# Introduction

LIRC stands for Linux Infrared Remote Control. The LIRC device interface is a bi-directional interface for transporting raw IR and decoded scancodes data between userspace and kernelspace. Fundamentally, it is just a chardev (/dev/lircX, for X = 0, 1, 2, ...), with a number of standard struct file_operations defined on it. With respect to transporting raw IR and decoded scancodes to and fro, the essential fops are read, write and ioctl.

It is also possible to attach a BPF program to a LIRC device for decoding raw IR into scancodes.

Example dmesg output upon a driver registering w/LIRC:

**System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 21)**

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    $ dmesg |grep lirc_dev
    rc rc0: lirc_dev: driver mceusb registered at minor = 0, raw IR receiver, raw IR transmitter
```

What you should see for a chardev:

**System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 28)**

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    $ ls -l /dev/lirc*
    crw-rw---- 1 root root 248, 0 Jul 2 22:20 /dev/lirc0
```

Note that the package v4l-utils contains tools for working with LIRC devices:

- ir-ctl: can receive raw IR and transmit IR, as well as query LIRC device features.
- ir-keytable: can load keymaps; allows you to set IR kernel protocols; load BPF IR decoders and test IR decoding. Some BPF IR decoders are also provided.

# LIRC modes

LIRC supports some modes of receiving and sending IR codes, as shown on the following table.

`LIRC_MODE_SCANCODE`

This mode is for both sending and receiving IR.

For transmitting (aka sending), create a struct lirc_scancode with the desired scancode set in the `scancode` member, :c:type:`rc_proto` set to the :ref:`IR protocol <Remote_controllers_Protocols>`, and all other members set to 0. Write this struct to the lirc device.

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 60); *backlink***

Unknown interpreted text role "c:type".

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 60); *backlink***

Unknown interpreted text role "ref".

For receiving, you read struct lirc_scancode from the LIRC device. The `scancode` field is set to the received scancode and the :ref:`IR protocol <Remote_controllers_Protocols>` is set in :c:type:`rc_proto`. If the scancode maps to a valid key code, this is set in the `keycode` field, else it is set to `KEY_RESERVED`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 65); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 65); *backlink*

Unknown interpreted text role "c:type".

The `flags` can have `LIRC_SCANCODE_FLAG_TOGGLE` set if the toggle bit is set in protocols that support it (e.g. rc-5 and rc-6), or `LIRC_SCANCODE_FLAG_REPEAT` for when a repeat is received for protocols that support it (e.g. nec).

In the Sanyo and NEC protocol, if you hold a button on remote, rather than repeating the entire scancode, the remote sends a shorter message with no scancode, which just means button is held, a "repeat". When this is received, the `LIRC_SCANCODE_FLAG_REPEAT` is set and the scancode and keycode is repeated.

With nec, there is no way to distinguish "button hold" from "repeatedly pressing the same button". The rc-5 and rc-6 protocols have a toggle bit. When a button is released and pressed again, the toggle bit is inverted. If the toggle bit is set, the `LIRC_SCANCODE_FLAG_TOGGLE` is set.

The `timestamp` field is filled with the time nanoseconds (in `CLOCK_MONOTONIC`) when the scancode was decoded.

`LIRC_MODE_MODE2`

The driver returns a sequence of pulse and space codes to userspace, as a series of u32 values.

This mode is used only for IR receive.

The upper 8 bits determine the packet type, and the lower 24 bits the payload. Use `LIRC_VALUE()` macro to get the payload, and the macro `LIRC_MODE2()` will give you the type, which is one of:

`LIRC_MODE2_PULSE`

Signifies the presence of IR in microseconds, also known as *flash*.

`LIRC_MODE2_SPACE`

Signifies absence of IR in microseconds, also known as *gap*.

`LIRC_MODE2_FREQUENCY`

If measurement of the carrier frequency was enabled with :ref:`lirc_set_measure_carrier_mode` then this packet gives you the carrier frequency in Hertz.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 114); *backlink*

Unknown interpreted text role "ref".

`LIRC_MODE2_TIMEOUT`

When the timeout set with :ref:`lirc_set_rec_timeout` expires due to no IR being detected, this packet will be sent, with the number of microseconds with no IR.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 120); *backlink*

Unknown interpreted text role "ref".

`LIRC_MODE2_OVERFLOW`

Signifies that the IR receiver encounter an overflow, and some IR is missing. The IR data after this should be correct again. The actual value is not important, but this is set to 0xffffff by the kernel for compatibility with lircd.

LIRC_MODE_PULSE

In pulse mode, a sequence of pulse/space integer values are written to the lirc device using :ref:`lirc-write`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 135); *backlink*
>
> Unknown interpreted text role "ref".

The values are alternating pulse and space lengths, in microseconds. The first and last entry must be a pulse, so there must be an odd number of entries.

This mode is used only for IR send.

## Data types used by LIRC_MODE_SCANCODE

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 148)
>
> Unknown directive type "kernel-doc".
>
> ```
> .. kernel-doc:: include/uapi/linux/lirc.h
>     :identifiers: lirc_scancode rc_proto
> ```

## BPF based IR decoder

The kernel has support for decoding the most common :ref:`IR protocols <Remote_controllers_Protocols>`, but there are many protocols which are not supported. To support these, it is possible to load an BPF program which does the decoding. This can only be done on LIRC devices which support reading raw IR.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 155); *backlink*
>
> Unknown interpreted text role "ref".

First, using the bpf(2) syscall with the BPF_LOAD_PROG argument, program must be loaded of type BPF_PROG_TYPE_LIRC_MODE2. Once attached to the LIRC device, this program will be called for each pulse, space or timeout event on the LIRC device. The context for the BPF program is a pointer to a unsigned int, which is a :ref:`LIRC_MODE_MODE2 <lirc-mode-mode2>` value. When the program has decoded the scancode, it can be submitted using the BPF functions bpf_rc_keydown() or bpf_rc_repeat(). Mouse or pointer movements can be reported using bpf_rc_pointer_rel().

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\(linux-master)(Documentation)(userspace-api)(media)(rc)lirc-dev-intro.rst, line 161); *backlink*
>
> Unknown interpreted text role "ref".

Once you have the file descriptor for the BPF_PROG_TYPE_LIRC_MODE2 BPF program, it can be attached to the LIRC device using the bpf(2) syscall. The target must be the file descriptor for the LIRC device, and the attach type must be BPF_LIRC_MODE2. No more than 64 BPF programs can be attached to a single LIRC device at a time.