**Architecture Improvement**

# **Proposal**

### **Migration from joydev to evdev**

The current architecture of WeJoy uses the "joydev[[1]](#footnote-2)" driver in order to get information about connected joysticks. The current architecture manually talks to joydev and as a result, it has its own implementations of the exposed data structures exposed by joydev, and requires the use of the "udev[[2]](#footnote-3)" API for querying information about the connected joystick, such as a vendor or product id.

A better solution would be to use the "evdev[[3]](#footnote-4)" (Event Device) driver with a wrapper like "libevdev[[4]](#footnote-5)". This allows us to leave the implementation of the evdev up to libevdev, and we only need to ask for events using data structures it has already defined for us. This also allows us to remove the use of udev, as the evdev API exposes this information for us.

Another benefit is this removes the requirement for controllers to be connected via usb, and opens up both Bluetooth and even virtual devices.

### **Testing framework**

The current architecture of WeJoy has no tests. I would like to write a testing library that creates a virtual joystick, and then runs WeJoy with an example script configured for the joysticks. I would then like feed data into the virtual joystick and use libevdev to check that the virtual joysticks and keyboard created by WeJoy are outputting the correct data. I would like to implement this using Catch2 and BDD, as the Given When Then style of defining tests fits this domain very well, as we have specific inputs that should trigger specific outputs.

### **New build system**

WeJoy was using a shell script to build. This is not a recommended practise as it involves hard coding the building application instead of leaving it up to a build system. I would like to move this to a Cmake[[5]](#footnote-6) system. This gives me the ability to write a build script that can work on any Linux machine, and one that only builds code that has changed. This change also resolves the issue of hardcoded constants, as Cmake supports the ability to call custom commands during the build process, so we can generate headers based on the Linux device list, and we no longer need to hardcode a device list.

# **Alterations**

### **Migrate from joydev to evdev**

WeJoy was using on the older joystick API for querying device information. This API lacked certain features required for the program, such as querying product and vendor IDs. The original project used code based on the udev API to provide that functionality, which is unnecessary when existing APIs provide this. This udev provided code was also inflexible, as it needed different versions of code to handle different device types. I have migrated this code base to libevdev, and as a result, the information can now come directly from libevdev, and we no longer have a limitation of USB devices. This opens up the ability to pass virtual devices in for testing, or use Bluetooth devices.

### **Removal of the event structure**

WeJoy was previously using its own event structure in order to manually read data from joydev. This is no longer needed, as libevdev provides this type of data for us. As a result, the code for handling events is much simpler as it has access to the required data when called. This also allows us to no longer need to update data structures related to external APIs, which is useful to avoid features breaking if the APIs we are dependent on change.

### **Added functions for receiving many events at once**

While testing, I needed a way to identify what joystick button was pressed. To do this, methods have been added to the Lua script that allow a user to write general handlers for all axes or button events. I have added a method to Lua called axis\_event that is called when any axis change occurs, and this event is passed in a device, the changed axis and the value the axis was changed to. This allows the user to define a handler for multiple axes to clean up their scripts. The same was done for buttons, but with a function called button\_event.

### **Added an importer for the events from the Linux kernel instead of hardcoding basic events**

WeJoy had a hardcoded list of events that it could listen to. I thought this was bad practise, as hardcoding should be avoided if possible. This also means if support for a new device was added to the Linux kernel, and it had a new set of buttons or axes, we would need to manually add support to our application for this new data.

Instead, I found a script from the evdev[[6]](#footnote-7) repository on GitHub, and modified it to allow me to generate a list of events, and have changed the code to reference this list. This script opens the global device list file on Linux, and processes it to pull out useful information such as device names. It then reformats the device information into a c++ map, and exposes it to the code base. I then exposed this data to Lua, so it could be used by developers writing scripts.

### **Added functions to get the max and min values of a specific axis**

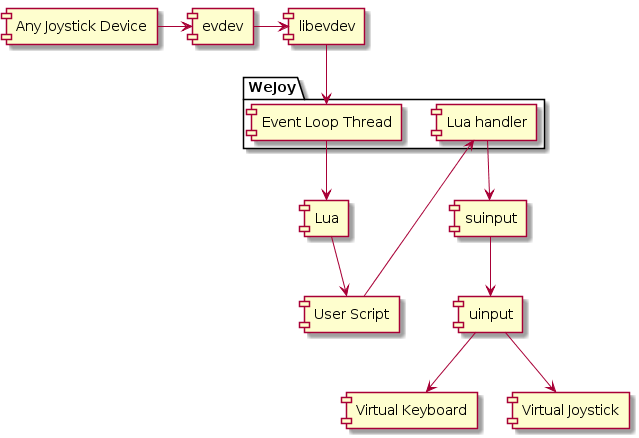
While testing, I found it difficult to map from a real device to a virtual one as they both used different max and min values. This meant that one joystick may have been at the max value, and the other joystick may not even have registered a change. I added the ability to retrieve this information so a user could write a simple script in Lua to map from one coordinate system to another. I also added constants for the joystick axis ranges used the virtual joystick.

