // The error is necessary as a call to DBus can also fail
    let res: Result<(), Error> =
        proxy.method_call(
          "org.Xetibo.ReSet.Daemon", // The DBus interface
          "UnregisterClient",        // The DBus method
          ("ReSet",));               // The provided parameters
    res
});
```

Listing 2: Example DBus usage in a client of the ReSet-Daemon

In order to further understand the role of the ReSet-Daemon, Figure 22 elaborates on how both the client and daemon interact with other DBus applications.

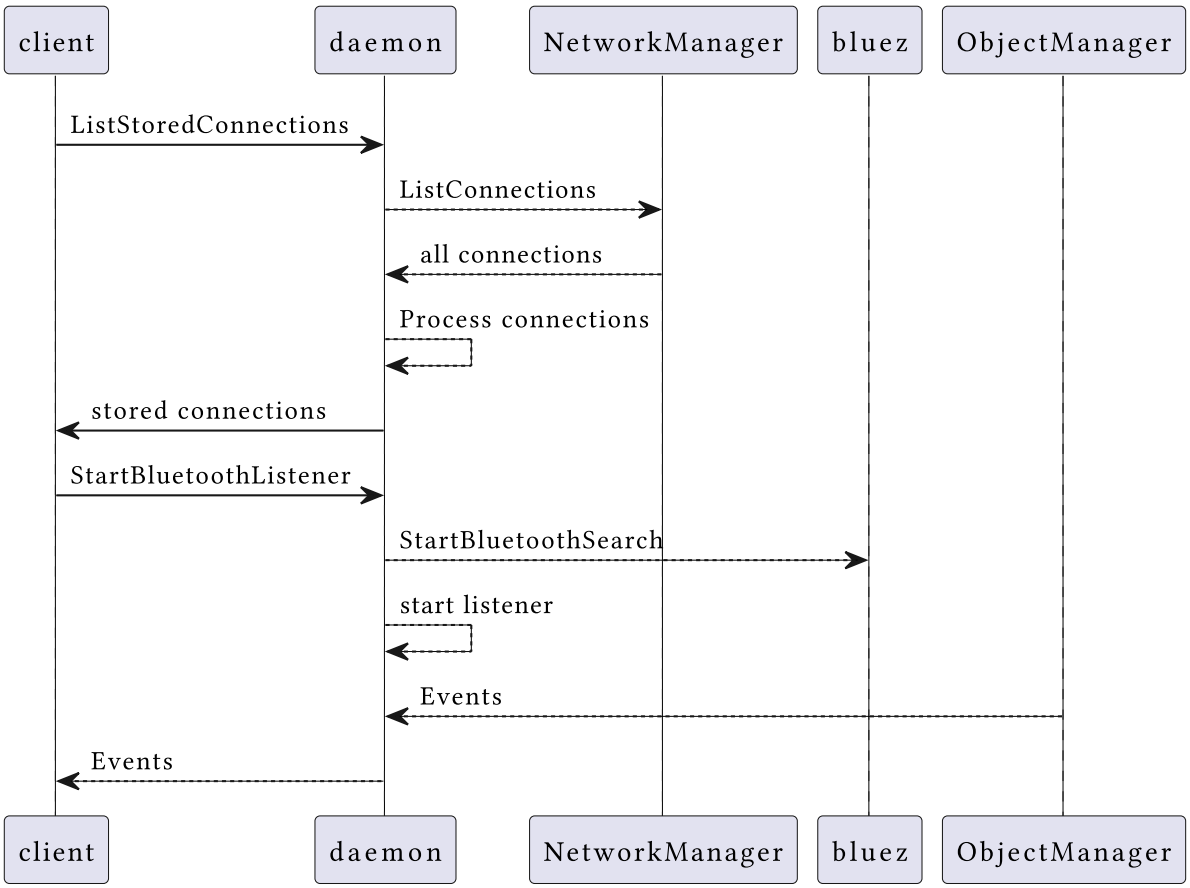


Figure 22: DBus sequence diagram of ReSet

As can be seen in figure Figure 22, the client would need to implement a lot of base functionality in order to communicate with multiple DBus sources. It would also mean that any non-DBus source, such as PulseAudio for ReSet, would have to be accessed separately by the client. Limiting communication to only use one DBus namespace hence simplifies the implementation for the client.

The ObjectManager is a DBus object provided by freedesktop and is implementable for every DBus object, it provides managing (fetching/events) of all objects on a namespace.

4.2 Daemon Implementation

This section documents the code for the DBus service part of ReSet.

4.2.1 Data structure References

All relevant data structures for this section can be found in the appendix Section 11.9.

4.2.2 Main Loop

The main loop exposes the DBus interface to other applications and responds to method calls on this API. In order to provide functionality, the main loop requires a different set of data from different functionalities. This data is provided as a context and can be accessed only within the main loop.

```
// omitted creation of data -> type is DaemonData
// insert data and features into context
let token = cross.register("org.Xetibo.ReSet.Daemon", |c| {
   c.method(
       "UnregisterClient",
       ("client_name",),
       ("result",),
       move |_, data: &mut DaemonData, (client_name,): (String,)| {
           data.clients.remove(&client_name);
           Ok((true,))
       },
   );
});
cross.insert(DBUS_PATH, &token, data);

// process events until shutdown
conn.start_receive(
    MatchRule::new_method_call(),
    Box::new(move |msg, conn| {
        cross.handle_message(msg, conn).unwrap();
        true
    }),
);
```

Listing 3: Code snippet from the daemon DBus main loop

In Listing 3, the daemon registers a function in the org.Xetibo.ReSet.Daemon interface which will provide the function “UnregisterClient”. Whenever this function is called the main loop will execute the registered closure that provides the functionality.

This setup alone would suffice for regular request/response functions, however, ReSet is also required to implement events as certain technologies like Bluetooth require them. They are also required to provide instant feedback for a user interface application upon a state change. For this reason, ReSet also offers a listener for each functionality which will be explained in detail in their respective sub-sections.

4.2.3 Audio

As planned in Section 3.2, PulseAudio was used to implement the audio portion of the ReSet daemon. In concrete terms, the library libpulse-binding [45], which wraps the C PulseAudio functions to Rust was used.

Similarly to the main loop, a listener loop is created for PulseAudio which will continue to receive events from both the PulseAudio server and the DBus daemon. PulseAudio events are handled directly by the library listener, which allows for a listener flag for customized event filters, while the daemon events are handled via multi-producer multi-consumer message passing channels provided by the crossbeam library [46].

```rust
// filter for PulseAudio events
let mut mask = InterestMaskSet::empty();
mask.insert(InterestMaskSet::SINK);
mask.insert(InterestMaskSet::SOURCE);
mask.insert(InterestMaskSet::SINK_INPUT);
mask.insert(InterestMaskSet::SOURCE_OUTPUT);
context.borrow_mut().subscribe(mask, |_| {});

// listener loop
pub fn listen_to_messages(&mut self) {
    loop {
        // handle events from the main loop of the daemon
        let message = self.receiver.recv();
        if let Ok(message) = message {
            self.handle_message(message);
        }
    }
}
```

Listing 4: Audio main loop and event subscription

Libpulse-binding features a separate event loop which is created automatically by the context. Within this loop, all PulseAudio events are handled, which allows for separation between user requests, and events.

```rust
// example PulseAudio event
pulse::context::subscribe::Facility::Sink => {
    if operation == Operation::Removed {
        handle_sink_removed(&connection_ref, index);
        return;
    }
    introspector.get_sink_info_by_index(index, move |result| match result {
        ListResult::Item(sink) => {
            handle_sink_events(&connection_sink, Sink::from(sink), operation);
        }
        ListResult::Error => (),
        ListResult::End => (),
    });
}
```

Listing 5: Audio event handling

After the events are processed, they are directly converted into DBus signals which will be propagated to potential clients by the thread-safe reference to the DBus connection of the DBus main loop. This method skips the transfer of data back to the DBus main loop.

```
// Example of DBus event
let msg = Message::signal(
    &Path::from(DBUS_PATH),
    &AUDIO.into(),
    &"OutputStreamRemoved".into(),
)
.append1(index);
let _ = conn.send(msg);
```

Listing 6: Example DBus event

For user requests, two channels are required in order to both receive and send messages. This system is used for every direct request from the daemon and will block the PulseAudio main loop, however, it does not block the daemon main loop. Overall this will result in blocked events being queued up and executed when possible.

```
// example daemon event
// this example is taken from the audio DBus method GetDefaultSource:

// send the request to get the default source
let _ = data.audio_sender.send(AudioRequest::GetDefaultSource);
// wait for the response from the audio listener
let response = data.audio_receiver.recv();
```

Listing 7: Audio event handling

Important to note for audio is that the listener has to be active for the entire lifecycle of the DBus daemon. This is because the listener is used to provide basic features such as listing of all audio devices, hence no functionality would be provided without listener.

4.2.4 Bluetooth

For Bluetooth, no further library was necessary, as bluez, the default Bluetooth module for Linux is accessed via DBus.

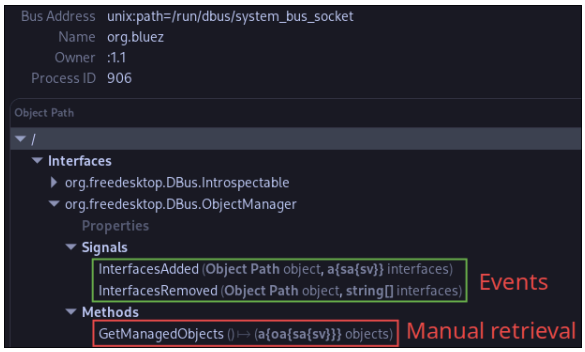
The Bluetooth part is created as a DBus client to bluez that will call DBus methods, and listen for events on various DBus objects.

