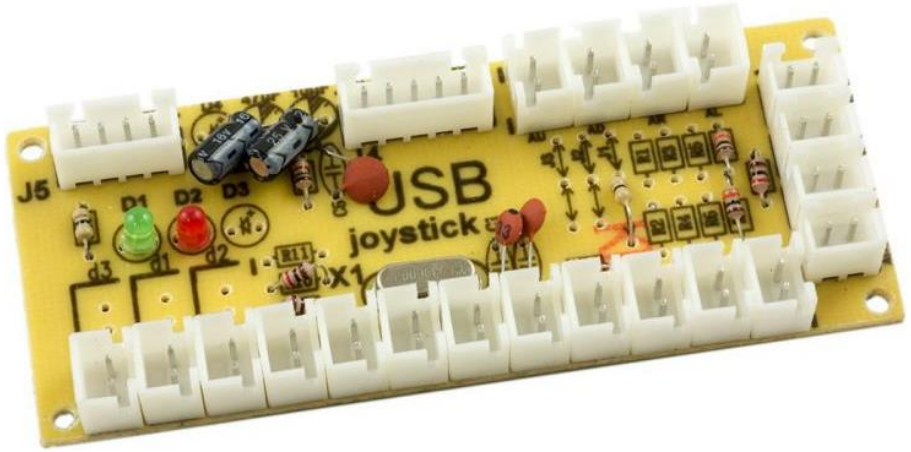
Lectura interruptores en Linux con tarjeta “Generic USB joystick” de DragonRise para aplicaciones IoT

V04
junio 2020
junavarg
junavarg@hotmail.com



Introducción

El programa dragonrise permite:

1) Monitorizar el estado de los 12 interruptores (y 2 conmutadores) de la popular tarjeta controladora “Generic USB joystick” de DragonRise Inc.
2) Publicar en un servidor o broker MQTT el estado de estos interruptores cada vez que se produce un cambio en cualquiera de ellos.

Está escrito en Golang por lo que es un ejecutable directo sobre el procesador (no requiere runtime) y puede funcionar en background como un demonio o servicio .

El mismo proceso permite monitorizar varias tarjetas controladoras lo que es ideal para aplicaciones que requieran manejar un elevado número de interruptores.

Permite transportar el protocolo MQTT sobre TCP o sobre Websockets, incluido la versión cifrada sobre TLS. En este último caso no se precisa la inclusión de los certificados de la cadena de confianza del servidor MQTT.

Dirigido a ejecutarse en single-board computers (SBC), al estar escrito en Golang, puede ejecutarse en Linux o incluso, con las debidas modificaciones, en Windows.

Junto al programa se incluye un fichero de reglas UDEV para asegurar el orden y la correcta identificación de las tarjetas controladoras en función del puerto USB donde estén enchufadas.

The dragonrise program allows:

1) Monitor the status of the 12 switches (and 2 three position switches) of the popular "Generic USB joystick" controller card from DragonRise Inc.

2) Publish the status of these switches to a server or MQTT broker every time a change occurs in any of them.

It is written in Golang so it is a direct executable on the processor (it does not require a runtime) and it can work in the background as a daemon or service.

The same process allows multiple controller cards to be monitored which is ideal for applications requiring a high number of switches.

It allows transporting the MQTT protocol over TCP or over Websockets, including the encrypted version over TLS. In the latter case, the inclusion of the certificates of the MQTT server trust chain is not required.

Oriented to run on single-board computers (SBC), being written in Golang, it can run on Linux or even, with the necessary modifications, on Windows.

Along with the program, a UDEV rules file is included to ensure the order and correct identification of the controller cards according to the USB port where they are plugged in.

Motivación

En los proyectos de IoT es frecuente necesitar manejar información que se obtiene por el cierre o apertura de interruptores: pulsadores, interruptores, contactos reed, conmutadores, finales de carrera, etc.

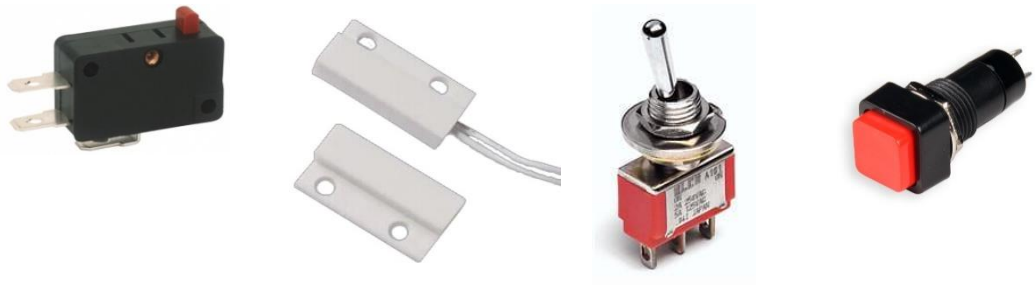


Ilustración 1. Ejemplo de interruptores conectables a sistemas IoT

Una forma de conectar esto interruptores es haciendo uso de puertos de entrada de los GPIO presentes en los SBC “single-board computers” (por ejemplo Raspberry pi) o “single-board microcontrollers” (por ejemplo Arduino). Los principales problemas de esta aproximación son:

- Cableado muy desordenado especialmente en sistemas que precisen una alto número de interruptores.
- Posibilidad de avería en la placa al manipular en contactos no estructurados y desprotegidos y con diferentes tensiones.
- Número limitado de interruptores disponibles sin posibilidad de escalado.
- En proyectos con orientación comercial el montaje, mantenimiento y reparación son más complicados.
- Mayor dificultad de desarrollo, depuración y pruebas en entornos PC/Mac al no disponer estos de los elementos hardware (bus GPIO).

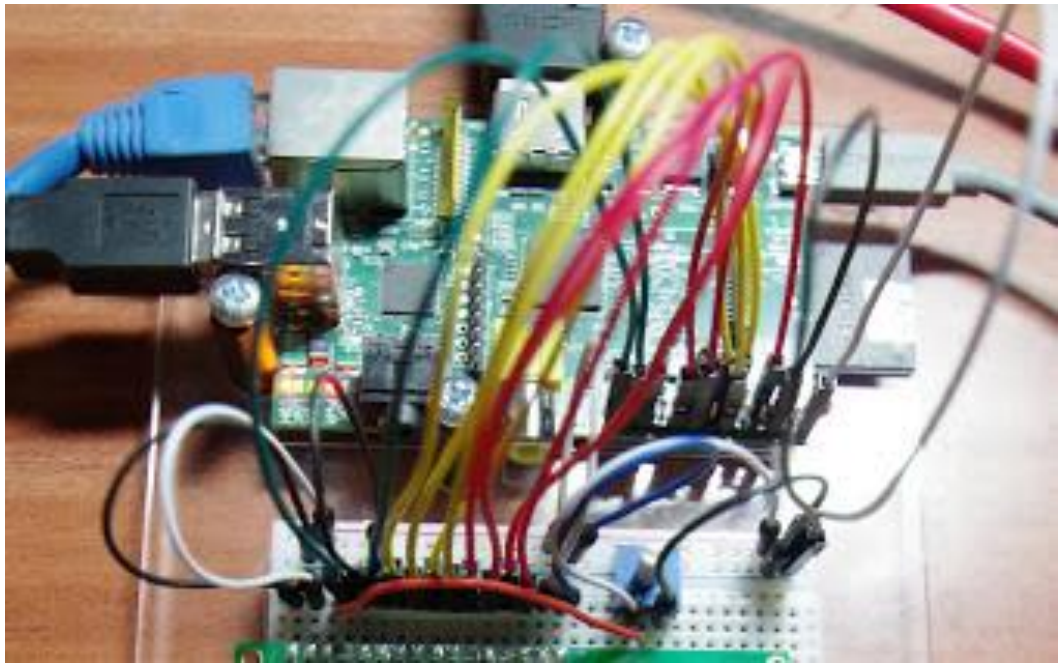


Ilustración 2. Conexiones en conector GPIO de una Raspeberri PI haciendo uso de una breadboard.

