LIRC Python Package

Release 0.1.0

Jul 16, 2020

Up Running

This is a python package that allows you to interact with the daemon in the Linux Infrared Remote Control package. Interacting with the daemon allows you to send IR signals from a computer.

More information on the lircd daemon, socket interface, reply packet format, etc. can be found at https://www.lirc.org/html/lircd.html

CHAPTER 1

Installation

This package is hosted on PyPI and can be installed through pip.

```
$ pip install lirc
```

However since this is a wrapper around the LIRC daemon, it is expected that LIRC is installed and setup on the given system as well.

More information on that can be found in the installation portion of the documentation.

CHAPTER 2

Using the Lirc Package

```
from lirc import Lirc

lirc = Lirc()
response = lirc.version()

print(response.command)
>>> 'VERSION'
print(response.success)
>>> True
print(response.data)
>>> ['0.10.1']
```

To get started with the package, we import `Lirc` and can initialize it with the defaults by passing it no arguments.

This will assume a socket path of `/var/run/lirc/lircd`. Furthermore, this will also then assume a socket connection using `AF_UNIX` and `SOCK_STREAM`. These are both the defaults that should work on a Linux system. There are ports of LIRC to Windows and macOS but using the package there is far less common. However, both of these are configurable through options passed to `Lirc` to allow it to be used on those operating systems as well.

After sending any command to the LIRC daemon, this package will create a `LircResponse` for us that it returns. That response contains the command we sent to LIRC, whether it was successful, and any data that was returned back to us.

CHAPTER 3

Further Documentation

More information on how to setup the system installed LIRC, how to use this python library, and a full API specification can be found at https://lirc.readthedocs.io/

CHAPTER 4

Site Documentation

4.1 Installing the LIRC Python & System Package

Since this package is merely a wrapper around the LIRC daemon, it is expected that LIRC is installed and setup on the given system as well to be able to use the python package.

4.1.1 Python Package

This package is hosted on PyPI and can be installed through pip.

```
$ pip install lirc
```

4.1.2 System LIRC Package

While there are ports of LIRC to macOS and Windows, the original Linux version is generally easier to get working and install.

Linux:

- It is highly likely that the package manager on your system already has LIRC packaged up and ready to be installed for you. e.g. `sudo apt install lirc` on Ubuntu.
- If not, you may have to compile and install it manually, but I would avoid that if possible.

Windows:

- WinLIRC at http://winlirc.sourceforge.net/ is a port for Windows. It works a bit differently since it is just a collection of files in a folder that you run so you'll have to adjust the `socket` and `socket_path` parameter. More information on that can be found at using LIRC on Windows.

macOS:

- There is a port on MacPorts at https://ports.macports.org/port/lirc/summary with it's source code on GitHub at https://github.com/andyvand/LIRC. However, it doesn't appear to be maintained any longer and is not the latest LIRC version. You can then run `port install lirc` or build the package from source using the instructions on the README of the GitHub repository. See using LIRC on macOS for more information on getting LIRC setup on macOS and how to use this python package with it.

4.2 Hardware Setup

In order to use this package, you'll need to have an IR emitter or transciever hooked up to your computer.

4.3 Configuring System LIRC

Once you have your IR emitter or transciever hooked up to your computer, you'll want to configure the system installed LIRC to ensure it works for emitting IR.

You'll also want to ensure you have a configuration file for the remote control that you want to emulate when emitting IR since whatever you're sending IR to may only understand IR codes from certain remotes.

4.3.1 Linux

4.3.2 Windows

4.3.3 macOS

4.4 Usage

Once you've installed the `lirc` python package, there will be a number of things you can now import from it to get started.

```
from lirc import (
  Lirc,
  LircResponse,
  LircError,
  LircSocketError,
  LircSocketTimeoutError,
  InvalidReplyPacketFormatError
)
```

The most relevant of these is `Lirc`, since this is the main class you will be using.

4.4.1 Initializing Lirc

The `Lirc` class takes in three separate options, which all have default values, that we may pass into it to construct it and override those default values.

The simplest way to construct `Lirc` is with no arguments at all.

```
from lirc import Lirc

lirc = Lirc()
```

This will attempt to connect to the lircd socket at "/var/run/lirc/lircd" on your system, create a socket using `AF_UNIX` and `SOCK_STREAM`, and sets a socket timeout of 5 seconds.

4.5 API Specification

class lirc.**Lirc**(*socket_path: str = '/var/run/lirc/lircd'*, *socket: socket.socket = <socket.socket fd=3, family=AddressFamily.AF_UNIX, type=SocketKind.SOCK_STREAM, proto=0>*, *socket_timeout: int = 5*)

Bases: `object`

Communicate with the lircd daemon.