```
1  // Bluetooth listener as DBus client
2  let bluetooth_device_added =
3      BluetoothDeviceAdded::match_rule(Some(&"org.bluez".into()),
   None).static_clone();
4  let bluetooth_device_removed =
5      BluetoothDeviceRemoved::match_rule(Some(&"org.bluez".into()),
   None).static_clone();
6  let mut bluetooth_device_changed = PropertiesPropertiesChanged::match_rule(
7      Some(&"org.bluez".into()),
8      Some(&path.clone()),
9  )
10 .static_clone();
11 // omitted: handling of each event
```

Listing 8: Bluetooth listener code snippet

The biggest challenge with the bluez interface is the clean access to all needed functionality. For example, fetching all currently available Bluetooth devices has to be done via the ObjectManager object provided by the freedesktop DBus API, while the actual information about these devices is provided by the bluez DBus API.

All DBus objects on bluez can either be accessed via events or via a manual method call:



After fetching the objects, it is now necessary to differentiate between the object types(Adapters and Devices).

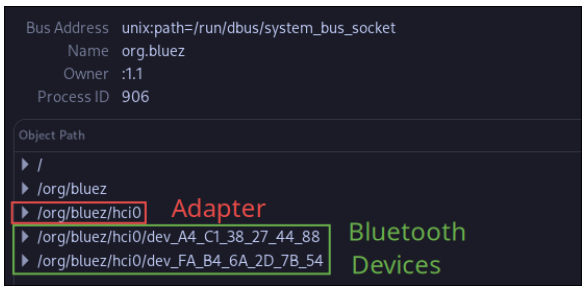


Figure 23: bluez ObjectManager Usage

Within the listener for events, the responses can also be sent to DBus directly with another thread-safe reference to the daemon context as already shown in Listing 6.

4.2.5 Wireless Network

The last piece of functionality, Wi-Fi, is also accessible via DBus.

The challenge with NetworkManager is the amount of features it offers. Besides regular networks, it also offers VPN configuration and usage, as well as other connections, including older protocols. For now, ReSet only offers wireless network configuration, however in the future, this may be expanded upon. The library repository for ReSet (ReSet-Lib [47]) already features entries for both VPN and wired connections.

The base DBus client usage of Wi-Fi is implemented like Bluetooth with an optional listener exposed for clients.

4.2.5.1 Network Manager

ReSet uses six different parts of NetworkManager. Namely, the base NetworkManager, the settings manager which handles existing connections and their properties, the Wi-Fi devices that are used to connect to access points, the access points themselves, and all currently active connections.

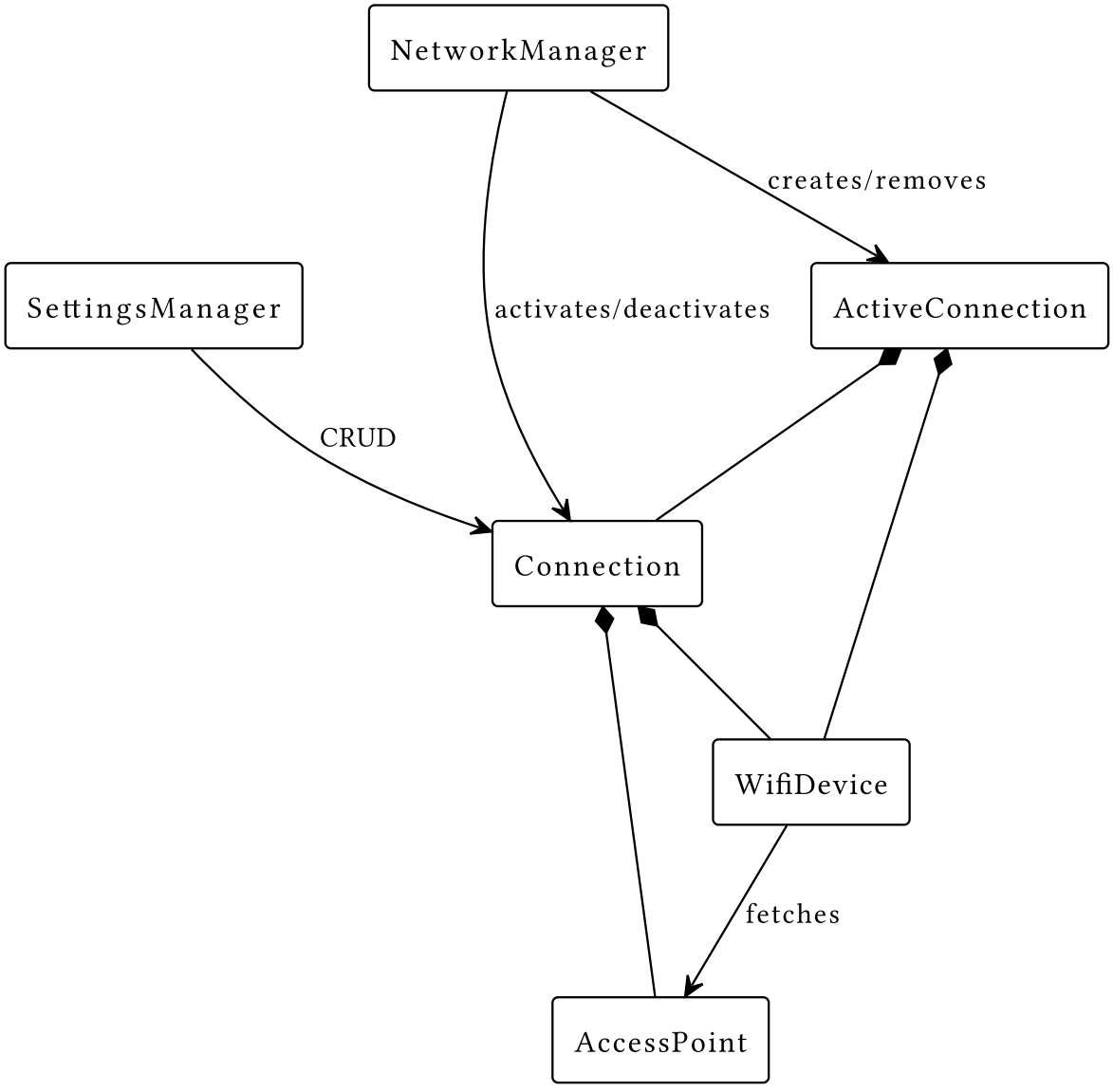


Figure 24: Wi-Fi architecture

All interactions as well as fetching properties of each DBus object have to be done manually via methods, therefore, ReSet abstracts this underlying architecture and only provides two data structures to clients. As an example, the mentioned connection and active connection are both stored within these structures, which removes the need for an additional call by the client.

As mentioned in Section 4.2.1, the structures can be found in Section 11.9.

Special for the WifiDevice struct is the event, originally it was planned to not include this within ReSet, however due to a lack of events on connection changes from any other source, the WifiDeviceChanged event was included to provide the missing functionality.

```
// omitted fetching of current access point
// code slightly changed from source to omit unnecessary code
if let Some(parsed_access_point) = access_point {
    // send new access point
    let msg = Message::signal(
        &Path::from(DBUS_PATH),
        &WIRELESS.into(),
        &"WifiDeviceChanged".into(),
    )
    .append1(WifiDevice {
        path: device.dbus_path.clone(),
        name: device.name.clone(),
        active_access_point: parsed_access_point.ssid,
    });
    let _ = active_access_point_changed_ref.send(msg);
} else {
    // omitted: access point removed, notify client
}
```

Listing 9: WifiDeviceChanged event

A big difference to the PulseAudio and Bluetooth listener is that the majority of the functions can be used without the listener, for PulseAudio, **every** function has to use the listener via the message-passing channel, while for Bluetooth the technology requires a manual request for scans. For clients of ReSet, this means that simpler usage without events is possible as well.

4.3 Frontend Implementation

This section covers the implementation of the ReSet user interface application logic which interacts with the ReSet daemon.

4.3.1 GTK Mainloop

GTK handles its main loop via callbacks, this means users of GTK can define a function for different scenarios like startup or UI building. For ReSet, both a startup and a shutdown function were passed to GTK.

```rust
fn main() {
    let app = Application::builder().application_id(APP_ID).build();

    app.connect_startup(move |_| {
        // libadwaita is a library with predefined GTK modules
        // similar to material UI for various web frameworks
        adw::init().unwrap();
        load_css();
    });

    app.connect_activate(build_ui);
    app.connect_shutdown(shutdown);
    // this will run until user closes the application
    app.run();
}
```

Listing 10: Example GTK application

4.3.2 Modular Design

In order to provide easier implementation for plugins, ReSet was developed with a modular user interface, this can be seen with the consistent design language across the current features. This means that the same container was used for audio, Bluetooth and wireless networks, and can be used by plugins in the future to keep the consistent design.

In Listing 11, the currently used box is shown in the code. The box is technically empty and just serves as a container for each functionality, similar to a custom div component used in web development.

```rust
// The container used for every functionality
#[template(resource = "/org/Xetibo/ReSet/resetSettingBox.ui")]
pub struct SettingBox {}

#[glib::object_subclass]
impl ObjectSubclass for SettingBox {
    const ABSTRACT: bool = false;
    const NAME: &'static str = "resetSettingBox";
    type Type = setting_box::SettingBox;
    type ParentType = gtk::Box;
}
```

Listing 11: ReSet settings box

4.3.3 Listeners

The listeners are DBus clients that activate a callback function on receiving an event. The events themselves always have the same structure with possible events being: added, removed, changed.

```
// Example listener setup(Audio)
// put listener into different thread in order to prevent blocking behavior
gio::spawn_blocking(move || {
    let conn = Connection::new_session().unwrap();
    if listeners.pulse_listener.load(Ordering::SeqCst) {
        // don't start another thread if already running
        return;
    }

