Description

Overview

WeJoy is an open-source program written for linux, that allows a user to write scripts that take input from game controllers, and then reroute that input to either a keyboard or any number of virtual game controllers. WeJoy uses the joystick API[[1]](#footnote-1) provided by linux to read data from connected controllers and uses the udev[[2]](#footnote-2) API to query joysticks for information. This also puts a limitation on the type of controllers that can be used, limiting the usage of this application to USB controllers. WeJoy then uses the uinput[[3]](#footnote-3) API to create a virtual keyboard and any number of virtual controllers to send scripted events to.

History

WeJoy was started by Johannes Bergmark (Vantskruv on GitHub), and began in March 2015. WeJoy began as a simple copy and paste of libsuinput[[4]](#footnote-4) and parts of unnamed\_lua\_binder[[5]](#footnote-5), with a small wrapper around it to allow uinput functions to be called from a lua script.

Lua

WeJoy uses a scripting language called Lua to provide the ability for uses to define how different actions relate to each other. This gives the advantage of giving the user complete control over how the joystick works, but also means we can use a simple Lua API for passing control to and taking control from the script.

API

WeJoy uses four APIs to make the application function. It uses the joystick API, which gives a developer the ability to poll for Joystick events. It also uses the udev API, as it allows a developer to query information about connected devices, such as the vender and product ids of a controller. These are then used in scripts to select what device you want to listen to. WeJoy then uses the suinput API as it provides a wrapper around uinput, which gives a developer the ability to create and control virtual input devices such as controllers and keyboards.

WeJoy also uses a Lua scripting API that allows us to interface with user defined scripts written in Lua. This allows the application to be extremely extendable, as the user is in control of what different inputs and outputs do.

Domain

WeJoy exists in the domain of input device emulation, input device interfacing and scripting.

Scripting

This domain requires an easy to understand but flexible language to sit as a translation layer between input controllers and output controllers. This language needs to allow the user to easily write scripts that can handle user input, and then forward that to another device such as a virtual controller or keyboard. This also means that the scripting system needs to have a decent API for talking to virtual devices.

Device Emulation and Device Interfacing

There needs to be careful attention paid to the speed of this interface. If the emulated device or interfaced device react too slow, then this application would not be useful, as someone may want to use it to play a game, and in that case, a quick reaction time is required. We also need to make sure that we can handle multiple types of devices, so we need to make sure that our interface is flexible enough to handle a wide variety of joysticks and user input devices, but it also needs to be rigid enough that someone can specify a specific joystick on their system and know that that is the device they are interfacing with. The emulation also needs to be flexible enough to handle emulating a wide variety of output devices, with a varying number of buttons and joystick axes.

Component Architecture

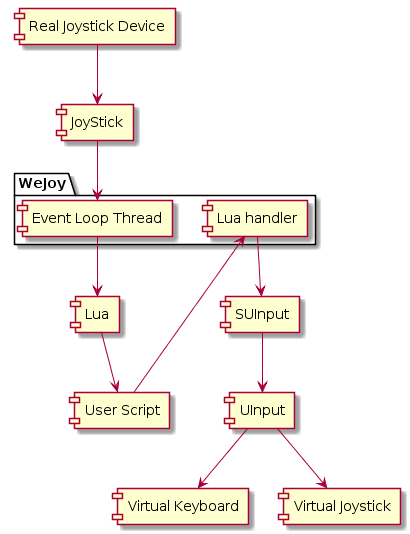
Event Driven Architecture

Figure 1 – WeJoy Structure

WeJoy is modelled after an event driven architecture. As shown in *Figure 1*, It is engaged after a real joystick sends an event to the JoyStick API, which WeJoy polls for joystick updates. After this, the event loop thread calls a method in the user defined lua script, dependant on what button / axis on which joystick it received an input for. This user script then runs user defined code, which can call a method in our lua handler, which will then forward that request to SUInput, which forwards it to UInput, and then it will control either a virtual keyboard or any number of virtual joysticks, depending on what the user specified.

As an example, a user may wish for the first button on their first controller to press the ‘A’ key on their keyboard. The user would write a script such as the example script, and then they would start WeJoy. When the user presses the first button, the JoyStick API receives an event, and then the Event Loop Thread polls the JoyStick API and finds that the first button has been pushed. It then calls the function d0\_b0\_event in the user script. The example lua script then directly calls the send\_keyboard\_event with the A key and the state of the button. This tells the lua handler that you want to press a keyboard key, and then it uses the SUInput API to tell UInput to press a key on the virtual keyboard. Releasing the first button has the same effect, only it will turn the key off.

Components

Event loop Thread

The event loop thread is responsible for requesting data from the joystick API and passing that data on to Lua. The event loop thread loops forever, and it receives an event structure from the joystick API, and then parses this to work out what functions to call inside the lua script.

Lua Handler

The lua handler is called directly by the lua script and is responsible for giving the scripts the ability to control virtual devices. It takes in arguments like what button to press and has a function for keyboard control and a function for joystick control.

Data Structures

One of the main data structures on WeJoy is the event structure. While it is redefined in joystick.h, it is a structure provided by the joystick API, and is where data is written to when we poll the joystick API for data. It contains information about when the event occurred, the type of event, what axis / button was affected and what the effected input was set to.

There are a few data structures in the lua script to give the user the ability to define the joysticks they would like to connect. There is a devices data structure, that stores several devices. Each device has a set vendor and product id, and since this is unique for each different kind of joystick, this is enough to identify that a specific controller is sending events. There is also a v\_devices data structure that stores information about user defined virtual devices. A v\_device stores the amount of buttons and the amount of axes that a virtual device has.

Appendix

Example Script

**function** d0\_b0\_event(value)  
 send\_keyboard\_event(KEY\_A, value)  
**end**

1. <https://www.kernel.org/doc/Documentation/input/joystick-api.txt> [↑](#footnote-ref-1)
2. <http://man7.org/linux/man-pages/man3/libudev.3.html> [↑](#footnote-ref-2)
3. <https://www.kernel.org/doc/html/v4.16/input/uinput.html> [↑](#footnote-ref-3)
4. <https://github.com/tuomasjjrasanen/libsuinput> [↑](#footnote-ref-4)
5. <https://github.com/eliasdaler/unnamed_lua_binder> [↑](#footnote-ref-5)