Estas dificultades son las que motivaron la búsqueda de una solución que permitiera dotar de entrada de información binaria de forma más flexible y con bajo coste a los sistemas basados en SBC. La solución que se ha encontrado como más conveniente es el empleo de bancos de interruptores conectables vía USB. Un ejemplo de esta aproximación es el teclado de un ordenador con conexión USB.

Otro ejemplo que para el tipo de aplicaciones es aún más conveniente para este tipo de aplicaciones es el empleo de un dispositivo tipo joystick. Entre estos dispositivos destaca por su bajo coste, ubicuidad y estandarización, las tarjetas “Generic USB joystick” con controlador DragonRise. El empleo de tarjetas USB de joystick aporta la ventaja en sistemas empotrados o empotrables tipo Linux (por ejemplo Raspbian) la disponibilidad de controladores de dispositivo que facilitan notablemente la integración del conjunto de interruptores en sistema.

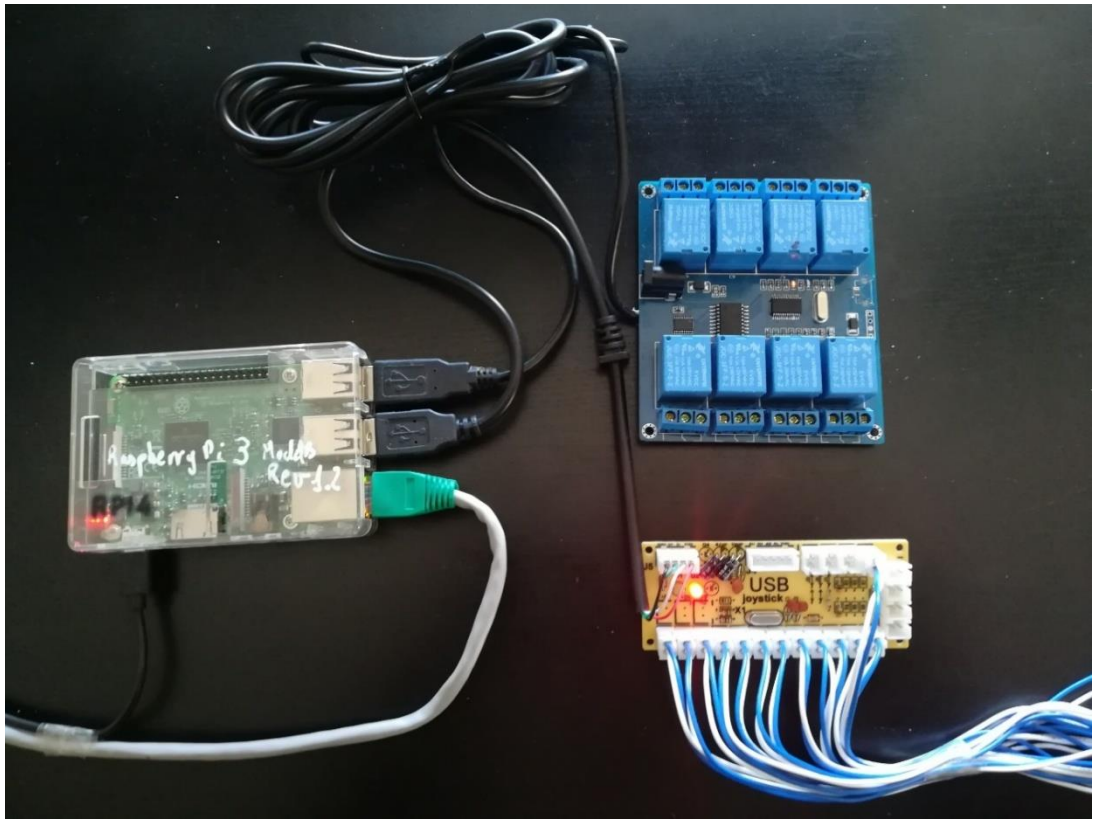


Ilustración 3. Tarjeta DragonRise con 13 conexiones para interruptores conectada a un puerto USB de una Raspberry Pi. La imagen también muestras una placas de 8 relés conectada a otro puerto. La foto permite apreciar la mejor estructuración y limpieza de las conexiones.

El sistema así concebido es fácilmente escalable haciendo uso de más tarjetas USB y, en caso necesario, de concentradores (hubs) USB.

No todo son ventajas, la introducción de la tarjeta aumenta la complejidad del sistema por lo que el tiempo confirmará si la fiabilidad del sistema queda comprometida. Por lado la tarjeta provocará un mayor consumo en el bus USB que puede afectar al funcionamiento del conjunto de dispositivos conectados a éste bus. Este punto puede ser parcialmente aliviado eliminando el led rojo que permanente está encendido en la tarjeta o, de forma definitiva, , haciendo uso de concentradores USB con alimentación propia pese a aumentar la complejidad del conjunto.

Descripción de la placa DragonRise

La placa a emplear se muestra en la figura

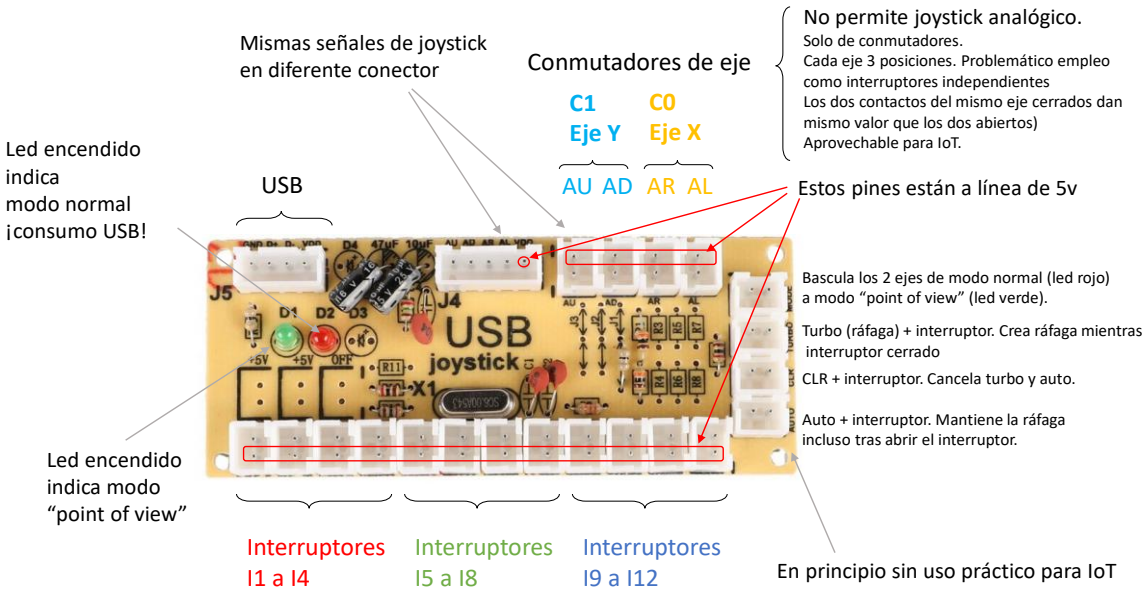


Ilustración 4. Tarjeta DragonRise con comentarios de los puntos más relevantes

¡Ojo!. Al contrario que la mayoría de tarjetas, la tarjeta Dragonrise destina la mayor (y más expuesta) parte la superficie de cobre a la línea de potencial de 5v en lugar de a GND o tierra (0v).

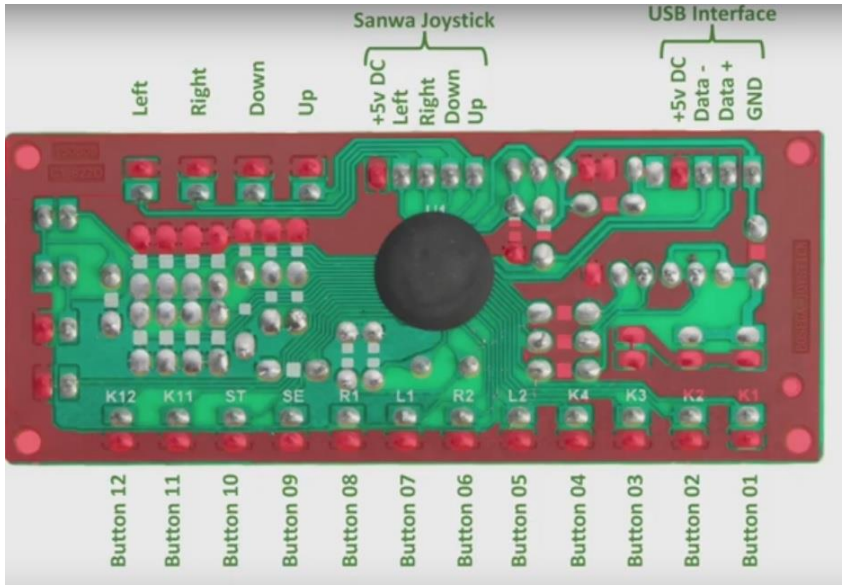


Ilustración 5. Reverso de la tarjeta DragonRise donde de muesta que la superficie mas extensa no es tierra-común sino la linea de tension de 5 voltios.