    if let Some(sink_box) = sink_box {
        // add events to listen for speakers etc.
        conn = start_output_box_listener(conn, sink_box);
    }
    if let Some(source_box) = source_box {
        // add events to listen for microphones etc.
        conn = start_input_box_listener(conn, source_box);
    }
    listeners.pulse_listener.store(true, Ordering::SeqCst);
    loop {
        // process event -> blocking within thread
        let _ = conn.process(Duration::from_millis(1000));
        if !listeners.pulse_listener.load(Ordering::SeqCst) {
            break;
        }
    }
});
```

Listing 12: Example setup for a DBus listener to the ReSet daemon

It is important to note that the added and changed events provide structs from the DBus API defined in Section 11.8, while removed events provide a DBus path that denotes the removed structure. The reason for this disparity is the inability to fetch data from removed DBus objects, hence only the path is provided.

Regular DBus function calls are handled as described in Listing 2.

4.4 User Interface

In this section, the results of the user interface of ReSet are discussed.

The final UI design has generally followed the intended vision. The UI is clean and simple, and the user can easily navigate through the UI. It is also responsive, which means that it will adjust itself depending on window sizes.

There was one feature that was cut after some discussion which is the breadcrumb menu. It was decided that the breadcrumb menu is not necessary because the UI is not as nested as initially planned. The depth of it is only one level deep at maximum, which means that a breadcrumb bar would be redundant. If the UI ever gets deeply nested, the breadcrumb bar could be reconsidered.

As mentioned in Section 1, ReSet is created with the intent to provide a dynamic user interface that fits into all environments. Specifically standalone environments that currently lack an integrated settings application were the target. As such it was a priority to ensure maximum compatibility with these environments.

Many of such environments are based on window tiling, a concept that ensures the entire screen is used by splitting the screen space with various layout rules. The paper "The anatomy of the modern window manager" explains the concept of tiling in a more detailed manner [48].

For ReSet, the importance of tiling is the behavior of applications in this environment. Concepts such as minimum window size, maximum window size or popups are often a hindrance to tiling. A window will always be placed according to the rules of the layout, and this could mean having as little as a few pixels left to be placed or the entire screen, hence size constraints are incompatible. For popups, the most breaking change is the focus change and the placement of the popup. A tiling window manager has to consider a popup as a different type of window that should not be considered for tiling, and should instead use a traditional stacking approach that allows for overlapping windows. Some tiling window managers simply place the popups at the center of the currently focused monitor, which might be unexpected when trying to open a popup manually, while expected for a password prompt. The second issue is the focus, a tiling window manager is very keyboard-focused, and can for the most part be used entirely via keyboard shortcuts. A popup requires the user to change the focus to the newly created popup and therefore takes away the focus of the last window, a behavior that is not welcome in every situation.

For ReSet, the solution was to provide a nearly size-agnostic application that refrains from using popups wherever possible. In order to ensure the application is usable with any size, each functionality of ReSet is put into a dynamically allocated box. This allows not only the size-agnostic design but also provides the mentioned responsive design by changing from horizontal to vertical orientations. Implemented into ReSet are therefore three different stages, vertical orientation without a sidebar, vertical orientation with a sidebar and horizontal orientation with a sidebar. The stages are shown from minimum size to maximum size respectively, in Figure 25, Figure 26 and Figure 27, each stage is shown visually.

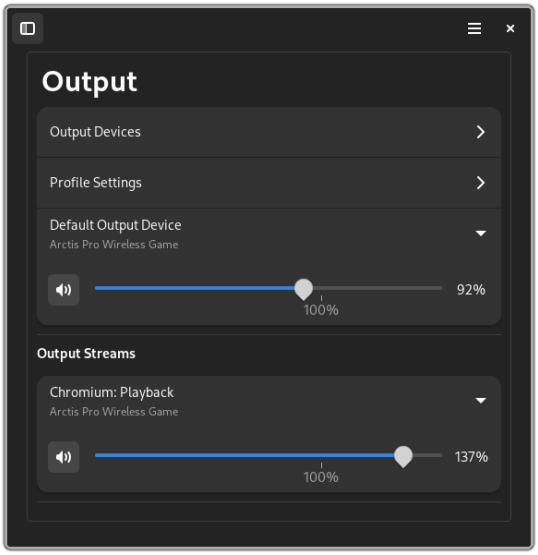


Figure 25: ReSet in vertical view without sidebar

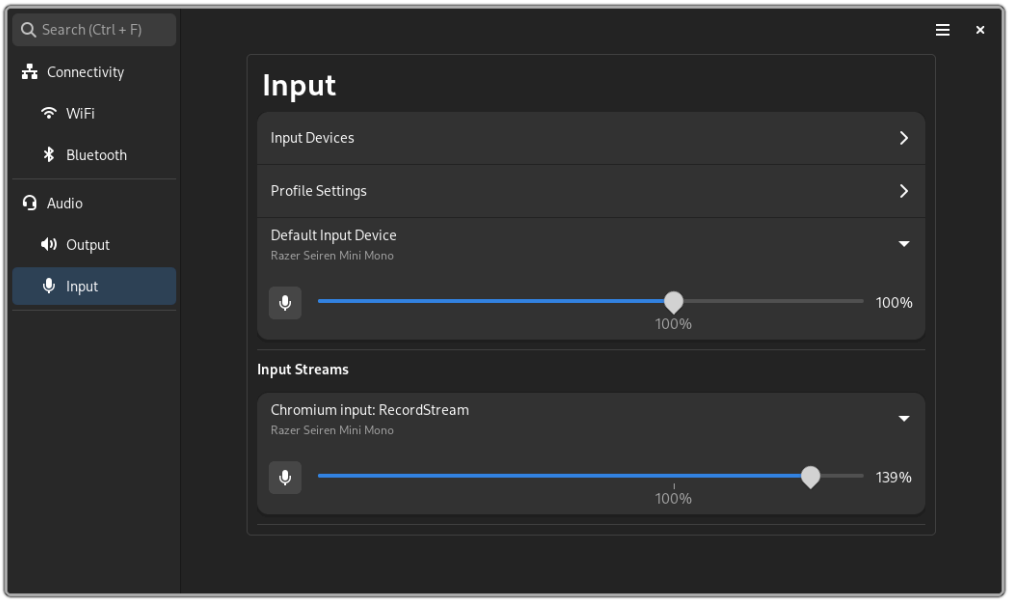


Figure 26: ReSet in vertical view

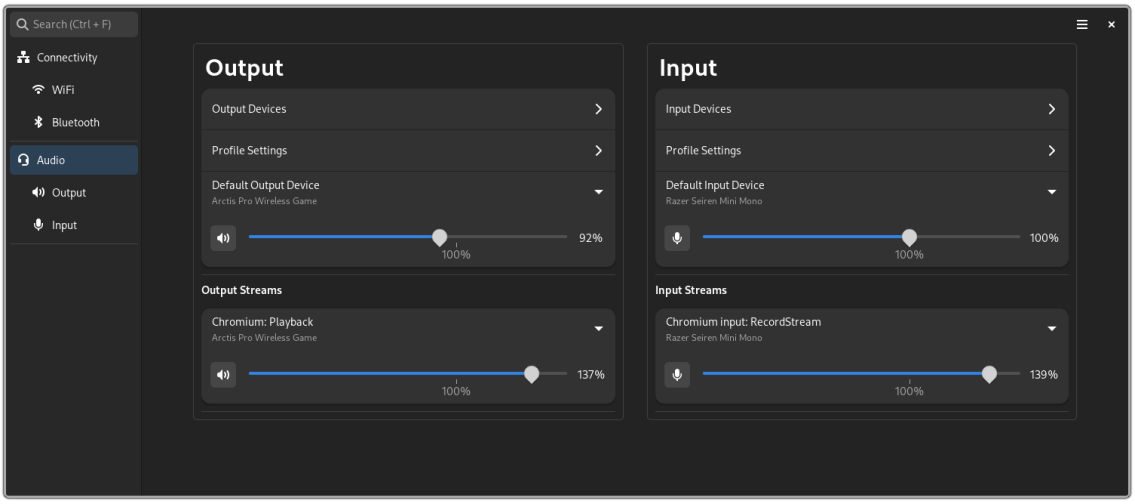


Figure 27: ReSet in horizontal view

To prevent using popups or relying on hamburger menus, ReSet opted to provide advanced configuration of each functionality using the AdwNavigationPage provided by libadwaita [42]. This module allows for a seamless transition from a parent to a child window by clicking the button with the target page name. In Figure 28 and Figure 29, the AdwNavigationPage can be seen in use.

4.4.1 Audio User Interface

For audio, ReSet intends to provide as much relevant information to the user as possible. The intention is to provide not only the central audio settings but also provide adjustments for all currently open programs utilizing audio.

In Figure 28, the user interface can be seen with both the default output device and active programs utilizing audio output.

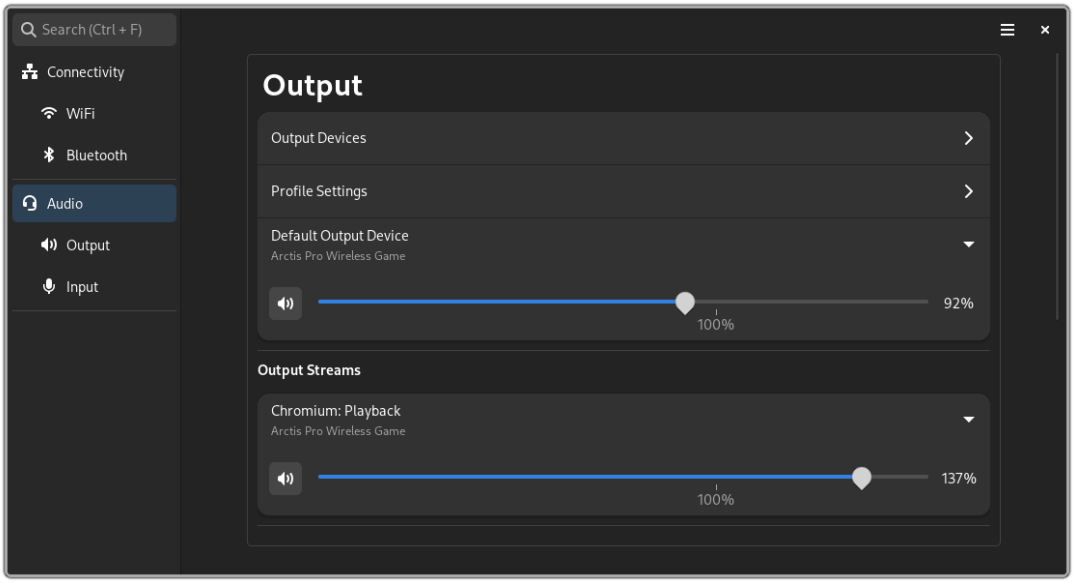


Figure 28: ReSet Audio section

Additional configuration, such as audio profiles and per-device adjustment are available in the device and profile settings respectively.

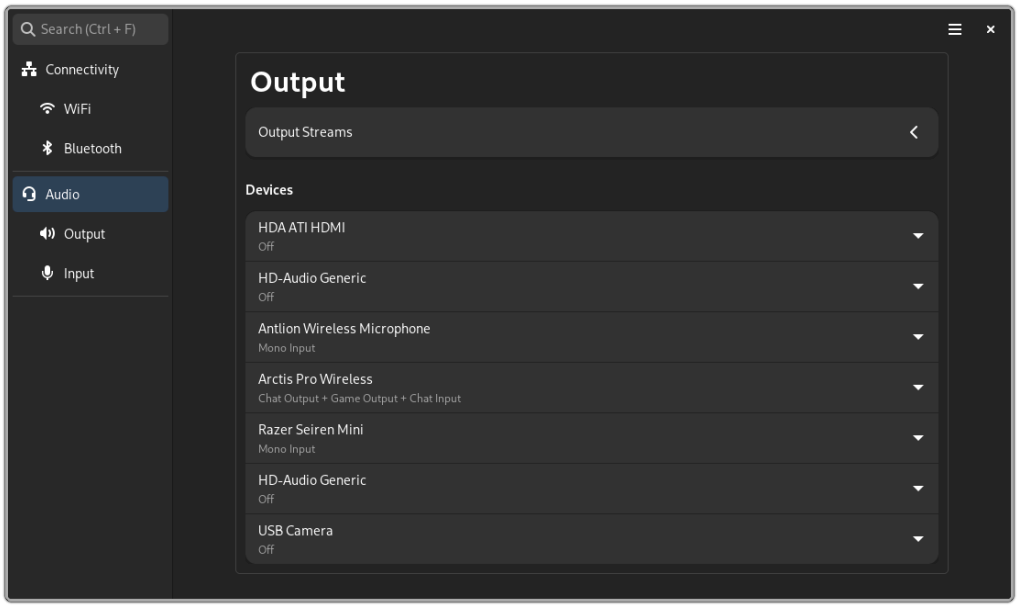


Figure 29: ReSet Audio profiles section

4.4.2 Wi-Fi User Interface

The Wi-Fi settings provide a short general adjustment at the top, using a global switch to enable or disable Wi-Fi in general, while the other entries are changing Wi-Fi adapters or adjusting stored Wi-Fi connections respectively.

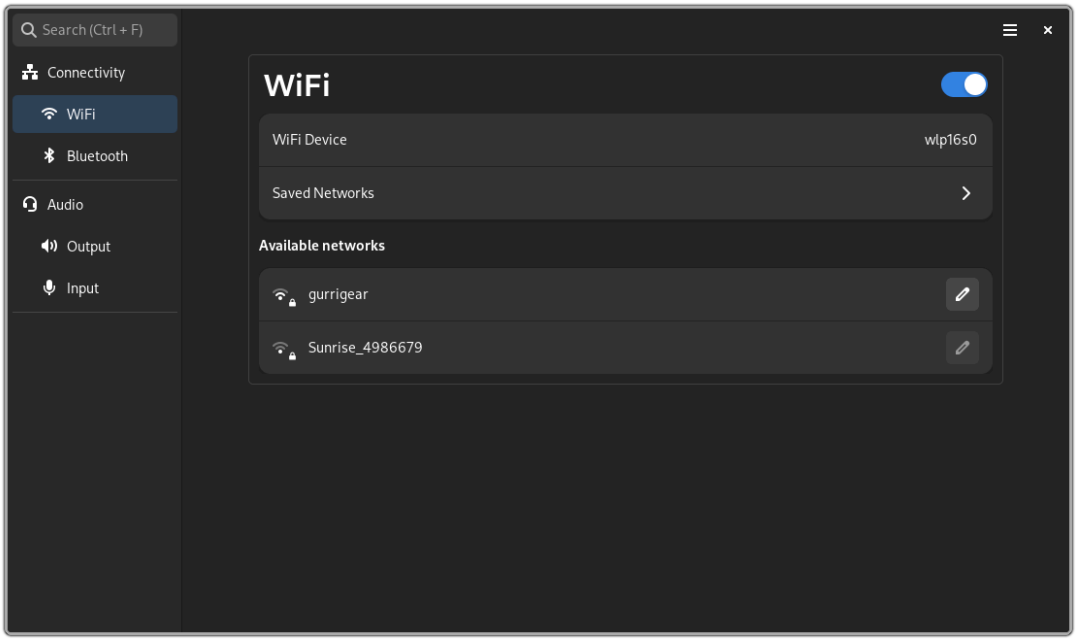


Figure 30: ReSet Wi-Fi section

The access points themselves are shown in a continuous list, using the same module as implemented in the audio section.

For adjustments to connections, ReSet offers general configuration, basic IPV4 and IPV6 configuration (not complete) and basic security settings. The settings page is created with individual tabs for each setting, providing a seamless transition for each functionality. In Figure 31, the security configuration is visualized.

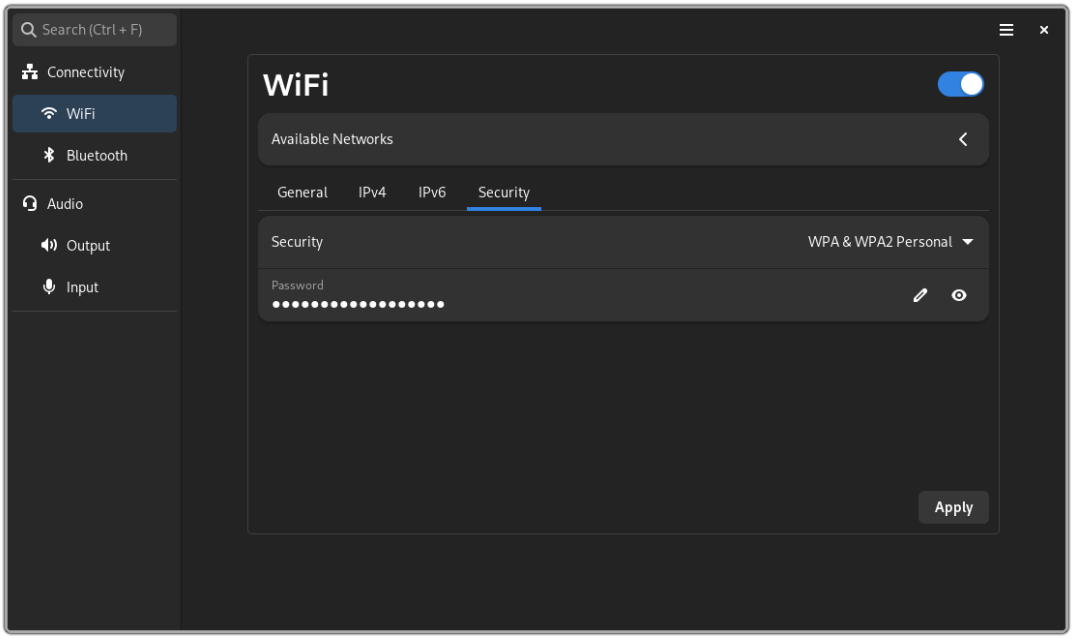


Figure 31: ReSet Wi-Fi security section

4.4.3 Bluetooth User Interface

Bluetooth has the same setup as Wi-Fi, with the only difference being the differentiation between connected and available devices.

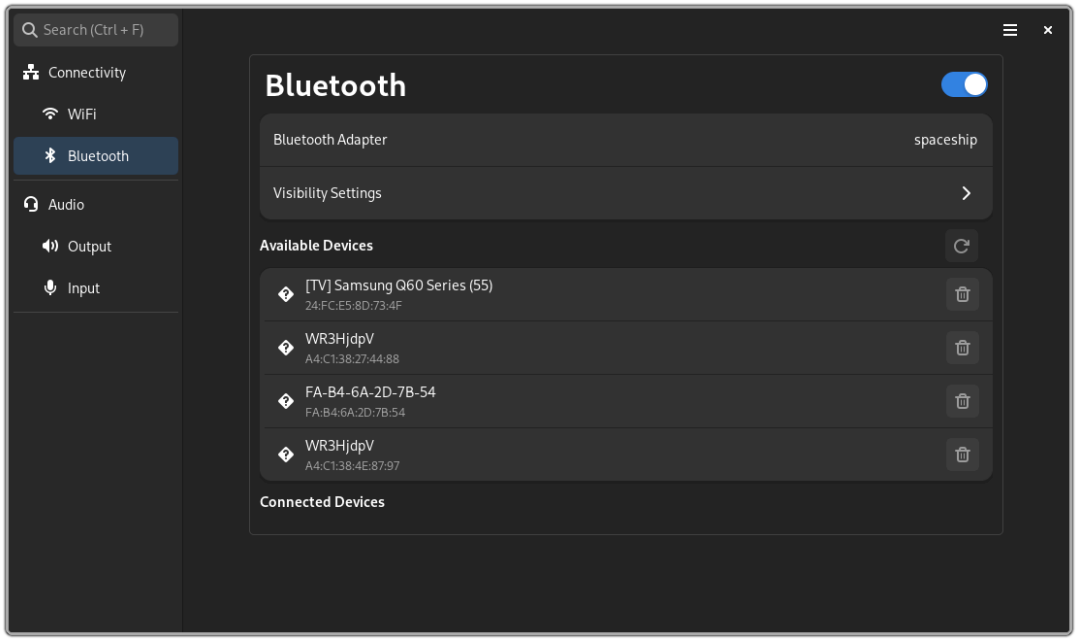


Figure 32: ReSet Bluetooth section

As mentioned in Section 2.3 ReSet intends to be as consistent as possible, which is why the same style principles were applied to all pages within ReSet. This ensures that users who can use parts of ReSet will also be able to use other parts without additional knowledge about the application.

4.4.4 Sidebar and Menu

The sidebar offers simple navigation by click and has a prominent search bar which can also be accessed with a shortcut. Future entries can be created using the sidebar entry developed for ReSet, with the functionality being handled by a callback function.