DEFAULT_SOCKET_PATH = '/var/run/lirc/lircd'

ENCODING = 'utf-8'

list_remote_keys(*remote: str*) → lirc.response.LircResponse

List all the keys for a specific remote.

Parameters **remote** – The remote to list the keys of.

Returns The response of the command.

list_remotes() → lirc.response.LircResponse

List all the remotes in LIRC

Returns The response of the command.

send_once(*key: str*, *remote: str*, *repeat_count: int = 1*) → Union[lirc.response.LircResponse, List[lirc.response.LircResponse]]

Send an LIRC SEND_ONCE command.

Structure of the command:

- SEND_ONCE <remote-name> <key-name-from-remote-file> [repeat-count]

Parameters

- **key** – The name of the key to send.
- **remote** – The remote to use keys from.
- **repeat_count** – The number of times to press this key.

Returns a response from the command or a list of those responses if repeat_count > 1.

send_start(*key: str*, *remote: str*) → lirc.response.LircResponse

Send an LIRC SEND_START command.

Structure of the command:

- SEND_START <remote-name> <key-name-from-remote-file>

Parameters

- **key** – The name of the key to start sending.
- **remote** – The remote to use keys from.

Returns The response of the command.

send_stop(*key: str*, *remote: str*) → lirc.response.LircResponse

Send an LIRC SEND_STOP command.

Structure of the command:

- SEND_STOP <remote-name> <key-name-from-remote-file>

Parameters

- **key** – The name of the key to start sending.
- **remote** – The remote to use keys from.

Returns The response of the command.

set_inputlog(*path: str*) → lirc.response.LircResponse
Set the path to log all lircd received data to.

Returns The response of the command.

stop_inputlog() → lirc.response.LircResponse
Stop logging to the inputlog path from set_inputlog.

Returns The response of the command.

version() → lirc.response.LircResponse
Retrieve the version of LIRC

Returns The response of the command with the version in the data field.

class lirc.**LircResponse**(*command: str*, *success: bool*, *data: list*)
Bases: object

A response from the LIRC daemon. Stores the command that had been sent, whether or not it was successful, and the parsed reply packet data.

exception lirc.**LircError**
Bases: Exception

A generic error that comes from this package.

exception lirc.**LircSocketError**
Bases: lirc.exceptions.LircError

For when a generic error occurs with the lircd socket

exception lirc.**LircSocketTimeoutError**
Bases: lirc.exceptions.LircSocketError

For when a timeout error occurs with the socket. This can happen when recv does not find any data for a given amount of time.

exception lirc.**InvalidReplyPacketFormatError**
Bases: lirc.exceptions.LircError

The reply packet from LIRC was in an invalid format.

4.6 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

4.6.1 0.1.0 - 2020-07-13

- Initial Release

4.7 Contributor Covenant Code of Conduct

4.7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.7.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at eugenetriguba@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

4.8 MIT License

4.9 Contributing Guidelines

Thank you for your interest! If you find a bug, want to suggest an improvement, or any other change, please open a issue first. This ensures your time is not wasted if you were planning on creating a pull request, but the changes suggested do not work for this project.

4.9.1 General Guidelines

Commit history:

- Try to keep a clean commit history so it is easier to see your changes. Keep functional changes and refactorings in separate commits.

Commit messages:

- Have a short one line summary of your change followed by as many paragraphs of explanation as you need. This is the place to clarify any subtleties you have in your implementation, document other approaches you tried that didn't end up working, any limitations on your implementation, etc. The most important part here is to describe why you made the change you did, not simply what the change you made is.

Changelog:

- Please ensure to update the changelog by adding a new bullet under an Added, Changed, Deprecated, Removed, Fixed, or Security section headers under the Unreleased version. If any of those sections are not present, feel free to add the one you need. See Keep a Changelog if you need guidance on what makes a good entry since this project follows those principles.

Tests:

- Ensure the tests pass: `poetry run task test` to run all tests.
- For any significant code changes, there must be tests to accompany them. All unit tests are written with `pytest`.

Code Format:

- There is a pre-commit pipeline to ensure a standard code format. Make sure to install the pre-commit hooks before making any commits with `pre-commit install`.

CI Pipeline:

- There is a CI pipeline that is run using Github Actions on commits to master, dev, and on pull requests. This pipeline must pass for your changes to be accepted.

4.9.2 Getting Up & Running

This project uses Poetry for the build system and dependency management. To get started, you will want that installed on your system.

Once you've installed `Poetry`, you can install the dependencies, this package, and go into the virtual environment.

```
$ poetry install
$ poetry shell
```

From inside the virtual environment, we can work with the package and easily run the tasks for this project such as `task test` and `task lint` that are in the `pyproject.toml` file.

CHAPTER 5

Indices and tables

- genindex
- search

Python Module Index

I

Index