Esto implica que, en general, solo se podrán conectar interruptores y conmutadores “flotantes” que no tengan tensión alguna en ninguno de sus terminales.

Más información en https://www.s-config.com/zero-delay-usb-joystick-encoder/

Programa dragonrise

El programa para Linux objeto de este proyecto se puede descargar en https://github.com/junavarg/dragonrise.

Devuelve por stdout una estructura JSON con información de eventos y estado de interruptores y ejes(conmutadores) de la tarjeta 'Generic USB joystick' de DragonRise para uso en IoT.

También permite la publicación en un MQTT broker esta información.

Use: dragonrise [options] [device_file1] [device_file2]… ")

Opciones:

-mqpub <url>

Especifica la URL de brocker MQTT y la raiz de un topic (basetopic) donde publicar el estado cada vez que se produzca un evento. El formato de url es

*protocol***://**[*user*[:*password*]@]*host.dominio.tld***:***puerto***/***base_topic*

Opciones para protocol: tcp, ssl,ws,wss

Ejemplos:

-mqpub=tcp://host.dominio.dom:1883/base_topic

-mqpub=ssl://pepe@host.dominio.dom:8883/base_topic

-mqpub=ws://host.dominio.dom:80/base_topic

-mqpub=wss://pepe:p2ssw0d@host.dominio.dom:443/base_topic

-mqpub2

-mqpub3

Broker adicionales a los que el programa envía eventos

Los mensajes se publican en 'clean session' con qos 0 y con 'retained flag' para que en cada nueva conexión el subcriptor reciba un mensaje con el estado actual.

-cbc

Check Broquer Certificate. Habilita que se verifique el certificado presentado por el broker MQTT en los protocolos protegidos por TLS así como la cadena de certificación hasta el root certificate

Detalles técnicos

El programa está realizado en Golang y se puede generar código ejecutable para x86 y arm6, arm7.

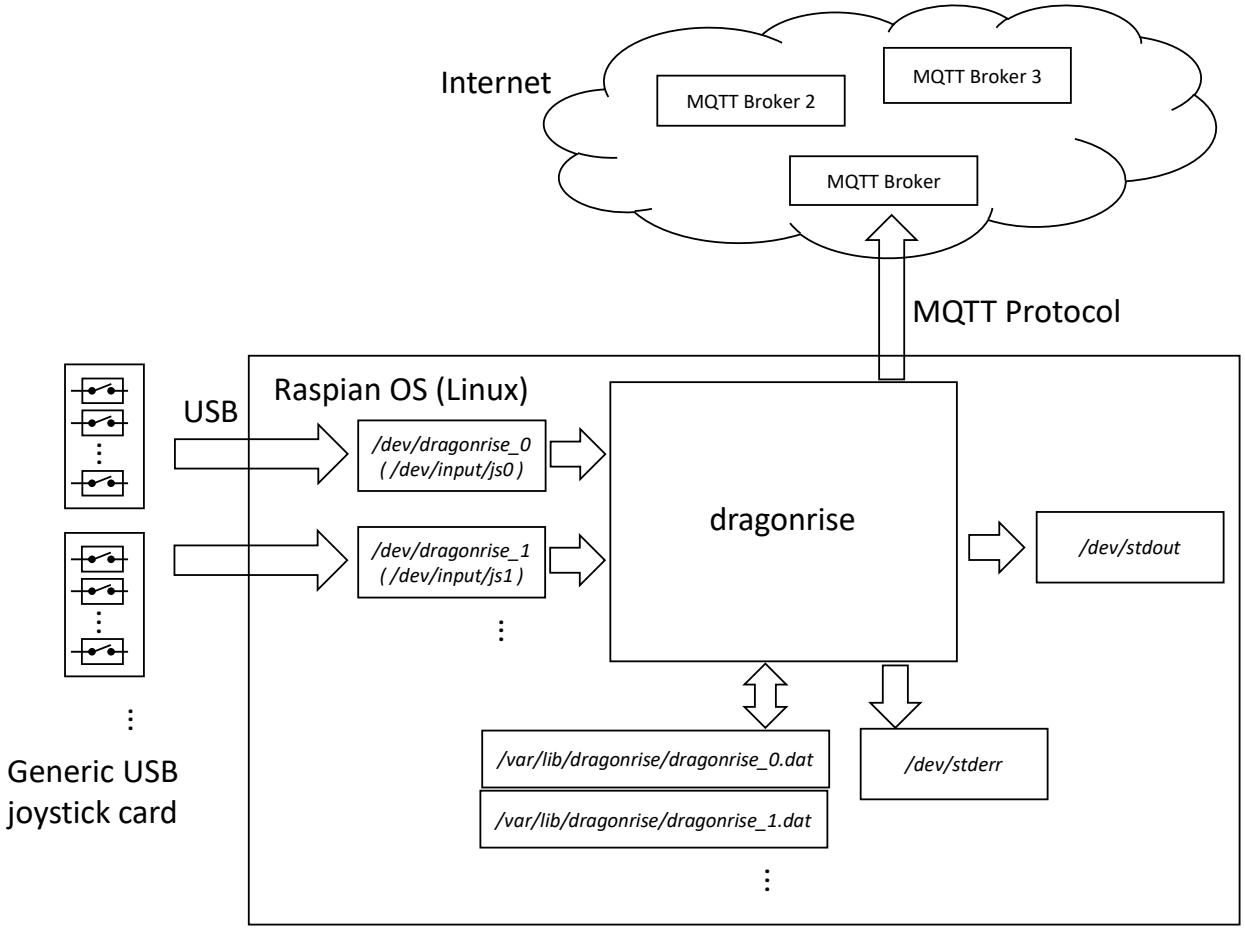


Ilustración 6. Diagrama de recursos que intervienen en el programa DragonRise.

Lee fichero de dispositivo */dev/input/js0* u otro especificado en la línea de comando. Cada fichero de dispositivo se corresponde con una tarjeta DragonRise.

La inicialización de la tarjeta consiste en la apertura de fichero de dispositivo y la lectura de eventos sintéticos que permite conocer el estado actual de interruptores y conmutadores que se guarda en un fichero '*<device_file>.dat*' en */var/lib/dragonrise/*.

Estas primeras lecturas tras la apertura del fichero consisten en 12 lecturas de interruptores seguidas de 7 lecturas ejes/conmutadores (total 19 lecturas) para anotar internamente el estado actual. Cada lectura es de 8 bytes con una estructura idéntica (ver apartado Anexo I. Interfaz Linux para joystick

/dev/input/js0). Sin embargo la tarjeta empleada solo facilita conexiones físicas para 2 ejes/conmutadores (sanwa joystick) por lo que sólo tendrán información los números 0 y 1.

Se permite que un mismo proceso pueda atender a más de una tarjeta Dragonrise.

El programa elimina los eventos espurios que aleatoriamente se generan para ejes/conmutadores que no están físicamente disponibles en la tarjeta.

El mensaje JSON que se saca por stdout tiene el siguiente formato:

```
{"time":<unix_time>,
"device":"<devicefile>","event":{"type":t,"sensor":<s>,"value":<v>},"switches":[v,v,v,v,v,v,v,v,v,v,v,v],"axes":[v,v,0,0,0,0,0]}
```

Donde te puede tener valores: 0: Valor inicial (no evento real), 1: evento de interruptor, 2: evento de eje/conmutador, -1: Error

Esta misma información es la que se guarda en un fichero como estado actual en el fichero '*<device_file>.dat*' en */var/lib/dragonrise/*.