### **Modified the original code base to support virtual devices**

I required the ability to test the orignal code base to make sure my changes did not break any existing functionality. In order to do this, I required a way to run the orignal code against my current testing framework. To do this, I made minimal changes to the original code to remove the limitation on USB, by changing the udev code to look up vendor id and product ids for a virtual device instead of a USB device. One issue I found was that the original code was slower, so I needed to modify my tests to run slower to make sure the original code would send events out. However, one set of tests refuses to pass, as there appears to be a bug in the original code.

# **Evaluation**

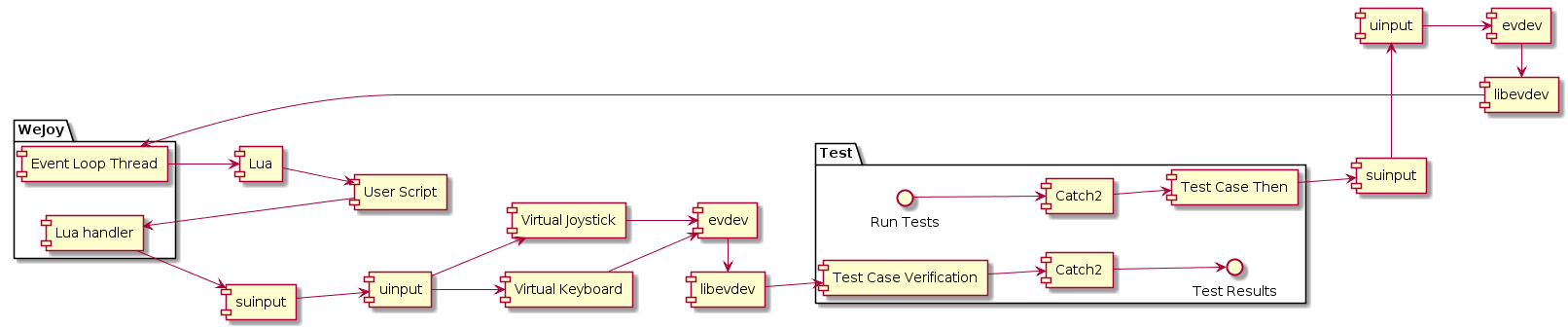
The main problem that this improvement solves is extensibility. In order to solve this, I have migrated the code base from joydev to libevdev. This allows us to handle more devices than USB, and also allows us to use newer devices that do not support joydev.

  
Figure 1: The new architecture

### **Extensibility**

Extensibility of software is a recognised aspect of quality in a product, as defined by ISO/IEC 25010[[7]](#footnote-8). This change has made the software more compatible with other devices, allowing it to be used for more purposes. There is also a lot less hard-coded values, which means the product will stay up to date if Linux decides to add more devices.

Additionally, my alterations have improved the testability of the project, solving a major issue of the original program. I have implemented a full testing framework based on Catch2, which allows a developer to easily write tests using a BDD (Behaviour Driven Design) syntax. This allows someone to write a demonstration script, and then using a virtual controller, test that script and as a result test the program as long as the script is comprehensive enough. See *Figure 2* for a full overview of the test framework architecture.

  
Figure 2: Architecture diagram for the testing framework

# **Stakeholder Concerns**

The two main stakeholders of WeJoy are the users that are writing scripts that target the program and the developers of the program. The testing architectural improvement primarily targets the developers, while the extensibility improvements target both the user and the developer.

The developers of a software project have an interest ensuring that a product stays up to date and relevant at all times. If a product is designed with outdated components, it will eventually be replaced with a newer product if that product supports a broader range of devices. In this case, updating the API will allow the product to stay relevant for longer as it can now be used to solve problems that other products cannot.

Developers of a product will find that a product with a proper testing framework is much more enjoyable product to work on. A better testing framework would allow developers to ensure validity and functionality of their extensions to the project, as well as assisting with debugging. Using BDD, we can make sure that tests are both easy to write, and can test all parts of the application in unison. For this reason, developers as stakeholders will find the architectural changes I have made useful.

Developers of this product will also find that a proper build system is a lot more pleasant than a single script that runs a command, as it gives the ability to run other scripts and tests from the build scripts, instead of being limited to a single command.

Users of this product will find that the project is much more pleasant to work with, as extensions to the Lua scripts mean that the user can write simpler scripts with less repetition. While testing, I found that I needed a way to get information about a joystick from the Lua side, and so I have implemented features for getting calibration constants for a joystick. I have also added features that allow targeting multiple axes and buttons at once, to allow the user to work out key bindings, or skip repetitive scripts.

Users of this product will also find that the project is much more compatible as a result of the changes, and it in some cases no longer requires root access. It also now works with a wider range of devices, and for a project in this domain that is an important feature for a stakeholder.

1. <https://www.kernel.org/doc/Documentation/input/joystick-api.txt> [↑](#footnote-ref-2)
2. <http://man7.org/linux/man-pages/man3/libudev.3.html> [↑](#footnote-ref-3)
3. <https://www.x.org/archive/X11R7.6/doc/man/man4/evdev.4.xhtml> [↑](#footnote-ref-4)
4. <https://www.freedesktop.org/software/libevdev/doc/latest/> [↑](#footnote-ref-5)
5. <https://cmake.org/> [↑](#footnote-ref-6)
6. https://github.com/whot/libevdev/blob/master/libevdev/make-event-names.py [↑](#footnote-ref-7)
7. <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&start=6> [↑](#footnote-ref-8)