pyMultiSerial

Release 1.1.0

unknown

Oct 09, 2022

CONTENTS

A Python module for continuous communication with multiple serial ports, based on pyserial module

Features: - Monitor incoming data from multiple serial ports simultaneously. - Detect connections to port automatically and starts monitoring them. - Detect disconnections from port automatically.

Other pages (online)

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

Contents:

CHAPTER

ONE

PYMULTISERIAL

1.1 Overview

A Python module for continuous communication with multiple serial ports, based on pyserial module.

It is released under a free software license, see https://github.com/SunitRaut/pyMultiSerial/blob/main/LICENSE for more details.

Other pages (online)

- Download Page with releases (PyPi)
- This page, when viewed online is at https://pymultiserial.readthedocs.io/en/latest/ or http://pythonhosted.org/pymultiserial/ .

1.2 Features

- Monitor multiple serial ports simultaneously.
- Detect connections to port automatically and starts monitoring them.
- Raises a trigger whenever data is received from the port. You can attach callback function to process this data on-demand.
- Detect disconnections from port automatically.
- You can add your own processing logic to the above events using callback functions

1.3 Requirements

- Python 3+
- If running on Windows: Windows 7 or newer

1.4 Installation

This installs a package that can be used from Python (`import pyMultiSerial`).

To install for all users on the system, administrator rights (root) may be required.

1.4.1 From PyPI

pySerial can be installed from PyPI:

```
python -m pip install pymultiserial
```

Using the *python*/*python3* executable of the desired version (2/3.x).

Developers also may be interested to get the source archive, because it contains examples, tests and the this documentation.

1.4.2 From Conda

pyMultiSerial can be installed from Conda:

```
conda install pyMultiSerial

or

conda install -c conda-forge pymultiserial
```

Conda: https://www.continuum.io/downloads

1.4.3 From source (zip/tar.gz or checkout)

Download the archive from http://pypi.python.org/pypi/pymultiserial or https://github.com/sunitraut/pymultiserial/releases.

Using the *python*/*python3* executable of the desired version (2/3.x).

CHAPTER

TWO

HOW TO USE

A short guide to get started quickly with this module!

2.1 Install

```
>>> pip install pyMultiSerial
```

2.2 Import

```
>>> import pyMultiSerial
```

2.3 Create Object

```
>>> ms = pyMultiSerial.MultiSerial
```

2.4 Set Properties of Object:

```
>>> ms.baudrate = 9600
>>> ms.timeout = 2
```

2.5 Define Callback Functions

Callback functions are the functions defined by you in your program which are triggered by pyMultiSerial module whenever an event occurs.

The pyMultiSerial module provides 5 callback events for which you can define functions:

1. **New Serial Port Connection Found** - Allows you to write a function which triggers when a new serial port connection is found. With this feature, you can perform many operations like authenticating, performing handshake or simply creating a list of newly connected devices.
2. **Data Received on a Serial Port** - Allows you to write a function which triggers when any data is received on the serial port. With this feature, you can process incoming data from serial ports.

3. **Device disconnected from Serial Port** - Allows you to write a function which triggers when any device is disconnected. With this feature, you can keep track of serial ports that have been disconnected.
4. **On Keyboard Interrupt (Ctrl+C) by user** - Allows you to write a function which triggers when you force stop the python script with keyborard interrupt or through your Python IDE. This feature allows you to perform any clean up activities necessary before exiting your application.
5. **Continuous Loop Execution** - Allows you to write a function which triggers continuously. With this feature, you can perform repitive / periodic tasks. Don't forget to add appropriate delay in this callback since this event occurs continuously. The user gets to decide what frequency / delay is needed.

Note: All the above callbacks are optional. You need to only program those callbacks which you need.

How to define and register callback function:

```
>>> def your_func_name(standard_arguments):
>>>   Your Statements
>>>   End of function
>>> #register callback function
>>> ms.callback_name = your_func_name
```

Here, **your_func_name** can be any name as decided by you. Arguments passed to this function should be in accordance to the **standard_arguments** as defined in below table. These arguments will be passed from pyMultiSerial module to your function. **callback_name** is the property name of the callback event to which you need to assign your function as shown in above snippet.

Table 1: Table

Event	callback_name	standard_arguments for callback function
New Serial Port Connection Found	port_connection_found_callback	Port Number, Serial Port Object
Data Received on a Serial Port	port_read_callback	Port Number, Serial Port Object, Data
Device disconnected from Serial Port	port_disconnection_callback	Port Number
On Keyboard Interrupt (Ctrl+C) by user	interrupt_callback	
Continuous Loop Execution	loop_callback	

In the above table, there are mainly three types of arguments:

1. Port Number - The Port Number of the port on which the event has occured.
2. Serial Port Object - pySerial Object of the Port on which event has occured. This object can be used to read, write to the port from your callback function.
3. Data - Data received from Serial Port in String format.