La siguiente tabla muestra diferentes mensajes por stdout en función de la situación reportada al ejecutar

```
dragonrise /dev/input/js0 2>/dev/null
```

Situación	Salida JSON por stdout
Arranque de programa con una tarjeta conectada al bus USB y asociada con dispositivo de fichero "`js0`"	{"time":1590225475,"device":"js0", "event":{"type":0,"sensor":0,"value":0}, "switches":[1,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0] }
Evento en interruptores en una tarjeta conectada al bus USB y asociada con dispositivo de fichero "`js0`"	{"time":1590225866,"device":"js0", "event":{"type":1,"sensor":0,"value":0}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0] }
Evento en eje/conmutador en una tarjeta conectada al bus USB y asociada con dispositivo de fichero "`js0`"	{"time":1590226116,"device":"js0", "event":{"type":2,"sensor":0,"value":-1}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[-1,0,0,0,0,0,0]}
Desconexión de la tarjeta del bus USB y asociada con dispositivo de fichero "`js0`"	{"time":1590226371,"device":"js0", "event":{"type":-1, "reason":"Dispositivo dejó de estar disponible"}}

Conexión de la tarjeta al bus USB y asociada con dispositivo de fichero “`js0`”	{"time":1590226517,"device":"`js0`", "event":{"type":0,"sensor":0,"value":0}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[-1,0,0,0,0,0,0]}

Se permite que un mismo proceso pueda atender a más de una tarjeta Dragonrise. Por ejemplo:

```
dragonrise /dev/input/js0 /dev/input/js1 2>/dev/null
{"time":1590230079,"device":"js0","event":{"type":0,"sensor":0,"value":0},"switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[-1,0,0,0,0,0,0]}
{"time":1590230079,"device":"js1","event":{"type":0,"sensor":0,"value":0},"switches":[0,1,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
```

Es conveniente (el script de instalación lo hace) que se cree este directorio con permisos chmod 777. De esta manera el programa dragonrise puede ejecutarse sin privilegios al poder crear y escribir los ficheros de estado en el directorio.

La estructura interna en memoria que mantiene el estado de los sensores es un array de dragonrise struct:

```
type dragonrise struct{
	Tiempo int64		`json:"time"`
	Dispositivo string	`json:"device"`
	Evento evento		`json:"event"`// último evento registrado
	Swt []int16		`json:"switches"`// estado actual de interruptores
	Com []int16		`json:"axes"`// estado actual de ejes/conmutadores
}

type evento struct{
	TipoSensor int8	`json:"type"`   //originalmente era de tipo byte. Se cambió a int8 para permitir valor "-1" que indica error
	NumSensor byte		`json:"sensor"`
	ValorSensor int16	`json:"value"`
}

maxTarjetas = 10 //número máximo de tarjetas (files devices) que se almacenaran en un array
const max_swt = 12    //número máximo de interruptores
const max_com = 7     //número máximo de ejes-conmutadores

//variables globales

var (
	numTarjetas int				//numero de tarjetas gestionadas por el proceso dragonrise. Siempre numTarjetas <= maxTarjetas
	tarjeta [maxTarjetas]dragonrise     // array de estructuras de ultimo evento y estados
	switches [maxTarjetas][maxSwt]int16     // array de interruptores
	conmutadores [maxTarjetas][maxCom]int16 // array de ejes/conmutadores
```

```
        fDispositivo [maxTarjetas]string    //nombre de fichero de dispositivo
        hDispositivo [maxTarjetas]*os.File  //handle de fichero de dispositivo
        fEstado [maxTarjetas]string         //nombre de fichero de estado
        hEstado [maxTarjetas]*os.File               //handle de fichero de estado
)
```

Se permite que un mismo proceso pueda atender a más de una tarjeta Dragonrise. La estructura arriba descrita realmente es un array indizada por número de tarjeta.

Conexión y publicación en broker MQTT

La documentación de la librería MQTT de golang empleada se puede consultar en https://godoc.org/github.com/eclipse/paho.mqtt.golang .

La conexión al broker MQTT se realiza con un "ClientID" que se genera con la MAC del primer adaptador de red enumerado en el sistema al que se le añade, separados con un guion el nombre del device_file especificado en la línea de comando.

Los mensajes se publican en 'clean session' con qos 0 y con 'retained flag' para que en cada nueva conexión el subcriptor reciba un mensaje con el estado actual.

EL "topic"de subscripción de la cola se obtiene componiendo

- el elemento base_topic especificado en la url de la opción -mqbub con
- un carácter "/"
- el device_file especificado en la línea de comando correspondiente la tarjeta.
- la cadena "/event"

Se permite hasta 2 brokers adicionales (opciones -mqpub2 y -mqpub3). El topic base es el mismo que el especificado en el primer broker (opción -mqpub).

Cada conexión a un broker corresponde un cliente MQTT. El programa emplea un ClientID que es un hash MD5 truncado a 10 caracteres hexadecimales de la concatenación de la dirección MAC , el device file y el índice del cliente MQTT (empezando en 0).

La siguiente tabla muestra diferentes mensajes recibidos en el topic del cliente subscriptor en función de la situación reportada al ejecutar:

```
dragonrise -mqpub tcp://test.mosquitto.org:1883/topic12345 /dev/input/js0 2>/dev/null
```

Situación	Mensaje JSON recibido en topic cliente subscriptor

Conexión inicial al topic del broker	{"time":1590231059,"device":"js0", "event":{"type":1,"sensor":0,"value":1}, "switches":[1,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]} El broker transmite el último mensaje recibido por el broker desde el publicador (emitido con retention=true). Sirve p sensores de la tarjeta.
Evento en interruptores en una tarjeta conectada al bus USB y asociada con dispositivo de fichero "js0"	{"time":1590231550,"device":"js0", "event":{"type":1,"sensor":0,"value":0}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
Desconexión de la tarjeta del bus USB y asociada con dispositivo de fichero "js0"	{"time":1590231778,"device":"js0", "event":{"type":-1,"reason":"Dispositivo dejó de estar disponible"}}
Conexión de la tarjeta al bus USB y asociada con dispositivo de fichero "js0"	{"time":1590231857,"device":"js0", "event":{"type":0,"sensor":0,"value":0}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
Parada de programa dragonrise	{"time":0,"device":"js0", "event":{"type":-1,"reason":"Mensaje LWT emitido por broker. No disponible cliente MQTT que publ dispositivo"}} Broker detecta la pérdida de conexión y envía mensaje LWT a subscriptores
Arranque de programa con una tarjeta conectada al bus USB y asociada con dispositivo de fichero "js0"	{"time":1590232743,"device":"js0", "event":{"type":0,"sensor":0,"value":0}, "switches":[1,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
Caída de broker	(no llega mensaje)
Recuperación de broker	{"time":1590250662,"device":"js0", "event":{"type":1,"sensor":0,"value":1}, "switches":[1,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]} (retained, enviado por broker) ..... pasado un tiempo ...

	{"time":1590250996,"device":"js0", "event":{"type":1,"sensor":0,"value":0}, "switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0]} (enviado por dragonrise con último de esta

Con respecto los dos últimos casos:

1) El subscriptor (mosquito_sub) no se entera de que el broker se ha caído. ¿Es esto siempre así? ¿Formas de enterarse de la caída de la comunicación?
2) Cuando subscriptor reconecta recibe un mensaje "retained" (el último recibido por el broker). Aunque parece un evento normal, el subscriptor puede darse cuenta de que es un menaje viejo por el valor tiempo.
3) ¿Alguna formas de saber que es "retained"?
4) Más tarde , cuando el publicador logra reconectar, este envía un mensaje con el último mensaje almacenado en su fichero de estado, posiblemente con un estado diferente al del mensaje "retained" recibido anteriormente.
   Desde el punto de vista del subscriptor, es un evento de interruptor (tipo 1) pero con el reloj retrasado.
5) Formalmente, ¿No sería mejor que el publicador mandase (¿además inmediata y posteriormente?) un mensaje de inicialización con tiempo y estado actual?

