Adafruit IRREMOTE Library Documentation

Release 1.0

Scott Shawcroft

Jun 07, 2021

Contents

CircuitPython driver for use with IR Receivers.

Examples of products to use this library with:

- Circuit Playground Express
- IR Receiver Sensor

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-irremote
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-irremote
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-irremote
```

CHAPTER 3

Usage Example

```
# Circuit Playground Express Demo Code
# Adjust the pulseio 'board.PIN' if using something else
import pulseio
import board
import adafruit_irremote

pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=120, idle_state=True)
decoder = adafruit_irremote.GenericDecode()


while True:
    pulses = decoder.read_pulses(pulsein)
    print("Heard", len(pulses), "Pulses:", pulses)
    try:
        code = decoder.decode_bits(pulses)
        print("Decoded:", code)
    except adafruit_irremote.IRNECRepeatException:  # unusual short code!
        print("NEC repeat!")
    except adafruit_irremote.IRDecodeException as e:     # failed to decode
        print("Failed to decode: ", e.args)

    print("----------------------------")
```

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/irremote_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Circuit Playground Express Demo Code
# Adjust the pulseio 'board.PIN' if using something else
import pulseio
import board
import adafruit_irremote

pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=120, idle_state=True)
decoder = adafruit_irremote.GenericDecode()


while True:
    pulses = decoder.read_pulses(pulsein)
    print("Heard", len(pulses), "Pulses:", pulses)
    try:
        code = decoder.decode_bits(pulses)
        print("Decoded:", code)
    except adafruit_irremote.IRNECRepeatException:  # unusual short code!
        print("NEC repeat!")
    except adafruit_irremote.IRDecodeException as e:  # failed to decode
        print("Failed to decode: ", e.args)

    print("----------------------------")
```

6.2 adafruit_irremote

Demo code for Circuit Playground Express:

```
# Circuit Playground Express Demo Code
# Adjust the pulseio 'board.PIN' if using something else
import pulseio
import board
import adafruit_irremote

pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=120, idle_state=True)
decoder = adafruit_irremote.GenericDecode()


while True:
    pulses = decoder.read_pulses(pulsein)
    print("Heard", len(pulses), "Pulses:", pulses)
    try:
        code = decoder.decode_bits(pulses)
        print("Decoded:", code)
    except adafruit_irremote.IRNECRepeatException:  # unusual short code!
        print("NEC repeat!")
    except adafruit_irremote.IRDecodeException as e:     # failed to decode
        print("Failed to decode: ", e.args)

    print("----------------------------")
```

- Author(s): Scott Shawcroft

6.2.1 Implementation Notes

Hardware:

- CircuitPlayground Express
- IR Receiver Sensor

Software and Dependencies:

- Adafruit CircuitPython firmware for the ESP8622 and M0-based boards: https://github.com/adafruit/circuitpython/releases

exception adafruit_irremote.**FailedToDecode**
Raised by decode_bits. Error argument is UnparseableIRMessage

class adafruit_irremote.**GenericDecode**
Generic decoding of infrared signals

bin_data(*pulses*)
Wraps the top-level function bin_data for backward-compatibility.

decode_bits(*pulses*)
Wraps the top-level function decode_bits for backward-compatibility.

read_pulses(*input_pulses*, ***, *max_pulse=10000*, *blocking=True*, *pulse_window=0.1*, *blocking_delay=0.1*)
Read out a burst of pulses until pulses stop for a specified period (pulse_window), pruning pulses after a pulse longer than `max_pulse`.

Parameters

- **input_pulses** (*PulseIn*) – Object to read pulses from
- **max_pulse** (*int*) – Pulse duration to end a burst
- **blocking** (*bool*) – If True, will block until pulses found. If False, will return None if no pulses. Defaults to True for backwards compatibility
- **pulse_window** (*float*) – pulses are collected for this period of time
- **blocking_delay** (*float*) – delay between pulse checks when blocking

class adafruit_irremote.**GenericTransmit**(*header*, *one*, *zero*, *trail*, ***, *debug=False*)
Generic infrared transmit class that handles encoding.

Parameters

- **header** (*int*) – The length of header in microseconds
- **one** (*int*) – The length of a one in microseconds
- **zero** (*int*) – The length of a zero in microseconds
- **trail** (*int*) – The length of the trail in microseconds, set to None to disable
- **debug** (*bool*) – Enable debug output, default False

transmit(*pulseout*, *data*, ***, *repeat=0*, *delay=0*, *nbits=None*)
Transmit the `data` using the `pulseout`.

Parameters

- **pulseout** (*pulseio.PulseOut*) – PulseOut to transmit on
- **data** (*bytearray*) – Data to transmit
- **repeat** (*int*) – Number of additional retransmissions of the data, default 0
- **delay** (*float*) – Delay between any retransmissions, default 0
- **nbits** (*int*) – Optional number of bits to send, useful to send fewer bits than in the data bytes

exception adafruit_irremote.**IRDecodeException**
Generic decode exception

class adafruit_irremote.**IRMessage**(*pulses*, *code*)
Pulses and the code they were parsed into

code
Alias for field number 1

pulses
Alias for field number 0

exception adafruit_irremote.**IRNECRepeatException**
Exception when a NEC repeat is decoded

class adafruit_irremote.**NECRepeatIRMessage**(*pulses*)
Pulses interpreted as an NEC repeat code

pulses
Alias for field number 0

class adafruit_irremote.**NonblockingGenericDecode**(*pulses*, *max_pulse=10000*)
Decode pulses into bytes in a non-blocking fashion.

Parameters

- **input_pulses** (*PulseIn*) – Object to read pulses from
- **max_pulse** (*int*) – Pulse duration to end a burst. Units are microseconds.

```
>>> pulses = PulseIn(...)
>>> decoder = NonblockingGenericDecoder(pulses)
>>> for message in decoder.read():
...     if isinstace(message, IRMessage):
...         message.code  # TA-DA! Do something with this in your application.
...     else:
...         # message is either NECRepeatIRMessage or
...         # UnparseableIRMessage. You may decide to ignore it, raise
...         # an error, or log the issue to a file. If you raise or log,
...         # it may be helpful to include message.pulses in the error message.
...         ...
```

read()
Consume all pulses from PulseIn. Yield decoded messages, if any.

If a partial message is received, this does not block to wait for the rest. It stashes the partial message, to be continued the next time it is called.

adafruit_irremote.**UnparseableIRMessage**
Pulses and the reason that they could not be parsed into a code

alias of *adafruit_irremote.IRMessage*

adafruit_irremote.**bin_data**(*pulses*)
Compute bins of pulse lengths where pulses are +-25% of the average.

Parameters **pulses** (*list*) – Input pulse lengths

adafruit_irremote.**decode_bits**(*pulses*)
Decode the pulses into bits.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index