```
// callback for sidebar  Wi-Fi click
pub const HANDLE_WIFI_CLICK: fn(Arc<Listeners>, FlowBox, Rc<RefCell<Position>>)
=
    |listeners: Arc<Listeners>, reset_main: FlowBox, position:
Rc<RefCell<Position>>| {
    // omitted more setup
        let wifi_box = WifiBox::new(listeners.clone());
        start_event_listener(listeners, wifi_box.clone());
        show_stored_connections(wifi_box.clone());
        scan_for_wifi(wifi_box.clone());
    // omitted more setup
    };

// template
pub struct SidebarEntry {
    // omitted other fields
    pub on_click_event: RefCell<SidebarAction>,
}
```

Listing 13: ReSet sidebar entry

The Menu on the top right is a standard GTK menu, providing a consistent experience with other GTK applications.

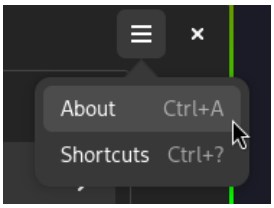


Figure 33: the ReSet popover menu

4.4.5 Template Binding

The user interface is built using the GTK-compatible XML markup language. This allows for a separate user interface definition which reduces code size and keeps functionality away from design. Alongside this, XML also allows for use of graphical tools in order to create the user interface itself, providing immediate feedback about the style and feel of the design.

The binding of these definitions is handled via a GTK-specific build.rs file, which defines GResources that will be integrated into the project. GResources represent a variety of tools and objects which can be used within the project, such as user interface templates, icons, images and more [49].

```
fn main() {
    glib_build_tools::compile_resources(
        &["src/resources"],
        "src/resources/resources.gresource.xml",
        "src.templates.gresource",
    );
    // more resources omitted
}
```

Listing 14: Example GResources

For ReSet, the vast majority of the user interface is handled via XML definitions, with only a fraction being inlined in the code. These inline definitions are usually temporary containers or need a more dynamic functionality that is not directly available with XML.

The binding for XML to Rust happens within regular Rust files, with various GTK macros being used. This means that one can use seemingly regular Rust code, which the gtk-rs library later restructures to GTK-compatible code via Rust macros. Rust macros are metaprogramming, which means the Rust toolchain will change the code before compiling it into a binary. For Rust, this feature expands to allowing full Rust code within metaprogramming, making entire programs possible at compile time [50]. ReSet currently only uses this feature via gtk-rs or default macros provided by the standard library, however, it could become a vital tool for the creation of a plugin system which is mentioned in Section 5.1.1.

```
// an example macro from "The Rust Programming Language"
fn impl_hello_macro(ast: &syn::DeriveInput) -> TokenStream {
    let name = &ast.ident;
    let gen = quote! {
        impl HelloMacro for #name {
            fn hello_macro() {
                println!("Hello, Macro! My name is {}!", stringify!(#name));
            }
        }
    };
    gen.into()
}
```

Listing 15: An example macro from the book: “The Rust Programming Language”[50]

The binding itself is created via the PImpl (Pointer to Implementation) idiom, which allows for a generic implementation of an object without immediate recompilation unless the object itself is changed. As defined by cppreference [51], [52], this idiom is usually applied in C++, but can also be used with Rust, and is also explained by an officially endorsed gtk-rs introduction [53].

```
glib::wrapper! {
    pub struct ListEntry(ObjectSubclass<list_entry_impl::ListEntry>)
    // define GTK classes to inherit from
    @extends gtk::ListBoxRow, gtk::Widget,
    // define interfaces to implement
    @implements gtk::Accessible, gtk::Buildable, gtk::ConstraintTarget,
gtk::Actionable;
}

impl ListEntry {
    pub fn new(child: &impl IsA<Widget>) -> Self {
        Object::builder().build()
    }
}
```

Listing 16: Example Pointer for an XML template

In figure Listing 16, there are two structs with the name ListEntry, the first is the pointer which holds generic GTK functionality, in this case, it will offer everything that the inherited classes provided, as well as the implemented interfaces. The second ListEntry is the implementation which will refer to the XML template visualized in Listing 17.

```
#[derive(Default, CompositeTemplate)]
#[template(resource = "/org/Xetibo/ReSet/resetListBoxRow.ui")]
pub struct ListEntry {
    #[template_child]
    // custom entries that will be included in the template
    pub example: TemplateChild<GTKTYPE>,
}

#[glib::object_subclass]
impl ObjectSubclass for ListEntry {
    const ABSTRACT: bool = false;
    // the name of the created GTK module
    const NAME: &'static str = "resetListBoxRow";
    // the pointer type
    type Type = list_entry::ListEntry;
    type ParentType = gtk::ListBoxRow;
}

// omitted more required setup
```

Listing 17: Example Implementation for an XML template

5 Conclusion

To summarize, ReSet achieved the vision of a universal Settings application. It offers basic usage of Bluetooth and Wi-Fi, as well as extensive audio configuration within one application without bundling with environment-specific software.

All functionality is offered using inter-process communication as planned and other applications can use the daemon service to utilize the API.

In terms of the user interface, user testing Section 11.5.3 has shown positive feedback, allowing ReSet to continue with the current design language.

5.1 Not Implemented Features

This section covers features which were not implemented due to time constraints, future work can potentially resolve these missing features.

5.1.1 Plugin System

This project will not cover a plugin system. However, with the implementation currently provided, it should be possible to implement this without major issues. For example, the current DBus daemon allows for a trivial addition of an additional API and its namespace.

```
// Example API addition
unsafe {
    // using the libloading crate
    let lib = libloading::Library::new("/path/to/liblibrary.so").unwrap();
    let interface: libloading::Symbol<unsafe extern fn() -> u32> =
        lib.get(b"interface").unwrap();
    let name: libloading::Symbol<&'static str> = lib.get(b"name").unwrap();
    feature_strings.push(name);
    features.push(interface);
}

features.push(setup_base(&mut cross, feature_strings));
cross.insert(DBUS_PATH, &features, data);
```

Listing 18: Example API addition

Each of these entries provides its own DBus namespace which can be used to provide methods and signals to clients. Here ReSet could later add a dynamic list of entries via loading of dynamic library functions instead of a hard-coded one.

5.1.2 Persistent Settings

Due to a lack of settings in general as of now, ReSet does not offer persistent settings storage, although the base code for this feature is implemented in the reset library [47].

5.1.3 Customizable Shortcuts

A common feature for many power users, especially on Linux is the heavy use of keyboard shortcuts. ReSet intended to provide customizable shortcuts in order to cater to as many users as possible. This however was also not implemented due to time limitations and is hence a possible improvement.

5.1.4 Further Network Features

Currently, ReSet only offers basic Wi-Fi settings without any wired or VPN support. Alongside this, ReSet also only offers a single security option at this time, WPAPSK, which is effectively a simple password without any certificate support. Basic code to ease the support for all of this exists in ReSet, however it is not functional at this point.

5.1.5 Further Bluetooth Features

For Bluetooth, it would still be possible to implement advanced features such as a pairing wizard for devices that require a pairing code, per device configuration, as well as making file transfers possible via ReSet.

5.1.6 Configurable Home Page

The default starting page for ReSet was supposed to be configurable. For now the Audio output page is set as default. The usage of anonymous functions should ease the integration of this feature.

5.2 Shortcomings

This section covers parts of ReSet that require work to improve existing features.

5.2.1 User Feedback

While technically not implemented due to time constraints, ReSet considers this a shortcoming and not just a missing feature. As of now, ReSet does not offer the user valuable feedback for every action, with some errors being ignored.

Another aspect is the lack of tooltips and the suboptimal keyboard usage within ReSet.

5.2.2 Testing

Testing with ReSet outside of manual integration tests is complicated, this also means that only a few tests were written and none of these can be run on a system without the functionality for the settings ReSet provides. In this case, a proper mock system needs to be built that can be used to test both the daemon and the application without needing real hardware functionality.

A mock system is also not perfect, as it needs to replicate the hardware and driver behavior in order to provide meaningful tests. During this project all team members had to first study all these features and how they behave with their driver and hardware implementations. This means a mock system from the start would not have been possible.

5.2.3 Code Abstraction

With both events and manual functions often providing similar but not the same functionality, some code is currently needlessly duplicated. This code should be abstracted further for better maintainability and readability.