Funcionamiento con brokers MQTT de pruebas

Se han realizado pruebas con diferentes brokers públicos de pruebas con los siguientes resultados:

Server	Broker	Transporte	Ports	Ok
test.mosquitto.org	Mosquitto	tcp	1883	**Ok**
		ssl	8883	**Ok**
		ssl [1]	8884	NP[2]
		ws	8080	**Ok**
		wss	8081	**Ok**
mqtt.eclipse.org		tcp	1883	**Ok**
tcp://broker.emqx.io		tcp	1883	**OK**

broker.hivemq.com	HiveMQ	tcp	1883	**OK**
		ssl	8000	**KO** [3]
broker.mqttdashboard.com[3]	HiveMQ	tcp	1883	**OK**
mqtt.flespi.io user: (Flespi token) umHbtbcvEQclGOglt0mvxn8j1kB4ubKNofMFgO1Ls3UYF07XOiictglkpKdHKNkY sin password	flsepi	tcp	1883	**Ok**
		Ssl	883	**Ok**
		ws	80	**Ok**
		wss	443	**OK**
test.mosca.io	mosca	tcp	1883	ND[4]
		ws	80	ND[4]

NOTAS:

1) Autenticación con certificado de cliente
2) NP: No probado
3) Network Error : EOF
4) ND: No disponible

Reglas UDEV para DragonRise

Con el fichero de reglas se pretende estabilizar el nombre del fichero de dispositivo en función de la posición en el bus USB. Esto es especialmente importante en el caso de que se conecten al mismo bus más de una tarjeta de tipo joystick que provoque la aparición de varios ficheros de dispositivos */dev/input/js<n>* sin que se pueda garantizar una correspondencia fija entre fichero de dispositivo y tarjeta.

Instalación manual de reglas

Se edita el fichero de reglas *31-dragonrise.rules*

```
sudo nano /etc/udev/rules.d/31-dragonrise.rules
```

con el siguiente contenido

```
# Fichero de reglas UDEV adecuadas pra Raspberry Pi
# para invocar inicializacion de tarjeta "Generic USB Joystick" de DragonRise el momento de su conexion/detección
# Autor Junav version 1 26/04/2020
# Este fichero debe ubicarse en /etc/udev/rules.d/

# La utilidad
# udevadm info -a -n /dev/input/js0
# reflejará
# KERNELS=="1-1.2" para placa DragonRise conectada directamente a puerto USB externo 1 de RPI (2 en el bus)
# KERNELS=="1-1.3" para placa DragonRise conectada directamente a puerto USB externo 2 de RPI (3 en el bus)
# Se hace notar que en una RPi el puerto USB 1 está empleado internamente por la tarjeta de red.
#
# Si se emplea un HUB, en lugar de una conexión directa, el tercer nivel de información despues del inicial "KERNEL"  daría
# KERNELS=="1-1.3.1" para placa conectada a puerto USB 1 del HUB en puerto USB externo 2 de RPI
# KERNELS=="1-1.3.3" para placa conectada a puerto USB 3 del HUB en puerto USB externo 2 de RPI

# En su caso ajustar estas reglas según sea preciso

# Deteccion de chip de DragonRise  idVendor=0079  idProduct=0006 asignando un symlink fijo segun numeracion bus USB, ejecucin de un programa con
parametro de numeracion
ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.2",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_2"
ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.3",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_3"
ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.4",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_4"
ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.5",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_5"

#Si se quiere ejecutar automaticamente el programa dragonrise (previamente ubicado en /usr/local/bin) se puese hacer aqui. Por ejemplo
#ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.5",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_5",
RUN+="/usr/local/bin/dragonrise -mqpub=tcp://host.dominio.dom:1883/topic_base /dev/dragonrise_5"

#Deteccion desenchufe en puerto USB de placa dragonrise y borrado de fichero de estado del programa dragonrise
ACTION=="remove", ENV{ID_VENDOR_ID}=="0079", ENV{ID_MODEL_ID}=="0006", ENV{DEVPATH}=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.2",
RUN+="/bin/rm /var/lib/dragonrise/dragonrise_2.dat"
ACTION=="remove", ENV{ID_VENDOR_ID}=="0079", ENV{ID_MODEL_ID}=="0006", ENV{DEVPATH}=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3",
RUN+="/bin/rm /var/lib/dragonrise/dragonrise_3.dat"
ACTION=="remove", ENV{ID_VENDOR_ID}=="0079", ENV{ID_MODEL_ID}=="0006", ENV{DEVPATH}=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4",
RUN+="/bin/rm /var/lib/dragonrise/dragonrise_4.dat"
ACTION=="remove", ENV{ID_VENDOR_ID}=="0079", ENV{ID_MODEL_ID}=="0006", ENV{DEVPATH}=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.5",
RUN+="/bin/rm /var/lib/dragonrise/dragonrise_5.dat"
```

Para asegurar la recarga de la reglas.

```
$ sudo udevadm control --reload-rules
```

En caso de que se necesite editar las reglas para adaptarse a un caso concreto (por ejemplo con presencia de concentradores-hubs USB), se puede consultar el Anexo I de este documento.

Ejecución como demonio/servicio systemd

Con permisos de root, se crea el script dragonrise_service.sh en /usr/bin/

```
$ sudo nano /usr/bin/dragonrise_service.sh
#!/bin/bash
# script de control de dragonrise como servicio systemD
# Autor junavarg version 1 04/07/2020
DATE=`date '+%Y-%m-%d %H:%M:%S'`
echo "dragonrise service service started at ${DATE}" | systemd-cat -p info

# se ejecuta dragonrise contra los 4 puertos USB externos de la Raspberry Pi
dragonrise -mqpub=tcp://test.mosquitto.org:1883/base_topic /dev/dragonrise_2 /dev/dragonrise_3 /dev/dragonrise_4 /dev/dragonrise_5
```

Se le dan permisos de ejecución

```
$ sudo chmod +x /usr/bin/ dragonrise_service.sh
```

Con permisos de root, se crea el fichero dragonrise.service en /lib/systemd/system/

```
$ sudo nano /lib/systemd/system/dragonrise.service
[Unit]
Description=dragonrise systemd service.

[Service]
Type=simple
ExecStart=/bin/bash /usr/bin/dragonrise_service.sh

[Install]
WantedBy=multi-user.target
```

Con permisos de root, se crea un enlace simbólico a este fichero en /etc/systemd/system/dragonrise.service

```
$ sudo ln -s /lib/systemd/system/dragonrise.service /etc/systemd/system/
```

Los siguientes comandos permiten respectivamente arrancar, comprobar estado, detener y rearrancar el servicio dragonrise.

```
sudo systemctl start dragonrise.service
sudo systemctl status dragonrise.service
sudo systemctl stop dragonrise.service
sudo systemctl restart dragonrise.service
```

Una vez comprobado su funcionamiento, ver siguiente apartado, se configura para que el servicio se ejecute automáticamente en cada arranque del sistema operativo.

```
sudo systemctl enable dragonrise.service
```

Finalmente se reboota la Raspberry Pi y se comprueba que arrancan automáticamente.

Para el uso del programa se deberá elegir y configurar adecuadamente: 1 el broker a emplear, 2) el protocolo de publicación, 3) el nombre del base_topic (no debe coincidir con topics de otros usuarios).

Subscripción al topic

Para comprobar o utilizar la publicación de eventos se pueden emplear cualquier cliente MQTT. A continuación se muestran dos ejemplos, uno con el cliente mosquitto (mosquitto_sub) y el otro con la aplicación para andorid MQTT Dash.

