pySerial-asyncio Documentation

Release 0.6

pySerial-team

Sep 30, 2021

Contents

Async I/O extension for the Python Serial Port package for OSX, Linux, BSD

It depends on pySerial and is compatible with Python 3.5 and later.

Other pages (online)

- project page on GitHub
- Download Page with releases
- This page, when viewed online is at https://pyserial-asyncio.readthedocs.io/en/latest/ or http://pythonhosted.org/pyserial-asyncio/ .

Contents:

CHAPTER 1

Overview

1.1 Serial transports, protocols and streams

This module layers asyncio support onto pySerial. It provides support for working with serial ports through *asyncio* Transports, Protocols, and Streams.

Transports are a low-level abstraction, provided by this package in the form of an `asyncio.Transport` implementation called `SerialTransport`, which manages the asynchronous transmission of data through an underlying *pySerial* `Serial` instance. Transports are concerned with *how* bytes are transmitted through the serial port.

Protocols are a callback-based abstraction which determine *which* bytes are transmitted through an underlying transport. You can implement a subclass of `asyncio.Protocol` which reads from, and/or writes to, a `SerialTransport`. When a serial connection is established your protocol will be handed a transport, to which your protocol implementation can write data as necessary. Incoming data and other serial connection lifecycle events cause callbacks on your protocol to be invoked, so it can take action as necessary.

Usually, you will not create a `SerialTransport` directly. Rather, you will define a `Protocol` class and pass that protocol to a function such as `create_serial_connection()` which will instantiate your `Protocol` and connect it to a `SerialTransport`.

Streams are a coroutine-based alternative to callback-based protocols. This package provides a function `open_serial_connection()` which returns `asyncio.StreamReader` and `asyncio.StreamWriter` objects for interacting with underlying protocol and transport objects, which this library will create for you.

1.2 Protocol Example

This example defines a very simple Protocol which sends a greeting message through the serial port and displays to the console any data received through the serial port, until a newline byte is received.

A call is made to `create_serial_connection()`, to which the protocol *class* (not an instance) is passed, together with arguments destined for the `Serial` constructor. This call returns a coroutine object. When passed to `run_until_complete()` the coroutine is scheduled to run as an `asyncio.Task` by the *asyncio* library, and the result of the coroutine, which is a tuple containing the transport and protocol instances, return to the caller.

While the event loop is running (`run_forever()`), or until the protocol closes the transport itself, the protocol will process data received through the serial port asynchronously:

```
import asyncio
import serial_asyncio

class OutputProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        self.transport = transport
        print('port opened', transport)
        transport.serial.rts = False  # You can manipulate Serial object via transport
        transport.write(b'Hello, World!\n')  # Write serial data via transport

    def data_received(self, data):
        print('data received', repr(data))
        if b'\n' in data:
            self.transport.close()

    def connection_lost(self, exc):
        print('port closed')
        self.transport.loop.stop()

    def pause_writing(self):
        print('pause writing')
        print(self.transport.get_write_buffer_size())

    def resume_writing(self):
        print(self.transport.get_write_buffer_size())
        print('resume writing')

loop = asyncio.get_event_loop()
coro = serial_asyncio.create_serial_connection(loop, OutputProtocol, '/dev/ttyUSB0',␣
↪baudrate=115200)
transport, protocol = loop.run_until_complete(coro)
loop.run_forever()
loop.close()
```

CHAPTER 2

pySerial-asyncio API

The following high-level functions are provided for initiating a serial connection:

CHAPTER 3

Appendix

3.1 License

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

S