```rust
// Example problematic code
pub fn populate_sources(input_box: Arc<SourceBox>) {
    gio::spawn_blocking(move || {
        let sources = get_sources();
    // ... omitted code
  });
}
// same code in SinkBox
pub fn populate_sinks(output_box: Arc<SinkBox>) {
    gio::spawn_blocking(move || {
        let sinks = get_sinks();
    // ... omitted code
  });
}
```

Listing 19: Example of problematic code to de-duplicate

The challenge here is to ensure that ReSet does not compromise on performance with too many thread-safe references, and while still providing a clean abstraction. This is especially hard with GTK subclassing, which does not always work perfectly with Rust features.

5.2.4 UI Code Duplication

This is a debatable shortcoming. The UI files generated from Cambalache contain several duplications, which makes it more difficult to maintain, as changes have to be made in multiple places. However, the advantage of this is that the amount of boilerplate code in ReSet can be reduced. There could be an argument made that most of these difficulties arise from margin inconsistencies, which could be fixed by extracting those to a CSS class instead of hardcoding.

6 Glossary

Compositor A combination of display server(not a true server with wayland) and window composition system. Often used to describe standalone wayland environments like Mutter [54], KWin [55], Hyprland [56], Sway [57] and River [58].

Daemon Background process, most commonly used to handle functionality for a frontend.

DBus Low-level API providing inter-process communication (IPC) on UNIX operating systems.

Desktop Environment A collection of software, enabling a graphical user interface experience to do general computing tasks.
This includes most basic functionality like starting programs, shutting down the PC or similar.

DTO Data Transfer Object This represents data that can be sent over inter-process communication pipelines such as web requests, DBus or sockets. Data in this state can only be represented as text (usually JSON, TOML, etc.), this means that the endpoint needs to recreate a data structure for a programming language.

Gnome A Linux desktop environment.

KDesktop Environment (KDE) A Linux desktop environment.
The K has no particular meaning.

Language Server Protocol (LSP) Protocol designed to help program software by providing quick fixes to errors, linting, formatting and refactoring.

Minimal Environment A window manager or compositor without the usually bundled applications expected from a full desktop environment. While feature amount vary, usually only window management and low-level configuration are provided out of the box.

Shell Component This refers to an application that integrates into the compositor, it differs to windows in layering, meaning shell components cannot be dragged around, and they can either be drawn beneath all windows (desktop widgets) or on top of windows (overlays).

Status Bar A shell component that usually offers information such as open programs, time, battery and more.
Can be compared to the top bar on MacOS or the taskbar on Windows.

Target Triple String used to define compilation target platforms in Rust.

Wayland The current display protocol used on Linux.
It replaces the previous X11 protocol, which is no longer in development. (it is still maintained for security reasons)

Window Manager Provides window management without compositing them. With the X11 protocol, implementing an entire display server is not needed, therefore one can choose to simply provide, window spawning and window management. Examples include Mutter [54], KWin [55], dwm [59], Xmonad [60], i3 [61] and herbstluftwm [62].

X11 A network transparent windowing system used by a variety of systems. It is usually used with the reference implementation Xorg.

7 Bibliography

[1] feathericons, *settings.svg*. [Online]. Available: https://github.com/feathericons/feather/

[2] OST, *OST.svg*. [Online]. Available: https://www.ost.ch/de/die-ost/organisation/medien

[3] "wayland documentation". [Online]. Available: https://wayland.freedesktop.org/architecture.html

[4] GNOME, "gnome-control-center". [Online]. Available: https://gitlab.gnome.org/GNOME/gnome-control-center

[5] GTK, "GTK". [Online]. Available: https://www.gtk.org/

[6] GNOME, "GNOME Human Interface Guidelines". [Online]. Available: https://developer.gnome.org/hig/

[7] GNOME, "Dconf". [Online]. Available: https://gitlab.gnome.org/GNOME/dconf

[8] KDE, "SystemSettings". [Online]. Available: https://invent.kde.org/plasma/systemsettings

[9] QT, "QT". [Online]. Available: https://www.qt.io/

[10] KDE, "KCM Introduction". [Online]. Available: https://develop.kde.org/docs/features/configuration/kcm/#introduction

[11] pulseaudio, "pavucontrol". [Online]. Available: https://gitlab.freedesktop.org/pulseaudio/pavucontrol

[12] blueman-project, "Blueman". [Online]. Available: https://github.com/blueman-project/blueman

[13] NetworkManager(freedesktop.org), "NetworkManager". [Online]. Available: https://gitlab.freedesktop.org/NetworkManager/NetworkManager

[14] GNOME, "network-manager-applet". [Online]. Available: https://gitlab.gnome.org/GNOME/network-manager-applet

[15] Steve Krug, *Don't Make Me Think (Revisited)*.

[16] Ben Shneiderman and Catherine Plaisant, *Designing the User Interface*.

[17] GNOME, "GNOME Human Interface Guidelines Menus". [Online]. Available: https://developer.gnome.org/hig/patterns/controls/menus.html

[18] "Revamp "New Document" submenu". [Online]. Available: https://gitlab.gnome.org/GNOME/nautilus/-/issues/2205

[19] "Reserve the "New Document" submenu even when there's no templates available". [Online]. Available: https://gitlab.gnome.org/GNOME/nautilus/-/issues/407

[20] "Is possible add shortcut with accels use backspace key for win.up?". [Online]. Available: https://gitlab.gnome.org/GNOME/nautilus/-/issues/2545

[21] "Defining custom accels". [Online]. Available: https://www.reddit.com/r/GTK/comments/16lsdyg/defining_custom_accels/

[22] David Thomas and Andrew Hunt, *The Pragmatic Programmer*, 20th Anniversary Edition.

[23] Mario Ortiz Manero, "Plugins in Rust: The Technologies". [Online]. Available: https://nullderef.com/blog/plugin-tech/

[24] Neovim, "Noevim Lua Documenation". [Online]. Available: https://neovim.io/doc/user/lua.html

[25] Kirottu, "anyrun". [Online]. Available: https://github.com/Kirottu/anyrun

[26] "First class Linux support developed by Microsoft". [Online]. Available: https://github.com/dotnet/maui/discussions/339

[27] "Rust VS Javascript benchmarks". [Online]. Available: https://programming-language-benchmarks.vercel.app/rust-vs-javascript

[28] Maynak Choubey, "Node.js vs Rust: Performance comparison for Hello World case". [Online]. Available: https://medium.com/deno-the-complete-reference/node-js-vs-rust-performance-comparison-for-hello-world-case-2b548fb77c8c

[29] "Rust types". [Online]. Available: https://doc.rust-lang.org/book/ch03-02-data-types.html

[30] "Rust tools". [Online]. Available: https://www.rust-lang.org/tools

[31] "Rust Repository". [Online]. Available: https://github.com/rust-lang/rust

[32] "Microsoft .NET Core Debugger licensing and Microsoft Visual Studio Code". [Online]. Available: https://github.com/dotnet/vscode-csharp/wiki/Microsoft-.NET-Core-Debugger-licensing-and-Microsoft-Visual-Studio-Code

[33] "Kindly reconsider the licensing for .NET Core debugging libraries". [Online]. Available: https://github.com/dotnet/core/issues/505

[34] "C++ compiler support". [Online]. Available: https://en.cppreference.com/w/cpp/compiler_support#C.2B.2B20_library_features

[35] "gtk-rs". [Online]. Available: https://gtk-rs.org/

[36] iced, "iced". [Online]. Available: https://iced.rs/

[37] QT, "QML". [Online]. Available: https://doc.qt.io/qt-6/qtqml-index.html

[38] pulseaudio, "PulseAudio". [Online]. Available: https://gitlab.freedesktop.org/pulseaudio/pulseaudio

[39] bluez, "bluez". [Online]. Available: https://git.kernel.org/pub/scm/bluetooth/bluez.git

[40] GNOME, "Glade". [Online]. Available: https://gitlab.gnome.org/GNOME/glade

[41] Juan Pablo Ugarte, "Cambalache". [Online]. Available: https://gitlab.gnome.org/jpu/cambalache

[42] GNOME, "libadwaita-rs". [Online]. Available: https://gitlab.gnome.org/World/Rust/libadwaita-rs

[43] Xetibo, "ReSet-Dameon API". [Online]. Available: https://docs.rs/reset_daemon/0.5.8/reset_daemon/api/API/index.html

[44] freedesktop, "dbus". [Online]. Available: https://www.freedesktop.org/wiki/Software/dbus/

[45] jnqnfe, "pulse-binding-Rust". [Online]. Available: https://github.com/jnqnfe/pulse-binding-rust

[46] crossbeam-rs, "crossbeam". [Online]. Available: https://github.com/crossbeam-rs/crossbeam

[47] Xetibo, "ReSet-Lib". [Online]. Available: https://github.com/Xetibo/ReSet-Lib

[48] Max van Duerzen, "The anatomy of the modern window manager". [Online]. Available: http://www.cs.ru.nl/bachelors-theses/2019/Max_van_Deurzen___4581903___The_anatomy_of_the_modern_window_manager_-_a_case_study_for_X_in_an_Agile_manner.pdf

[49] GNOME, “GNOME Gresources documentation”. [Online]. Available: https://developer-old.gnome.org/gio/stable/GResource.html

[50] w. c. f. t. R. C. Steve Klabnik and Carol Nichols, “The Rust Programming Language”. [Online]. Available: https://doc.rust-lang.org/book/ch19-06-macros.html

[51] cppreference, “PImpl”. [Online]. Available: https://en.cppreference.com/w/cpp/language/pimpl

[52] Herb Sutter, “The Fast PImpl idiom”. [Online]. Available: http://www.gotw.ca/gotw/028.htm

[53] Julian Hofer, “GUI development with Rust and GTK 4”. [Online]. Available: https://gtk-rs.org/gtk4-rs/stable/latest/book/

[54] GNOME, “mutter”. [Online]. Available: https://gitlab.gnome.org/GNOME/mutter

[55] KDE, “kwin”. [Online]. Available: https://invent.kde.org/plasma/kwin

[56] Hyprland, “Hyprland”. [Online]. Available: https://hyprland.org/

[57] Sway, “Sway”. [Online]. Available: https://swaywm.org/

[58] river, “river”. [Online]. Available: https://github.com/riverwm/river

[59] suckless, “dwm”. [Online]. Available: https://dwm.suckless.org/

[60] xmonad, “xmonad”. [Online]. Available: https://xmonad.org/

[61] Michael Stapelberg, “i3”. [Online]. Available: https://i3wm.org/

[62] herbstluftwm, “herbstluftwm”. [Online]. Available: https://herbstluftwm.org/

[63] “ReSet Issues”. [Online]. Available: https://github.com/Xetibo/ReSet/issues

[64] “ReSet Project Board”. [Online]. Available: https://github.com/orgs/Xetibo/projects/1

[65] Github, “About Github-hosted runners”. [Online]. Available: https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners/about-github-hosted-runners

[66] diwic, “dbus-rs”. [Online]. Available: https://github.com/diwic/dbus-rs

[67] gtk-rs, “gtk4-rs”. [Online]. Available: https://github.com/gtk-rs/gtk4-rs

[68] gtk-rs, “gtk-rs-core”. [Online]. Available: https://github.com/gtk-rs/gtk-rs-core

[69] tokio-rs, “tokio”. [Online]. Available: https://github.com/tokio-rs/tokio

[70] immortal, “fork”. [Online]. Available: https://github.com/immortal/fork

[71] achanda, “ipnetwork”. [Online]. Available: https://github.com/achanda/ipnetwork

[72] xdg-rs, “dirs”. [Online]. Available: https://github.com/xdg-rs/dirs/tree/master/directories

8 List of Tables

9 List of Figures

10 List of Listings

11 Appendix

11.1 Planning

11.2 Methods

Project Management | ReSet is managed using Scrum.
Due to the small team size, no scrum master or product owner is chosen, the work of these positions is done in collaboration.

11.2.1 User Stories

User stories are handled via GitHub issues [63]. This allows for simple handling of labels and function points, which can also be changed later on if needed.

11.2.2 Sprint Backlog/Product Backlog

Both the sprint backlog and the product backlog are handled using the GitHub project board [64]. This allows for a clean and automatic handling of created user stories via GitHub issues [63].

11.2.3 Tools

Rust is an editor-agnostic language, hence both team members can use their tools without conflicts. For documentation typst is used, which also does not require a special editor, it just requires a compiler that works on all platforms. The user interface is made with Cambalache.

11.3 Time

As an agile methodology is used, long-term time management is only broad guidance and does not necessarily overlap with reality. In other words, with a 2 week sprint, only 2 weeks are considered well planned out. By week 4, work on the backend implementation should be started. The minimal working UI milestone is in week 7, so UI Reviews can be assessed and corrections can be applied early. By week 12 the minimal viable product should be in production state, in order to leave time for reflection and documentation.

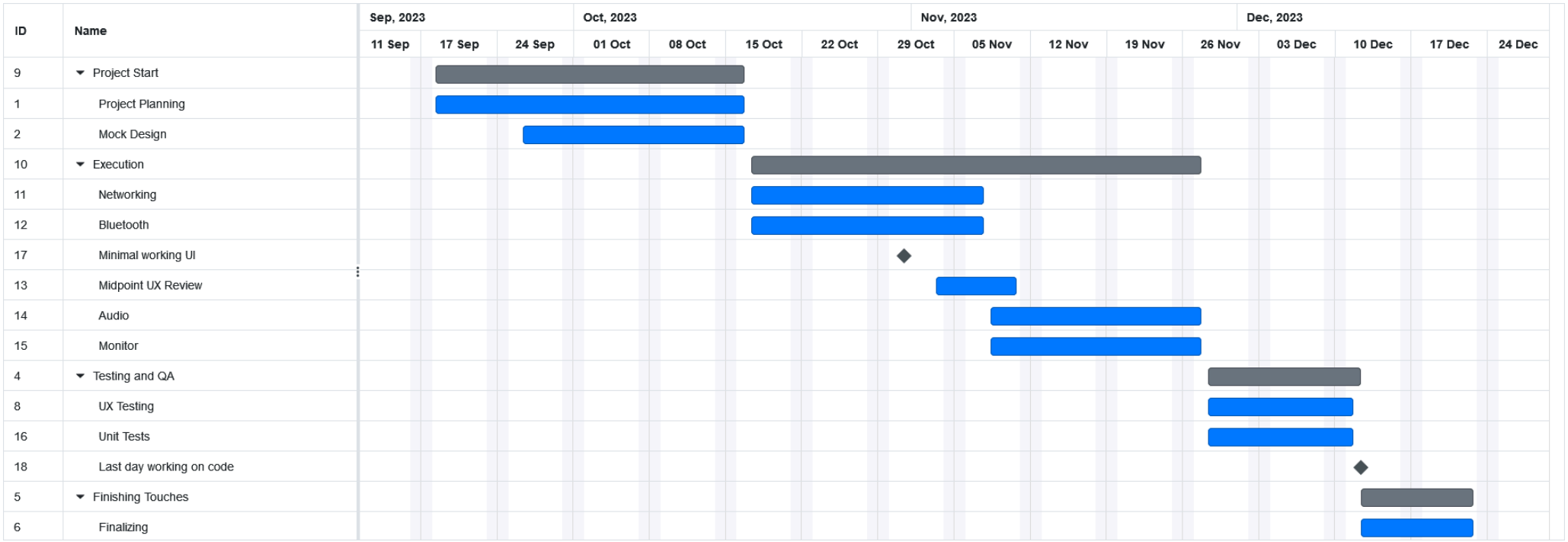


Figure 34: Time management

11.4 Requirements

11.4.1 Functional Requirements

As per Scrum, functional requirements are handled using the user story format which can be found in the ReSet and ReSet-Daemon repositories respectively.

11.4.2 Non-Functional Requirements

Subject	User Interface
Requirement	Usability
Priority	Medium
Description	Loading operations should not block the UI and the user should be informed with a loading icon or similar.
Measures	• Show loading icons or similar during blocking operations. • Restrict actions during the loading period. • Use async functions in order to keep UI responsive.
Subject	User Interface
Requirement	Usability
Priority	Medium
Description	Errors should be visible to the user, and the user should be able to solve issues accordingly. Example: The user has not installed any Bluetooth software, therefore ReSet should inform the user of the situation.
Measures	• Provide adequate error handling in the user interface via popups or similar. • The same functionality should be available in CLI and API. • ReSet should not crash if a dependency is not installed.
Subject	Code Duplication
Requirement	Maintainability
Priority	Low
Description	ReSet should not have the same logic implemented as ReSet-Daemon. In general, ReSet should not have any logic other than necessary.
Measures	• Provide adequate functions over the inter-process connection • Well-documented API

11.5 Risks

Risks are assessed according to the ISO standard with a risk matrix.

Certain	High	High	Very High	Very High
Likely	Medium	High	High	Very High
Possible	Low	Medium	High	Very High
Unlikely	Low	Medium	Medium	High
Rare	Low	Low	Medium	Medium
Eliminated	Eliminated	Eliminated	Eliminated	Eliminated
	Minor	Marginal	Critical	Catastrophic
Probability	Severity			

Subject	Toolkit
Risk	GTK might not be perfectly suitable for the required use case, especially considering the usage of Rust bindings and not C directly.
Priority	Medium
Probability&Severity	Unlikely & Critical
Measures	Iced can be used as a backup toolkit if GTK turns out to not be usable for the project.
Subject	Plugin System
Risk	The plugin system might be too ambitious and could take too much time to realize.
Priority	High
Probability&Severity	Likely & Marginal
Measures	Instead of a plugin system, DBus[[44]] or socket connections can be used to realize a limited implementation of expected features.
Subject	System Interaction
Risk	System feature integration like audio, Bluetooth and more might not be as seamless as planned.
Priority	Low
Probability&Severity	Rare & Marginal
Measures	Potential use of alternative integrations. Example: The standard NetworkManager does not integrate well → use Systemd-networkd instead.

11.5.1 ReSet Tests

This section covers both the Mock user interface tests and the proper user tests.
The data sources for the tests are provided as CSV files bundled with this paper.

11.5.2 Mock UI Test

Reaching the project's midpoint, the UI mock of ReSet was given out to a select few people for testing and reviewing purposes. The idea is the get feedback early so that there is still enough time to change the UI without much effort. Because it is only a UI mock test, it was not possible to ask for feedback for all tests defined in Section 11.5.3 because it has not been implemented yet.

Navigation feedback

The feedback is very positive, especially with the mention of the ability to show all settings or just a specific part. The structure is clean and makes it easy to understand which subcategory belongs to which category.

Readability feedback

There was not much feedback regarding readability, because it is not something that can or should be influenced directly. Font size and contrast are set in the settings, meaning it is consistent throughout the system, which includes ReSet.

Layout and Design feedback

Users were generally positive about the layout and design of the app, but there were also some ideas for improvement. One user noted that the window was padded, which was noticeable when using a white theme. In addition, when opening a category (e.g. connectivity), users noted that each subcategory lacked clear separation and labeling. As far as the layout is concerned, inconsistencies were found in the sizes and margins, which can be attributed to the manual design of each element. Users also requested features such as the ability to collapse subcategories and the inclusion of bold font for categories. Despite these areas for improvement, the overall sentiment towards the app was positive, as users appreciated the design while providing valuable advice on how to improve it.

Conclusion

Feedback around navigation and readability was positive, mostly attributed to the currently simple UI. It is important to note that readability is already configured by the user or by using system defaults. ReSet does not overwrite these configurations to keep a consistent look throughout the user's system. Concerning the layout and design, there will be improvements made by templating common UI building blocks, which will fix the issue of inconsistent design and make it easier to keep it consistent. Features such as collapsing subcategories have to be discussed further, to determine if it is necessary and if it fits in the timeframe.

11.5.3 ReSet User Test

Subject	Search bar and Sidebar
Description	User can filter settings by typing in the search bar. Clicking on a setting will open the corresponding settings box and reset the sidebar.
Feedback	Searching worked for all users.
Suggestions	• Search field is easy to overlook and should be bigger • Highlight search text • Search bar is redundant with few categories

Subject	Dynamic window
Description	Multiple settings can be opened at the same time and each setting box will reflow based on how much space is available.
Feedback	Resizing worked for all users and overall feedback to resizing was positive.
Suggestions	• Search bar hide is not very smooth, maybe add animation • Selected category is not highlighted (fixed) • Default category is arbitrary (Noted in Section 5.1.6)

Subject	Wi-Fi
Description	Users can connect and disconnect from a Wi-Fi network. They should also be able to change the password for an existing Wi-Fi network and remove it.
Feedback	• Changing to a wrong password crashes the program for one user. (un-reproducible) • Another user mentioned that the access points were fetched very slowly compared to nmcli. • Opening Wi-Fi with a computer without Wi-Fi crashes ReSet. (fixed)
Suggestions	• Apply button being clickable with no changes made was confusing • Not enough error messages • Not enough feedback when connecting/connected to Wi-Fi network • Perhaps use gear icon instead of pen icon for Wi-Fi settings • Clicking on Wi-Fi should close the details page • Use different colors for a connected network

Subject	Bluetooth
Description	Users can connect and disconnect Bluetooth devices.
Feedback	Most users were able to connect and disconnect without issues. • Conflict with KDE Bluetooth settings • One user mentioned that connecting or disconnecting worked, however, ReSet needs to be restarted in order to do both with one session. (fixed)
Suggestions	• Move connected devices above available devices (fixed) • Bluetooth search needs an indicator (fixed) • Show progress during connection would be helpful • Paired devices should be shown differently from available ones • It is not clear what the refresh button does • Add an option to hide devices that only offer a MAC address • Tooltips are missing

Subject	Volume of Audio devices
Description	Users can change the volume of their input and output devices.
Feedback	All users were able to change their volume without issues.
Suggestions	• Keyboard support missing (Keyboard input works, but the focus is not intuitive, user was focused on a different UI element) • Dropdowns on the right side are strange • Perhaps put the mute button on the right side

Subject	Change default Audio device
Description	Users can change the default input and output device.
Feedback	Changing the default audio device worked for all users and was intuitive.
Suggestions	

Subject	Audio of individual applications
Description	Users can change the input and output of individual applications. They should also be able to mute an audio source.
Feedback	All users were able to change their volume without issues.
Suggestions	

Subject	Disable Audio devices
Description	Users can disable input and output devices.
Feedback	• Disabling audio devices can cause ReSet to freeze. (fixed) • Removing the last audio devices causes said audio device to still be listed as selected, even though it is disabled.
Suggestions	• Profile configuration was not clear for some users

Subject	Performance
Description	Users should not experience any lags or stuttering.
Feedback	Users reported no performance issues.
Suggestions	

Subject	Intuitive UI
Description	The UI is structured logically so that the user can easily find the setting they are looking for. Settings that belong together are grouped.
Feedback	Some users were unfamiliar with Linux or GTK and were sometimes confused with GTK standard elements. Otherwise, the users reported that ReSet is intuitive and the current user interface makes sense.
Suggestions	• Submenu navigation is unintuitive (return button is across the screen) Only the icon is on the right, the entire row is the button.

Subject	Layout
Description	The layout should be visually appealing and feel organized to the users.
Feedback	Users are overall satisfied with the visuals and the structure of ReSet.
Suggestions	• Some users suggested more color themes Colors of ReSet are handled by GTK, hence changing the GTK color will also change the color of ReSet.

11.5.3.1 General Feedback

Users were overall happy with ReSet, with many issues being small improvements that were mostly overlooked due to time as mentioned in Section 5.2. During testing, some major issues were able to be discovered and fixed, like ReSet crashing without Wi-Fi being present on the computer.

11.6 Documentation

Next to this document, both ReSet and ReSet-Daemon will have their code documented using the Rustdoc functionality. This allows inline documentation, which will later be converted to an HTML file.

The documentation for both the API to interact with ReSet-Daemon and the API for the plugin system will also be handled with this method. It allows for dynamic documentation that can be changed directly inside the code. With cargo test, one can even create documentation tests, making sure that all examples work before releasing a new version.