Configuración cliente MQTT Mosquitto

El proceso subscriptor se puede ejecutar en un ordenador con Linux. Se va a suponer que se ha configurado dragonrise para publicar en el broker test.mosquitto.org, mediante el puerto TCP 1883, con topic base “base_topic” y que la tarjeta USB Dragonrise se conecta al puerto externo 4 (interno 5) y que wl switch 1 está en estado cerrado y todos los demás en abierto. El siguiente comando y su resultado deben ser similar a:

```
$ mosquitto_sub -h test.mosquitto.org -t base_topic/dragonrise_5/event
{"time":1594013622,"device":"dragonrise_5","event":{"type":1,"sensor":0,"value":1},"switches":[1,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
```

Si se abre el conmutador en la tarjeta Dragonrise se obtendrá:

```
{"time":1594013973,"device":"dragonrise_5","event":{"type":1,"sensor":0,"value":0},"switches":[0,0,0,0,0,0,0,0,0,0,0,0],"axes":[0,0,0,0,0,0,0]}
```

Configuración App MQTT Dash para Android

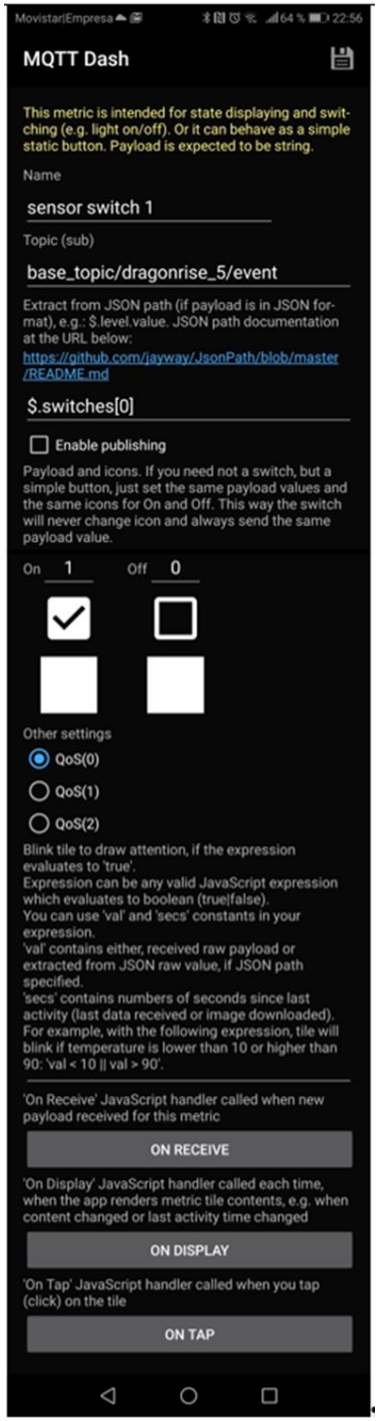
A continuación se muestra como configurar la aplicación MQTT Dash para Android

Pulsando en la parte superior derecha el símbolo más, se configura una nueva conexión a un broker MQTT como se indica en la siguiente figura.

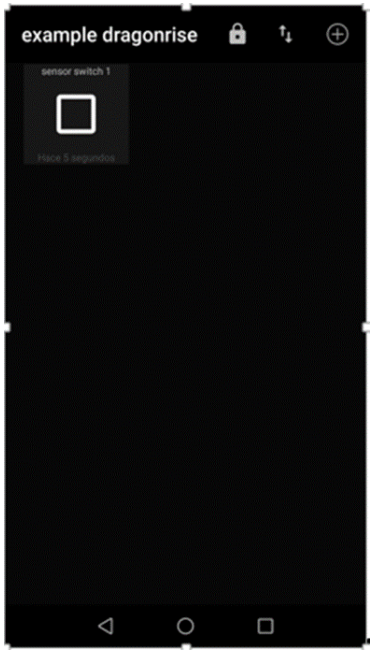

Movistar|Empresa
64 %
22:54
MQTT Dash
Default (automatically connect on start up). Note: this option is useful if you have just one connection configured.
If you have more than one connection, you can create home screen shortcut for every connection.
To create shortcut long press on any connection in connections list.
Keep screen on when connected to this broker
Allow metrics management. If disabled, you can't add, edit, delete or rearrange metrics. This serves as protection from unintentional metrics changing.
Name
example dragonrise
Address
test.mosquitto.org
Port
1883
Enable connection encryption (SSL/TLS). Note: if server certificate is self-signed, you need to install it to your device or enable option below, otherwise connection will fail. If server certificate issued by a known Certificate Authority (CA), it will work out of box, without installing to you device. Also don't forget, that MQTT servers have different ports for plain and SSL/TLS connections.
This broker uses self-signed SSL/TLS certificate. I trust this certificate at my own risk.
User name
User password
Client ID (must be unique)
mqttdash-97a6891e
Tile size
Small
Medium
Large
Metrics columns count for vertical orientation (0 - auto)
0
Metrics columns count for horizontal orientation (0 - auto)
0

Al finalizar se guarda la configuración pulsado en el símbolo “disquette”.

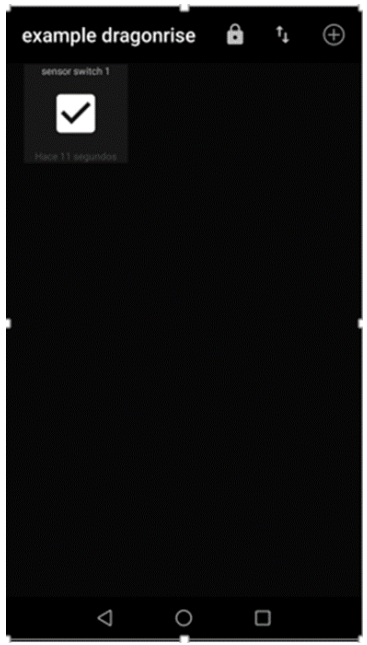
Pulsando en la recién creada conexión, se añade un nuevo elemento pulsando en el símbolo más en la parte superior derecha de la pantalla. Se elige un elemento de tipo Switch/button y se configura como sigue.


Movistar|Empresa
64 % 22:56
MQTT Dash
This metric is intended for state displaying and switching (e.g. light on/off). Or it can behave as a simple static button. Payload is expected to be string.
Name
sensor switch 1
Topic (sub)
base_topic/dragonrise_5/event
Extract from JSON path (if payload is in JSON format), e.g.: $.level.value. JSON path documentation at the URL below:
https://github.com/jayway/JsonPath/blob/master/README.md
$.switches[0]
Enable publishing
Payload and icons. If you need not a switch, but a simple button, just set the same payload values and the same icons for On and Off. This way the switch will never change icon and always send the same payload value.
On 1
Off 0
Other settings
QoS(0)
QoS(1)
QoS(2)
Blink tile to draw attention, if the expression evaluates to 'true'.
Expression can be any valid JavaScript expression which evaluates to boolean (true|false).
You can use 'val' and 'secs' constants in your expression.
'val' contains either, received raw payload or extracted from JSON raw value, if JSON path specified.
'secs' contains numbers of seconds since last activity (last data received or image downloaded).
For example, with the following expression, tile will blink if temperature is lower than 10 or higher than 90: 'val < 10 || val > 90'.
'On Receive' JavaScript handler called when new payload received for this metric
ON RECEIVE
'On Display' JavaScript handler called each time, when the app renders metric tile contents, e.g. when content changed or last activity time changed
ON DISPLAY
'On Tap' JavaScript handler called when you tap (click) on the tile
ON TAP

Se guarda la configuración del elemento recién creado pulsado en el símbolo “disquette” con lo que aparecerá el en interfaz de la App.



Una vez configurado se puede comprobar el funcionamiento abriendo y cerrando el interruptor número uno de la tarjeta Dragonrise.



Anexo I. Interfaz Linux para joystick */dev/input/js0*

En Linux, la tarjeta DragonRise se identifica en el bus USB como:

```
$ lsusb
...
Bus 001 Device 008: ID 0079:0006 DragonRise Inc. PC TWIN SHOCK Gamepad
...
```

El subdirectorio /dev/input contiene los ficheros de dispositivos de como teclados, ratones, tabletas, joystick, etc.

El fichero /proc/bus/input/devices contiene información relevante de estos periféricos de entrada y su correspondencia con los ficheros de dispositivos.

```
cat /proc/bus/input/devices
```

Más información en: https://thehackerdiary.wordpress.com/2017/04/21/exploring-devinput-1/

Los fichero de dispositivo con el formato *jsN* (por ejemplo */dev/input/js0*) están sujetos a la especificación Joystick API de Linux que se puede consultar en https://www.kernel.org/doc/Documentation/input/joystick-api.txt.

La lectura de este fichero de dispositivo se hace por bloques de 8 bytes que corresponde a un evento. Esta lectura queda bloqueada a la espera de un evento (cambio en un interruptor).

