Force feedback for Linux.

By Johann Deneux <johann.deneux@gmail.com> on 2001/04/22.
Updated by Anssi Hannula <anssi.hannula@gmail.com> on 2006/04/09.

----

1. Introduction

This document describes how to use force feedback devices under Linux. The goal is not to support these devices as if they were simple input-only devices (as it is already the case), but to really enable the rendering of force effects.
This document only describes the force feedback part of the Linux input interface. Please read joystick.txt and input.txt before reading further this document.

2. Instructions to the user

To enable force feedback, you have to:

1. have your kernel configured with evdev and a driver that supports your device.
2. make sure evdev module is loaded and /dev/input/event* device files are created.

Before you start, let me WARN you that some devices shake violently during the initialisation phase. This happens for example with my "AVB Top Shot Pegasus". To stop this annoying behaviour, move you joystick to its limits. Anyway, you should keep a hand on your device, in order to avoid it to break down if something goes wrong.

If you have a serial iforce device, you need to start inputattach. See joystick.txt for details.

2.1 Does it work ?

There is an utility called fftest that will allow you to test the driver.

```
% fftest /dev/input/eventXX
```

3. Instructions to the developer

All interactions are done using the event API. That is, you can use ioctl() and write() on /dev/input/eventXX.
This information is subject to change.

3.1 Querying device capabilities

```
#include <linux/input.h>
#include <sys/ioctl.h>

unsigned long features[1 + FF_MAX/sizeof(unsigned long)];
int ioctl(int file_descriptor, int request, unsigned long *features);
```

"request" must be EVIOCGBIT(EV_FF, size of features array in bytes)

Returns the features supported by the device. features is a bitfield with the following bits:

- FF_CONSTANT	can render constant force effects
- FF_PERIODIC	can render periodic effects with the following waveforms:
- FF_SQUARE	square waveform
- FF_TRIANGLE	triangle waveform
- FF_SINE	sine waveform
- FF_SAW_UP	sawtooth up waveform
- FF_SAW_DOWN	sawtooth down waveform
- FF_CUSTOM	custom waveform
- FF_RAMP	can render ramp effects
- FF_SPRING	can simulate the presence of a spring
- FF_FRICTION	can simulate friction
- FF_DAMPER	can simulate damper effects
- FF_RUMBLE	rumble effects
- FF_INERTIA	can simulate inertia
- FF_GAIN	gain is adjustable
- FF_AUTOCENTER	autocenter is adjustable

Note: In most cases you should use FF_PERIODIC instead of FF_RUMBLE. All devices that support FF_RUMBLE support FF_PERIODIC (square, triangle, sine) and the other way around.

Note: The exact syntax FF_CUSTOM is undefined for the time being as no driver supports it yet.

```
int ioctl(int fd, EVIOCGEFFECTS, int *n);
```

Returns the number of effects the device can keep in its memory.

3.2 Uploading effects to the device

```
#include <linux/input.h>
#include <sys/ioctl.h>

int ioctl(int file_descriptor, int request, struct ff_effect *effect);
```

"request" must be EVIOCSFF.

"effect" points to a structure describing the effect to upload. The effect is uploaded, but not played.
The content of effect may be modified. In particular, its field "id" is set to the unique id assigned by the driver. This data is required for performing some operations (removing an effect, controlling the playback).
This if field must be set to -1 by the user in order to tell the driver to allocate a new effect.

Effects are file descriptor specific.

See <linux/input.h> for a description of the ff_effect struct. You should also find help in a few sketches, contained in files shape.fig and interactive.fig. You need xfig to visualize these files.

3.3 Removing an effect from the device

```
int ioctl(int fd, EVIOCRMFF, effect.id);
```

This makes room for new effects in the device's memory. Note that this also stops the effect if it was playing.

3.4 Controlling the playback of effects

Control of playing is done with write(). Below is an example:

```
#include <linux/input.h>
#include <unistd.h>

	struct input_event play;
	struct input_event stop;
	struct ff_effect effect;
	int fd;
...
	fd = open("/dev/input/eventXX", O_RDWR);
...
	/* Play three times */
	play.type = EV_FF;
	play.code = effect.id;
	play.value = 3;

	write(fd, (const void*) &play, sizeof(play));
...
	/* Stop an effect */
	stop.type = EV_FF;
	stop.code = effect.id;
	stop.value = 0;

	write(fd, (const void*) &play, sizeof(stop));
```

3.5 Setting the gain

Not all devices have the same strength. Therefore, users should set a gain factor depending on how strong they want effects to be. This setting is persistent across access to the driver.

```
/* Set the gain of the device
int gain;		/* between 0 and 100 */
struct input_event ie;	/* structure used to communicate with the driver */

ie.type = EV_FF;
ie.code = FF_GAIN;
ie.value = 0xFFFFUL * gain / 100;

if (write(fd, &ie, sizeof(ie)) == -1)
	perror("set gain");
```

3.6 Enabling/Disabling autocenter

The autocenter feature quite disturbs the rendering of effects in my opinion, and I think it should be an effect, which computation depends on the game type. But you can enable it if you want.

```
int autocenter;         /* between 0 and 100 */
struct input_event ie;

ie.type = EV_FF;
ie.code = FF_AUTOCENTER;
ie.value = 0xFFFFUL * autocenter / 100;

if (write(fd, &ie, sizeof(ie)) == -1)
        perror("set auto-center");
```

A value of 0 means "no auto-center".

3.7 Dynamic update of an effect

Proceed as if you wanted to upload a new effect, except that instead of setting the id field to -1, you set it to the wanted effect id.
Normally, the effect is not stopped and restarted. However, depending on the type of device, not all parameters can be dynamically updated. For example, the direction of an effect cannot be updated with iforce devices. In this case, the driver stops the effect, up-load it, and restart it.

Therefore it is recommended to dynamically change direction while the effect is playing only when it is ok to restart the effect with a replay count of 1.

3.8 Information about the status of effects

Every time the status of an effect is changed, an event is sent. The values and meanings of the fields of the event are as follows:

```
struct input_event {
/* When the status of the effect changed */
        struct timeval time;

/* Set to EV_FF_STATUS */
        unsigned short type;

/* Contains the id of the effect */
        unsigned short code;

/* Indicates the status */
        unsigned int value;
};
```

FF_STATUS_STOPPED	The effect stopped playing
FF_STATUS_PLAYING	The effect started to play

NOTE: Status feedback is only supported by iforce driver. If you have a really good reason to use this, please contact linux-joystick@atrey.karlin.mff.cuni.cz or anssi.hannula@gmail.com so that support for it can be added to the rest of the drivers.