```rust
/// # Examples
/// takes a number and multiplies it with itself x(positive) amount of times.
/// '''
/// use test_package::pfact;
/// let num = pfact(2,3);
/// assert_eq!(8, num);
/// '''
pub fn pfact(number: i32, exponent: u32) -> i32 {
    if exponent == 0 {
        return 1;
    }
    if exponent == 1 {
        return number;
    }
    let mut result = number;
    for _ in 1..exponent {
        result = result * number;
    }
    result
}
```


```
running 1 test
test src/lib.rs - pfact (line 7) ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.18s
```

Listing 20: Rustdoc example entry for the code above.

11.6.1 Guidelines

ReSet is written in idiomatic Rust where possible(not always applicable due to C and GTK architecture), which means that the project uses the standard linter Rust provides, clippy, alongside the default Rust tools like Rust-analyzer and the default formatter Rustfmt.

11.7 Continous Integration

In this section, the continuous integration workflows for ReSet are discussed.

For ReSet, two different workflows are necessary, the first is a build after pull request and build on main branch push, which was realized using the GitHub workflows. Important to note is that for the ReSet application itself, the self-hosted GitHub runner was used. This is due to the regular GitHub runner only offering older versions of Ubuntu, which makes the use of newer GTK and libadwaita libraries impossible. Using the self-hosted runner, it was possible to provide a build system with an up-to-date version of Ubuntu [65].

```yaml
name: Rust

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

env:
  CARGO_TERM_COLOR: always

jobs:
  build:
    runs-on: [self-hosted, ubuntu]
    steps:
      - uses: actions/checkout@v3
      - name: nightly-rust
        uses: actions-rs/toolchain@v1
        with:
          profile: minimal
          toolchain: nightly
      - name: Build
        run: cargo build --verbose
      - name: Run clippy
        run: cargo clippy --fix
```

Listing 21: ReSet build workflow

For the ReSet frontend, there are currently no tests as of now. GTK has a few tools to test the user interface, but it is not the most extensive documentation, and considering the already limited time for this project, this was not pursued. Usually, the logic of the frontend would be tested instead, but here all functions call DBus functions which means all functionality is implemented in the backend.

On the backend there are initial tests, but those cannot be run on the build system either as they rely on hardware features(Wi-Fi, Bluetooth, audio), which cannot be provided appropriately within a test environment. In Section 5.2.2, a plan for a mock system is explained for future works.

The second workflow for ReSet is more complicated, as it requires different Linux version in order to provide easy packaging of ReSet. For ReSet, five different variations of packaging will be provided: Flatpak, Debian Package, Arch Package, regular binary and the crates.io cargo package. Nearly all of these will offer the same experience, with the only exception being the flatpak package, which cannot be used to run the daemon in a standalone fashion. In order to still provide ReSet, the daemon is built into the ReSet frontend, and will be run for the duration of the application lifetime, should the daemon not already be running.