NO se precisa privilegios de root.

time				value		event type	axis/button number
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

Time corresponde a un timestamp en milisegundos desde “algún momento pasado”

Tipo de evento señala si el evento corresponde a un botón (interruptor) o en un eje de joystick:

0x01 -> botón

0x02 -> eje

Tras hacer el open del dispositivo las primeras lecturas corresponden a los estados actuales de todos los dispositivos. NO se trata de eventos reales sino “sintéticos”. Para diferenciarlos de los reales el valor del campo tipo de evento se combina en unión lógica con el valor 0x80, dando los valores

0x81 -> Estado inicial de botón

0x82 -> Estado inicial de eje

Numero de eje o número de botón. Identifica el botón (interruptor) o eje del evento. Botones y ejes llevan numeradores de identificación separados iniciados en 0.

Valor. Es el valor comunicado del botón o eje**.** Los dos bytes están en “little endian”.

Para botones:

0x0000 si abierto

0x0001 si cerrado

Para ejes/conmutadores:

0x0000 si centrado. En el caso de ejes implementado con conmutadores (no analógicos) ambos conmutadores de cada semieje abiertos (o ambos cerrados)

0x7FFF si máximo positivo (decimal 32767). En el caso de ejes implementado con conmutadores (no analógicos) conmutador de semieje positivo cerrado y conmutador de semieje negativo abierto.

0x8001 si máximo negativo (decimal -32767). En el caso de ejes implementado con conmutadores (no analógicos), conmutador de semieje negativo cerrado y conmutador de semieje positivo abierto.

En el caso de la tarjeta DragonRise, el driver reconoce 12 interruptores y 7 ejes/conmutadores.

En el modo normal se tiene visibilidad del eje 0 (eje X, AL-AR) y del eje 1 (eje Y AU-AD).

En el modo alternativo “point of view” estos conectores se expresan en los ejes 5 y 6 respectivamente.

El interruptor AU (Up) cerrado corresponde al máximo ¡¡negativo!! del eje Y.

El interruptor AD (Down) cerrado corresponde al máximo ¡¡positivo!! del eje Y.

El interruptor AR (Right) cerrado corresponde al máximo positivo del eje X.

El interruptor AL (Left) cerrado corresponde al máximo negativo del eje X.

NOTA. Se ha observado que, de forma aleatoria, la tarjeta deja escapar falsos eventos del inexistente eje 2 a valor 0.

Anexo II. Procedimiento de descubrimiento para construcción de reglas UDEV

El procedimiento es el siguiente:

Se conecta la tarjeta DragonRise en el puerto USB exterior número 3.

Se verifica la existencia de fichero de dispositivo */dev/input/js0* .

Se ejecuta el comando siguiente. Se señala en rojo la info relevante para la construcción de la regla UDEV

```
$ udevadm info -a -n /dev/input/js0

Udevadm info starts with the device specified by the devpath and then
walks up the chain of parent devices. It prints for every device
found, all possible attributes in the udev rules key format.
A rule to match, can be composed by the attributes of the device
and the attributes from one single parent device.
```

```
looking at device '/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4/1-1.4:1.0/0003:0079:0006.0008/input/input7/js
  KERNEL=="js0"
  SUBSYSTEM=="input"
  DRIVER==""

looking at parent device '/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4/1-1.4:1.0/0003:0079:0006.0008/input/in
  KERNELS=="input7"
  SUBSYSTEMS=="input"
  DRIVERS==""
  ATTRS{name}=="DragonRise Inc.   Generic   USB  Joystick  "
  ATTRS{phys}=="usb-3f980000.usb-1.4/input0"
  ATTRS{properties}=="0"
  ATTRS{uniq}==""

looking at parent device '/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4/1-1.4:1.0/0003:0079:0006.0008':
  KERNELS=="0003:0079:0006.0008"
  SUBSYSTEMS=="hid"
  DRIVERS=="dragonrise"
  ATTRS{country}=="21"

looking at parent device '/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4/1-1.4:1.0':
  KERNELS=="1-1.4:1.0"
  SUBSYSTEMS=="usb"
  DRIVERS=="usbhid"
  ATTRS{authorized}=="1"
  ATTRS{bAlternateSetting}==" 0"
  ATTRS{bInterfaceClass}=="03"
  ATTRS{bInterfaceNumber}=="00"
  ATTRS{bInterfaceProtocol}=="00"
  ATTRS{bInterfaceSubClass}=="00"
  ATTRS{bNumEndpoints}=="02"
  ATTRS{supports_autosuspend}=="1"

looking at parent device '/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4':
  KERNELS=="1-1.4"
  SUBSYSTEMS=="usb"
  DRIVERS=="usb"
  ATTRS{authorized}=="1"
  ATTRS{avoid_reset_quirk}=="0"
  ATTRS{bConfigurationValue}=="1"
  ATTRS{bDeviceClass}=="00"
  ATTRS{bDeviceProtocol}=="00"
  ATTRS{bDeviceSubClass}=="00"
  ATTRS{bMaxPacketSize0}=="8"
  ATTRS{bMaxPower}=="500mA"
```

```
    ATTRS{bNumConfigurations}=="1"
    ATTRS{bNumInterfaces}==" 1"
    ATTRS{bcdDevice}=="0107"
    ATTRS{bmAttributes}=="80"
    ATTRS{busnum}=="1"
    ATTRS{configuration}==""
    ATTRS{devnum}=="20"
    ATTRS{devpath}=="1.4"
    ATTRS{devspec}==" (null)"
    ATTRS{idProduct}=="0006"
    ATTRS{idVendor}=="0079"
    ATTRS{ltm_capable}=="no"
    ATTRS{manufacturer}=="DragonRise Inc.   "
    ATTRS{maxchild}=="0"
    ATTRS{product}=="Generic   USB  Joystick  "
    ATTRS{quirks}=="0x0"
    ATTRS{removable}=="removable"
    ATTRS{speed}=="1.5"
    ATTRS{urbnum}=="28247844"
    ATTRS{version}==" 1.00"

  looking at parent device '/devices/platform/soc/3f980000.usb/usb1/1-1':
    KERNELS=="1-1"
    SUBSYSTEMS=="usb"
    DRIVERS=="usb"
    ATTRS{authorized}=="1"
    ATTRS{avoid_reset_quirk}=="0"
    ATTRS{bConfigurationValue}=="1"
    ATTRS{bDeviceClass}=="09"
    ATTRS{bDeviceProtocol}=="02"
    ATTRS{bDeviceSubClass}=="00"
    ATTRS{bMaxPacketSize0}=="64"
    ATTRS{bMaxPower}=="2mA"
    ATTRS{bNumConfigurations}=="1"
    ATTRS{bNumInterfaces}==" 1"
    ATTRS{bcdDevice}=="0200"
    ATTRS{bmAttributes}=="e0"
    ATTRS{busnum}=="1"
    ATTRS{configuration}==""
    ATTRS{devnum}=="2"
    ATTRS{devpath}=="1"
    ATTRS{idProduct}=="9514"
    ATTRS{idVendor}=="0424"
    ATTRS{ltm_capable}=="no"
    ATTRS{maxchild}=="5"
    ATTRS{quirks}=="0x0"
```

```
  ATTRS{removable}=="unknown"
  ATTRS{speed}=="480"
  ATTRS{urbnum}=="549"
  ATTRS{version}==" 2.00"

looking at parent device '/devices/platform/soc/3f980000.usb/usb1':
  KERNELS=="usb1"
  SUBSYSTEMS=="usb"
  DRIVERS=="usb"
  ATTRS{authorized}=="1"
  ATTRS{authorized_default}=="1"
  ATTRS{avoid_reset_quirk}=="0"
  ATTRS{bConfigurationValue}=="1"
  ATTRS{bDeviceClass}=="09"
  ATTRS{bDeviceProtocol}=="01"
  ATTRS{bDeviceSubClass}=="00"
  ATTRS{bMaxPacketSize0}=="64"
  ATTRS{bMaxPower}=="0mA"
  ATTRS{bNumConfigurations}=="1"
  ATTRS{bNumInterfaces}==" 1"
  ATTRS{bcdDevice}=="0414"
  ATTRS{bmAttributes}=="e0"
  ATTRS{busnum}=="1"
  ATTRS{configuration}==""
  ATTRS{devnum}=="1"
  ATTRS{devpath}=="0"
  ATTRS{idProduct}=="0002"
  ATTRS{idVendor}=="1d6b"
  ATTRS{interface_authorized_default}=="1"
  ATTRS{ltm_capable}=="no"
  ATTRS{manufacturer}=="Linux 4.14.50-v7+ dwc_otg_hcd"
  ATTRS{maxchild}=="1"
  ATTRS{product}=="DWC OTG Controller"
  ATTRS{quirks}=="0x0"
  ATTRS{removable}=="unknown"
  ATTRS{serial}=="3f980000.usb"
  ATTRS{speed}=="480"
  ATTRS{urbnum}=="25"
  ATTRS{version}==" 2.00"

looking at parent device '/devices/platform/soc/3f980000.usb':
  KERNELS=="3f980000.usb"
  SUBSYSTEMS=="platform"
  DRIVERS=="dwc_otg"
  ATTRS{busconnected}=="Bus Connected = 0x1"
  ATTRS{buspower}=="Bus Power = 0x1"
```

```
    ATTRS{bussuspend}=="Bus Suspend = 0x0"
    ATTRS{devspeed}=="Device Speed = 0x0"
    ATTRS{driver_override}=="(null)"
    ATTRS{enumspeed}=="Device Enumeration Speed = 0x1"
    ATTRS{fr_interval}=="Frame Interval = 0x1d4b"
    ATTRS{ggpio}=="GGPIO = 0x00000000"
    ATTRS{gnptxfsiz}=="GNPTXFSIZ = 0x01000306"
    ATTRS{gotgctl}=="GOTGCTL = 0x001c0001"
    ATTRS{gpvndctl}=="GPVNDCTL = 0x00000000"
    ATTRS{grxfsiz}=="GRXFSIZ = 0x00000306"
    ATTRS{gsnpsid}=="GSNPSID = 0x4f54280a"
    ATTRS{guid}=="GUID = 0x2708a000"
    ATTRS{gusbcfg}=="GUSBCFG = 0x20001700"
    ATTRS{hcd_frrem}=="HCD Dump Frame Remaining"
    ATTRS{hcddump}=="HCD Dump"
    ATTRS{hnp}=="HstNegScs = 0x0"
    ATTRS{hnpcapable}=="HNPCapable = 0x1"
    ATTRS{hprt0}=="HPRT0 = 0x00001405"
    ATTRS{hptxfsiz}=="HPTXFSIZ = 0x02000406"
    ATTRS{hsic_connect}=="HSIC Connect = 0x1"
    ATTRS{inv_sel_hsic}=="Invert Select HSIC = 0x0"
    ATTRS{mode}=="Mode = 0x1"
    ATTRS{mode_ch_tim_en}=="Mode Change Ready Timer Enable = 0x0"
    ATTRS{rd_reg_test}=="Time to read GNPTXFSIZ reg 10000000 times: 940 msecs (94 jiffies)"
    ATTRS{regdump}=="Register Dump"
    ATTRS{regoffset}=="0xffffffff"
    ATTRS{regvalue}=="invalid offset"
    ATTRS{rem_wakeup_pwrdn}=""
    ATTRS{remote_wakeup}=="Remote Wakeup Sig = 0 Enabled = 0 LPM Remote Wakeup = 0"
    ATTRS{spramdump}=="SPRAM Dump"
    ATTRS{srp}=="SesReqScs = 0x1"
    ATTRS{srpcapable}=="SRPCapable = 0x1"
    ATTRS{wr_reg_test}=="Time to write GNPTXFSIZ reg 10000000 times: 350 msecs (35 jiffies)"

  looking at parent device '/devices/platform/soc':
    KERNELS=="soc"
    SUBSYSTEMS=="platform"
    DRIVERS==""
    ATTRS{driver_override}=="(null)"

  looking at parent device '/devices/platform':
    KERNELS=="platform"
    SUBSYSTEMS==""
    DRIVERS==""
```