Note: You must include above paramenters in your function definition. However, it is optional to use those parameters.

2.6 Start Monitoring Ports

```
>>> ms.Start()
```

Caution: Since this module monitors all serial ports simultaneously, ms.Start() is a blocking function. Unless you don't stop monitoring using Stop() method, the execution will be stuck at this line. Start() method should ideally be called at the end of your code. To perform other opertions, you should use the provided callback functions. Caution: Callback functions should be defined before ms.Start() statement. The Callback functions should be registered with the object before monitoring is started, else your callback functions won't be called.

If you need to stop monitoring for any reason, use below statement:

```
>>> ms.Stop()
```

CHAPTER

THREE

MULTISERIAL CLASS

class pyMultiSerial.**MultiSerial**

__init__()

On creating object of class MultiSerial, port parameters are set up. All ports are configured with the same parameters.

Start()

Start the monitoring of ports.

Stop()

Stop the monitoring of ports.

Attributes

baudrate(*Necessary*)

Type
int

Sets baudrate for the serial ports.

timeout(*Optional*)

Type
int

Sets timeout for the serial ports. Default value = 2 sec

portno_range(*Optional*)

Type
int

Sets range of port numbers to monitor starting from 0. Default value = 29

monitoring_freq(*Optional*)

Type
int

Sets monitoring frequency. Default value = 0 msec

port_connection_found_callback(*Optional*)

Type
function

Sets callback function for Event when new serial port connection is found

`port_read_callback`(*Optional*)

Type
function

Sets callback function for Event when data is received on any serial port

`port_disconnection_callback`(*Optional*)

Type
function

Sets callback function for Event when a port is disconnected

`interrupt_callback`(*Optional*)

Type
function

Sets callback function for Event when execution of program is interrupted

`loop_callback`(*Optional*)

Type
function

Sets callback function to be executed in continuous loops.

CHAPTER

FOUR

CALLBACK FUNCTIONS

Callback functions are the functions defined by you in your program which are triggered by pyMultiSerial module whenever an event occurs.

The pyMultiSerial module provides 5 callback events for which you can define functions:

1. **New Serial Port Connection Found** - Allows you to write a function which triggers when a new serial port connection is found. With this feature, you can perform many operations like authenticating, performing handshake or simply creating a list of newly connected devices.
2. **Data Received on a Serial Port** - Allows you to write a function which triggers when any data is received on the serial port. With this feature, you can process incoming data from serial ports.
3. **Device disconnected from Serial Port** - Allows you to write a function which triggers when any device is disconnected. With this feature, you can keep track of serial ports that have been disconnected.
4. **On Keyboard Interrupt (Ctrl+C) by user** - Allows you to write a function which triggers when you force stop the python script with keyborard interrupt or through your Python IDE. This feature allows you to perform any clean up activities necessary before exiting your application.
5. **Continuous Loop Execution** - Allows you to write a function which triggers continuously. With this feature, you can perform repitive / periodic tasks. Don't forget to add appropriate delay in this callback since this event occurs continuously. The user gets to decide what frequency / delay is needed.

Note: All the above callbacks are optional. You need to only program those callbacks which you need.

How to define and register callback function:

```
>>> def your_func_name(standard_arguments):
>>>   Your Statements
>>>   End of function
>>> #register callback function
>>> ms.callback_name = your_func_name
```

Here, **your_func_name** can be any name as decided by you. Arguments passed to this function should be in accordance to the **standard_arguments** as defined in below table. These arguments will be passed from pyMultiSerial module to your function. **callback_name** is the property name of the callback event to which you need to assign your function as shown in above snippet.

Table 1: Table

Event	callback_name	standard_arguments for callback function
New Serial Port Connection Found	port_connection_found_callback	Port Number, Serial Port Object
Data Received on a Serial Port	port_read_callback	Port Number, Serial Port Object, Data
Device disconnected from Serial Port	port_disconnection_callback	Port Number
On Keyboard Interrupt (Ctrl+C) by user	interrupt_callback	
Continuous Loop Execution	loop_callback	

In the above table, there are mainly three types of arguments:

1. Port Number - The Port Number of the port on which the event has occured.
2. Serial Port Object - pySerial Object of the Port on which event has occured. This object can be used to read, write to the port from your callback function.
3. Data - Data received from Serial Port in String format.

Note: You must include above paramenters in your function definition. However, it is optional to use those parameters.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

B

I

L

M

P

S

T