```rust
async fn daemon_check() {
    let handle = thread::spawn(|| {
        let conn = Connection::new_session().unwrap();
        let proxy = conn.with_proxy(
            BASE,
            DBUS_PATH,
            Duration::from_millis(100),
        );
        let res: Result<(), Error> =
            proxy.method_call(BASE, "RegisterClient", ("ReSet",));
        res
    });
    let res = handle.join();
    if res.unwrap().is_err() {
        // daemon was not running, start the daemon now
        run_daemon().await;
    }
}
```

Listing 22: Daemon check within main.rs

In Listing 23 the Ubuntu runner configuration is shown, which builds the binary, the flatpak and the Debian package.

```yaml
# omitted setup
runs-on: [self-hosted, ubuntu]
steps:
  - uses: actions/checkout@v3
  - name: nightly-rust
    uses: actions-rs/toolchain@v1
    with:
      profile: minimal
      toolchain: nightly
  - name: Build rust package
    run: cargo build --release --verbose
  - name: Build Flatpak
    run: |
      cd flatpak
      ./build.sh
  - name: Build Ubuntu package
    run: |
      cp ./target/release/reset ./debian/.
      dpkg-deb --build debian
      mv debian.deb reset.deb
  - name: Release
    uses: softprops/action-gh-release@v1
    with:
      files: |
        target/release/reset
        flatpak/reset.flatpak
        reset.deb
```

Listing 23: ReSet build workflow on Ubuntu

In Listing 24 the Arch runner configuration is shown, which builds the Arch package.

```yaml
# omitted setup
runs-on: [self-hosted, arch]
steps:
  - uses: actions/checkout@v3
  - name: nightly-rust
    uses: actions-rs/toolchain@v1
    with:
      profile: minimal
      toolchain: nightly
  - name: Build rust package
    run: makepkg PKGBUILD
  - name: Release
    uses: softprops/action-gh-release@v1
    with:
      files: |
        reset-${{github.ref_name}}-0-x86_64.pkg.tar.zst
```

Listing 24: ReSet build workflow on Arch

```
pkgname=reset
pkgver=0.1.1
pkgrel=0
arch=('x86_64')
pkgdir="/usr/bin/${pkgname}"
pkgdesc="A wip universal Linux settings application."
depends=('rust' 'gtk4' 'dbus')

build() {
  cargo build --release
}

package() {
  cd ..
  install -Dm755 target/release/"$pkgname" "$pkgdir"/usr/bin/"$pkgname"
  install -Dm644 "$pkgname.desktop" "$pkgdir/usr/share/applications/
$pkgname.desktop"
  install -Dm644 "src/resources/icons/ReSet.svg" "$pkgdir/usr/share/pixmaps/
ReSet.svg"
}
```

Listing 25: PKGBUILD to build the Arch package

11.8 DBus API

11.8.1 Base API

Table 10: Base DBus Table

DBus interface name	org.Xetibo.ReSet.Daemon

```
// Returns all capabilities of the daemon as strings
fn GetCapabilities() -> Vec<String>;
//
// Register the client to the daemon.
// This is mainly useful for clients that want to ensure the daemon is running before
// starting calls.
// Later on this can be expanded for more functionality.
fn RegisterClient(client_name: String) -> bool;
//
// Deletes the entry for this client from the daemon.
fn UnregisterClient(client_name: String) -> bool;
//
// Shuts down the daemon.
fn Shutdown();
```

Listing 26: Base DBus API for the ReSet daemon

11.8.2 Wi-Fi API

Table 11: Wi-FI DBus Table

Type	Type Description
AccessPoint	DBus signature: ayyoob `Vec<u8>, u8, Path<'static>, Path<'static>, bool`
WifiDevice	DBus signature: osay `Path<'static>,String, Vec<u8>`

Event
AccessPointChanged -> AccessPoint
AccessPointAdded -> AccessPoint
AccessPointRemoved -> Path<'static>
WifiDeviceChanged -> WifiDevice
ResetWifiDevices -> Vec<WifiDevices>

DBus interface name	org.Xetibo.ReSet.Wireless

```
// Returns all access points for the current wireless network device.
fn ListAccessPoints() -> Vec<AccessPoint>;
//
// A check that returns the current status of Wifi.
// Returns a bool as a result of the operation.
fn GetWifiStatus() -> bool;
//
// Enables or disables Wifi for the entire system.
fn SetWifiEnabled(enabled: bool) -> bool;
//
// Returns the dbus path of the current wireless network device, as well as the name.
fn GetCurrentWifiDevice() -> WifiDevice;
//
// Returns all available wireless network devices.
fn GetAllWifiDevices() -> Vec<WifiDevice>;
//
// Sets the current network device based on the dbus path of the device.
// Returns true on success and false on error.
fn SetWifiDevice(device: Path<'static>) -> bool;
//
// Connects to an access point that has a known connection inside the NetworkManager.
// Note, for a new access point, use the ConnectToNewAccessPoint function.
// Returns true on success and false on error.
fn ConnectToKnownAccessPoint(access_point: AccessPoint) -> bool;
//
// Connects to a new access point with a password.
// Returns true on success and false on error.
fn ConnectToNewKnownAccessPoint(access_point: AccessPoint, password: String) -> bool;
//
// Disconnects from the currently connected access point.
// Calling this without a connected access point will return false.
// Returns true on success and false on error.
fn DisconnectFromCurrentAccessPoint() -> bool;
//
// Returns the stored connections for the currently selected wireless device from
NetworkManager.
// Returns dbus invalid arguments on error.
fn ListStoredConnections() -> Vec<(Path<'static>, Vec<u8>)>;
//
// Returns the settings of a connection.
// Can be used in combination with the Connection struct in order to provide easy
serialization
// and deserialization from and to this hashmap.
// Returns dbus invalid arguments on error.
fn GetConnectionSettings(path: Path<'static>) -> HashMap<String, PropMap>;
//
// Sets the settings of a connection.
// Can be used in combination with the Connection struct in order to provide easy
serialization
// and deserialization from and to this hashmap.
// Returns true on success and false on error.
fn SetConnectionSettings(path: Path<'static>, settings: HashMap<String, PropMap>) -
> bool;
```

Listing 27: Wifi API for the ReSet daemon (part1)

```
// Deletes the stored connection given the dbus path.
// Returns true on success and false on error.
fn DeleteConnection(path: Path<'static>) -> bool;
//
// Starts the wireless network listener which provides dbus events on access points
and the
// wireless device.
// Repeatedly starting the network listener twice will simply return an error on
consecutive
// tries.
// Returns true on success and false on error.
fn StartNetworkListener() -> bool;
//
// Stops the wireless network listener.
// Returns true on success and false on error.
fn StopNetworkListener() -> bool;
```

Listing 28: Wifi API for the ReSet daemon (part2)

11.8.3 Bluetooth API

Table 12: Bluetooth DBus Table

Type	Type Description
BluetoothDevice	**DBus signature: onssobbbbbss** `Path<'static>, u16, String, String, Path<'static>, bool, bool, bool, bool, bool, String, String`
BluetoothAdapter	**DBus signature: osbbb** `Path<'static>, String, bool, bool, bool`

Event
BluetoothDeviceAdded -> BluetoothDevice
BluetoothDeviceRemoved -> Path<'static>
BluetoothDeviceChanged -> BluetoothDevice

DBus interface name	**org.Xetibo.ReSet.Bluetooth**

```
// Starts searching for Bluetooth devices.
// Note this is without a listener, you would have to manually request Bluetooth
devices.
fn StartBluetoothSearch();
//
// Stops searching for Bluetooth devices.
fn StopBluetoothSearch();
//
// Starts the listener for Bluetooth events for a specified duration.
// Repeatedly starting the network listener while already active will do nothing.
fn StartBluetoothListener();
//
// Stops the listener for Bluetooth events.
fn StopBluetoothListener();
//
// Returns the currently available Bluetooth adapters.
fn GetBluetoothAdapters() -> Vec<BluetoothAdapter>;
//
// Returns the current default Bluetooth adapter.
fn GetCurrentBluetoothAdapter() -> BluetoothAdapter;
//
// Sets the default Bluetooth adapter.
// The path can be found inside the BluetoothAdapter struct.
fn SetBluetoothAdapter(path: Path<'static>) -> bool;
//
// Sets the discoverability of a specific Bluetooth adapter.
fn SetBluetoothAdapterDiscoverability(path: Path<'static>, enabled: bool) -> bool;
//
// Sets the pairability of a specific Bluetooth adapter.
fn SetBluetoothAdapterPairability(path: Path<'static>, enabled: bool) -> bool;
//
// Connects to a Bluetooth device given the DBus path.
// Note that this requires an existing pairing.
// Returns true on success and false on error.
fn ConnectToBluetoothDevice(path: Path<'static>) -> bool;
//
// Pairs with a Bluetooth device given the DBus path.
// Initiates the pairing process which is handled by the Bluetooth Agent.
// Returns true on success and false on error.
// NOTE: THIS IS CURRENTLY DISABLED!
fn PairWithBluetoothDevice(path: Path<'static>) -> bool;
//
// Disconnects a Bluetooth device given the DBus path.
// Returns true on success and false on error.
fn DisconnectFromBluetoothDevice(path: Path<'static>) -> bool;
//
// This will remove the pairing on the Bluetooth device.
fn RemoveDevicePairing(path: Path<'static>) -> bool;
//
// Returns all connected Bluetooth devices.
// The first part of the HashMap is the DBus path of the object, the second is the
object
// itself.
fn GetConnectedBluetoothDevices() -> Vec<BluetoothDevice>;
```

Listing 29: Bluetooth DBus API for the ReSet daemon

11.8.4 Audio API

Table 13: Audio DBus Table

Type	Type Description
Sink	DBus signature: ussqaubi `u32, String, String, u16, Vec<u32>, bool, i32`
Source	DBus signature: ussqaubi `u32, String, String, u16, Vec<u32>, bool, i32`
InputStream	DBus signature: ussuqaubb `u32, String, String, u32, u16, Vec<u32>, bool, bool`
OutputStream	DBus signature: ussuqaubb `u32, String, String, u32, u16, Vec<u32>, bool, bool`
Card	DBus signature: a(ussuqaubb) `Vec<(u32, String, String, u32, u16, Vec<u32>, bool, bool)>`

Event
SinkAdded -> Sink
SinkRemoved -> Path<'static>
SinkChanged -> Sink
SourceAdded -> Source
SourceRemoved -> Path<'static>
SourceChanged -> Source
InputStreamAdded -> InputStream
InputStreamRemoved -> Path<'static>
InputStreamChanged -> InputStream
OutputStreamAdded -> OutputStream
OutputStreamRemoved -> Path<'static>
OutputStreamChanged -> OutputStream

DBus interface name	org.Xetibo.ReSet.Audio

```
// Returns the default sink(speaker, headphones, etc.) from pulseaudio.
fn GetDefaultSink() -> Sink;
//
// Returns the default sink name(speaker, headphones, etc.) from pulseaudio.
// This is mainly useful for checking if an event-given sink is the default one, as
this
// information is not within the sink struct for performance reasons.
fn GetDefaultSinkName() -> String;
//
// Returns the default source(microphone) from pulseaudio.
fn GetDefaultSource() -> Source;
//
// Returns the default source name(microphone) from pulseaudio.
// This is mainly useful for checking if an event-given source is the default one, as
this
// information is not within the source struct for performance reasons.
fn GetDefaultSourceName() -> String;
//
// Sets the default sink via name.(this is a pulse audio definition!)
// The name can be found inside the Sink struct after calling ListSinks() or by listening
to
// events.
fn SetDefaultSink(sink: String) -> Sink;
//
// Sets the default sink via name.(this is a pulse audio definition!)
// The name can be found inside the Sink struct after calling ListSinks() or by listening
to
// events.
fn SetDefaultSource(source: String) -> Source;
//
// Returns all current sinks.
fn ListSinks() -> Vec<Sink>;
//
// Returns all current sources.
fn ListSources() -> Vec<Source>;
//
// Returns all streams that are responsible for playing audio, e.g. applications.
fn ListInputStreams() -> Vec<InputStream>;
//
// Returns all streams that are responsible for recording audio, e.g. OBS, voice chat
applications.
fn ListOutputStreams() -> Vec<OutputStream>;
//
// Returns the PulseAudio cards for every device. (The card holds information about
all possible
// audio profiles and whether or not the device is disabled.)
fn ListCards() -> Vec<Card>;
```

Listing 30: Audio DBus API for the ReSet daemon (part1)