Se construye un fichero de reglas en

```
$ sudo nano /etc/udev/rules.d/31-dragonrise.rules
...
# Deteccion de chip de Dragonrise  idVendor=0079  idProduct=0006 asignando un symlink fijo segun numeracion bus USB, ejecucin de un programa con parametro de numeracion
ACTION=="add",  KERNEL=="js*", KERNELS=="1-1.4",   ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", SYMLINK+="dragonrise_4",
RUN+="/usr/local/bin/dragonrise 4"
```

Para asegurar la recarga de la reglas (no es necesario si se reditan).

```
$ sudo udevadm control --reload-rules
```

Se inserta en el puerto USB exterior 3 (4 en el bus) la tarjeta DragonRise y se comprueba la existencia del enlace simbólico.

```
ls -l /dev
...
lrwxrwxrwx 1 root root           9 abr 26 11:21 dragonrise_4 -> input/js0
...
```

Anexo III. Interfaz software con */dev/hdiraw0*

Otra posibilidad de obtener el estado de los interruptores conectados a la tarjeta DragonRise es el empleo del fichero de dispositivo /dev/hdiraw0. De esto este fue el primer intento pero se descartó por encontrar más conveniente el interfaz proporcionado por /dev/input/js0.

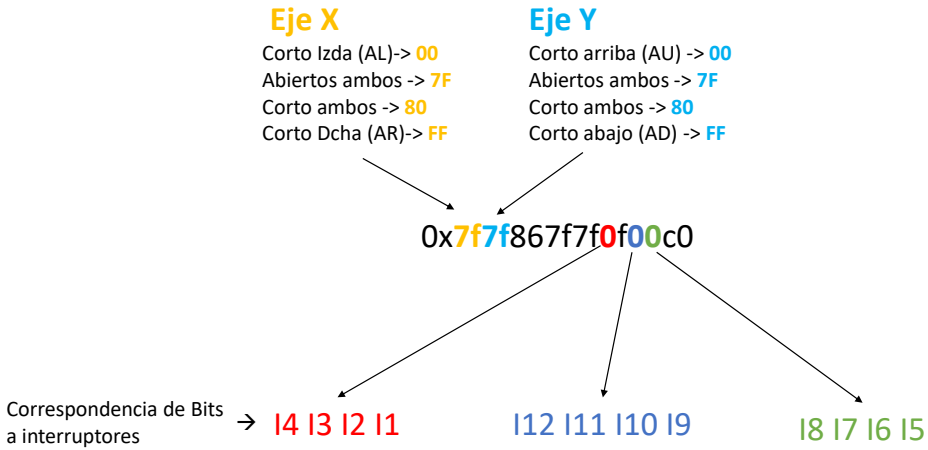
A la detección el sistema crea un fichero de dispositivo /dev/hidraw$\mathbb{N}$ (Human Interface Device RAW), donde "$\mathbb{N}$" típicamente será "0" si no hay otros dispositivos.

```
$ ls -l /dev/hid*
crw------- 1 root root 247, 0 abr 19 06:40 /dev/hidraw0
```

Obsérvese que solo permite leer y escribir a root.

https://www.kernel.org/doc/Documentation/hid/hidraw.txt

Mediante un programa que, con privilegios de root, lea constantemente 8 bytes de este dispositivo y los exprese en consola en hexadecimal se obtiene algo similar a:



Así por ejemplo:

si se mantiene cortocircuitado solo el I1, se obtiene: 0x7f7f867f7f1f00c0

si se mantiene cortocircuitado solo el I8, se obtiene: 0x7f7f867f7f0f08c0

si se mantiene cortocircuitado solo el I11, se obtiene: 0x7f7f867f7f0f40c0

El empleo de este fichero dispositivo para la lectura de los eventos tropieza con los siguientes inconvenientes:

- No parece un sistema standard. Otros dispositivos similares pueden tener una interpretación diferente de los bytes leidos.
- La lectura no se bloquea a la espera de eventos. Se producen múltiples lecturas que hay que ignorar a la espera de un cambio que signifique un evente. Esto supone un gasto de CPU (en torno a 2%) excesivo para esta función.
- El proceso de lectura hay que hacerlo con privilegios de root.