```
// Sets the default volume of the sink on all channels to the specified value.
// Currently ReSet does not offer individual channel volumes. (This will be added later)
// The index can be found within the Sink data structure.
fn SetSinkVolume(index: u32, channels: u16, volume: u32);
//
// Sets the mute state of the sink.
// True -> muted, False -> unmuted
// The index can be found within the Sink data structure.
fn SetSinkMute(index: u32, muted: bool);
//
// Sets the default volume of the source on all channels to the specified value.
// Currently ReSet does not offer individual channel volumes. (This will be added later)
// The index can be found within the Source data structure.
fn SetSourceVolume(index: u32, channels: u16, volume: u32);
//
// Sets the mute state of the source.
// True -> muted, False -> unmuted
// The index can be found within the Source data structure.
fn SetSourceMute(index: u32, muted: bool);
//
// Sets the default volume of the input_stream on all channels to the specified value.
// Currently ReSet does not offer individual channel volumes. (This will be added later)
// The index can be found within the InputStream data structure.
fn SetSinkOfInputStream(input_stream: u32, sink: u32);
//
// Sets the default volume of the input stream on all channels to the specified value.
// Currently ReSet does not offer individual channel volumes. (This will be added later)
// The index can be found within the InputStream data structure.
fn SetInputStreamVolume(index: u32, channels: u16, volume: u32);
//
// Sets the mute state of the input stream.
// True -> muted, False -> unmuted
// The index can be found within the InputStream data structure.
fn SetInputStreamMute(index: u32, muted: bool);
//
// Sets the target source of an output stream.
// (The target input device for an application)
// Both the output stream and the source are indexes,
// they can be found within their respective data structure.
fn SetSourceOfOutputStream(output_stream: u32, source: u32);
//
// Sets the default volume of the output stream on all channels to the specified value.
// Currently ReSet does not offer individual channel volumes. (This will be added later)
// The index can be found within the OutputStream data structure.
fn SetOutputStreamVolume(index: u32, channels: u16, volume: u32);
//
// Sets the mute state of the output stream.
// True -> muted, False -> unmuted
// The index can be found within the OutputStream data structure.
fn SetOutputStreamMute(index: u32, muted: bool);
//
// Sets the profile for a device according to the name of the profile.
// The available profile names can be found on the card of the device,
// which can be received with the ListCards() function.
// The index of the device can be found in the Device data structure.
fn SetCardOfDevice(device_index: u32, profile_name: String);
```

Listing 31: Audio DBus API for the ReSet daemon (part2)

11.9 Data structures

This section covers all relevant data structures used within ReSet and its daemon.

11.10 Daemon Data

This is the main data structure that serves as a state within the main loop.

```rust
// the data struct used by the main loop
pub struct DaemonData {
    pub n_devices: Vec<Arc<RwLock<Device>>>, // all wifi devices
    pub current_n_device: Arc<RwLock<Device>>, // current wifi device
    pub b_interface: BluetoothInterface,
    pub bluetooth_agent: BluetoothAgent,
    pub audio_sender: Rc<Sender<AudioRequest>>,
    pub audio_receiver: Rc<Receiver<AudioResponse>>,
    pub audio_listener_active: Arc<AtomicBool>,
    pub network_listener_active: Arc<AtomicBool>,
    pub network_stop_requested: Arc<AtomicBool>,
    pub bluetooth_listener_active: Arc<AtomicBool>,
    pub bluetooth_stop_requested: Arc<AtomicBool>,
    pub bluetooth_scan_active: Arc<AtomicBool>,
    pub clients: HashMap<String, usize>,
    pub connection: Arc<SyncConnection>, // connection reference used for event creation
    pub handle: JoinHandle<()>, // used for shutdown
}
```

Listing 32: DaemonData struct used as the sole data storage by the daemon

11.10.1 Audio/PulseAudio

For PulseAudio, ReSet offers a struct for sinks, sources, sink inputs, source outputs and cards. These translate to audio output device, audio input device, audio output stream, audio input stream and audio codec profiles respectively.

The data structures for sink/source and inputstream/outputstream have the same properties.

```rust
// Sink/Source
#[derive(Debug, Clone, Default)]
pub struct Sink {
    pub index: u32,
    pub name: String,
    pub alias: String,
    pub channels: u16,
    pub volume: Vec<u32>,
    pub muted: bool,
    pub active: i32,
}

#[derive(Debug, Clone, Default)]
pub struct InputStream {
    pub index: u32,
    pub name: String,
    pub application_name: String,
    pub sink_index: u32,
    pub channels: u16,
    pub volume: Vec<u32>,
    pub muted: bool,
    pub corked: bool,
}

#[derive(Debug, Clone, Default)]
pub struct Card {
    pub index: u32,
    pub name: String,
    pub profiles: Vec<CardProfile>,
    pub active_profile: String,
}
```

Listing 33: Bluetooth data structures

11.10.2 Bluetooth/Bluez

For bluez, ReSet-Lib offers two structures, the Bluetooth Adapter, which is used to connect to devices, and the devices themselves.

```rust
// Bluetooth Adapter
#[derive(Debug, Clone, Default)]
pub struct BluetoothAdapter {
    pub path: Path<'static>,
    pub alias: String,
    pub powered: bool,
    pub discoverable: bool,
    pub pairable: bool,
}

// Bluetooth Device
#[derive(Debug, Clone, Default)]
pub struct BluetoothDevice {
    pub path: Path<'static>,
    pub rssi: i16,
    pub alias: String,
    pub name: String,
    pub adapter: Path<'static>,
    pub trusted: bool,
    pub bonded: bool,
    pub paired: bool,
    pub blocked: bool,
    pub connected: bool,
    pub icon: String,
    pub address: String,
}
```

Listing 34: Bluetooth data structures

11.10.3 Wi-Fi/NetworkManager

For NetworkManager, ReSet offers two structures, WifiDevice and AccessPoint.

```rust
// Wi-Fi Device
#[derive(Debug, Clone, Default)]
pub struct WifiDevice {
    pub path: Path<'static>,
    pub name: String,
    pub active_access_point: Vec<u8>,
}

// Access Point
#[derive(Debug, Clone, Default)]
pub struct AccessPoint {
    pub ssid: Vec<u8>,
    pub strength: u8,
    pub associated_connection: Path<'static>,
    pub dbus_path: Path<'static>,
    pub stored: bool,
}
```

Listing 35: Wi-Fi data structures

11.11 Libraries

This section covers all direct third-party libraries used within ReSet.

- DBus, DBus-crossroads, DBus-toktio [66]: Apache-2.0, MIT
- libadwaita [42]: MIT
- gtk4 [67]: MIT
- glib [68]: MIT
- tokio [69]: MIT
- fork [70]: BSD-3-Clause
- ipnetwork [71]: Apache-2.0
- libpulse_binding [45]
- directories-next [72]: Apache-2.0, MIT
- crossbeam [46]: Apache-2.0, MIT

11.12 Additional Screenshots

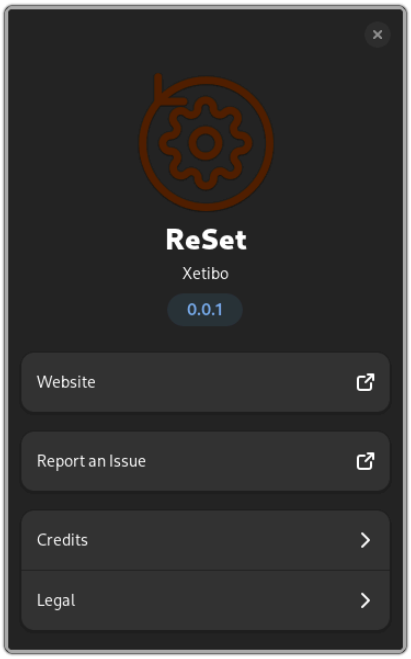


Figure 35: the ReSet About window

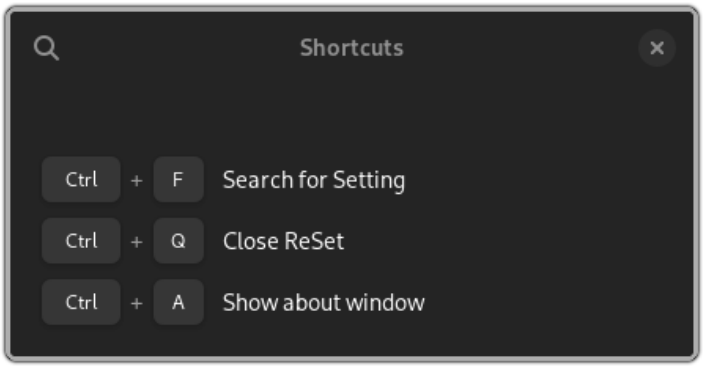


Figure 36: the ReSet shortcuts window

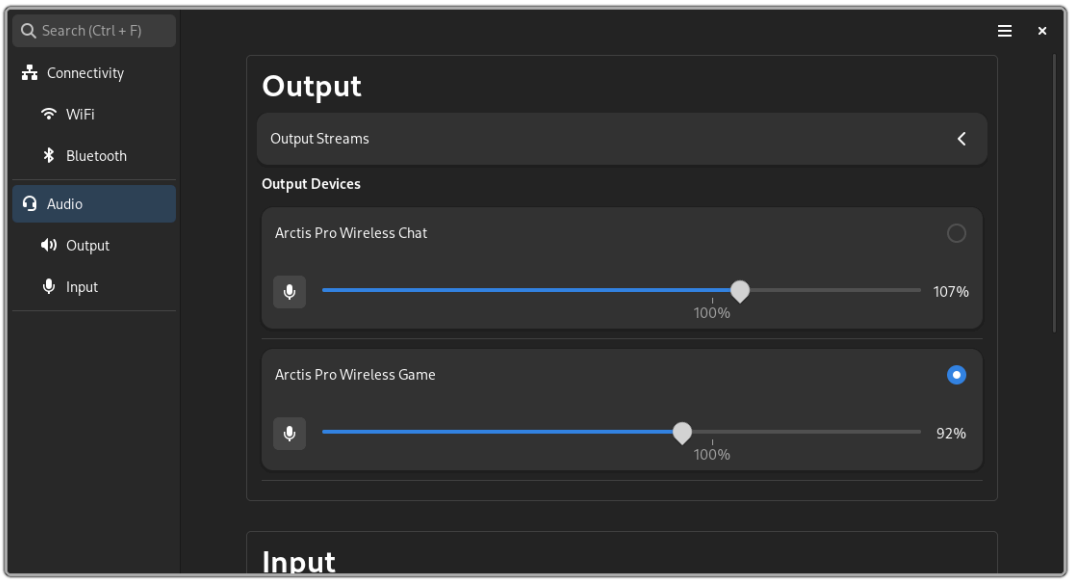


Figure 37: ReSet Audio devices section

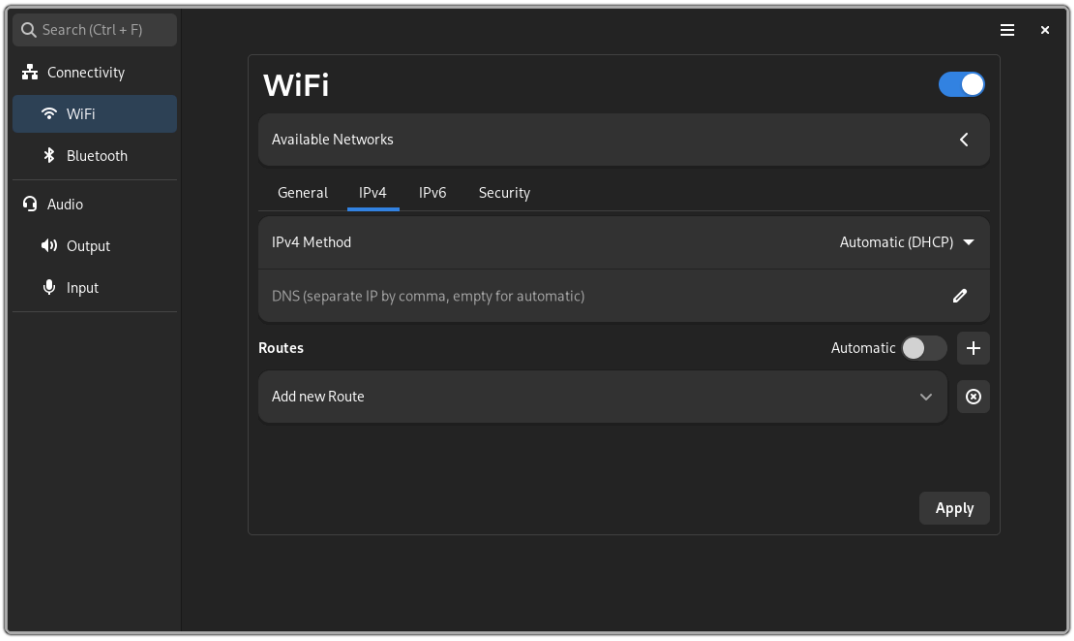


Figure 38: ReSet Wi-Fi IPV4 configuration