pySerial Documentation

Release 3.3

Chris Liechti

May 24, 2017

Contents

This module encapsulates the access for the serial port. It provides backends for Python running on Windows, OSX, Linux, BSD (possibly any POSIX compliant system) and IronPython. The module named "serial" automatically selects the appropriate backend.

Other pages (online)

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

Contents:

CHAPTER 1

pySerial

Overview

This module encapsulates the access for the serial port. It provides backends for Python running on Windows, OSX, Linux, BSD (possibly any POSIX compliant system) and IronPython. The module named "serial" automatically selects the appropriate backend.

Other pages (online)

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

Features

- Same class based interface on all supported platforms.
- Access to the port settings through Python properties.
- Support for different byte sizes, stop bits, parity and flow control with RTS/CTS and/or Xon/Xoff.
- Working with or without receive timeout.
- File like API with "read" and "write" ("readline" etc. also supported).
- The files in this package are 100% pure Python.
- The port is set up for binary transmission. No NULL byte stripping, CR-LF translation etc. (which are many times enabled for POSIX.) This makes this module universally useful.

- Compatible with `io` library
- RFC 2217 client (experimental), server provided in the examples.

Requirements

- Python 2.7 or Python 3.4 and newer
- If running on Windows: Windows 7 or newer
- If running on Jython: "Java Communications" (JavaComm) or compatible extension for Java

For older installations (older Python versions or older operating systems), see *older versions* below.

Installation

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

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

From PyPI

pySerial can be installed from PyPI:

```
python -m pip install pyserial
```

Using the *python*/*python3* executable of the desired version (2.7/3.x).

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

From source (zip/tar.gz or checkout)

Download the archive from http://pypi.python.org/pypi/pyserial or https://github.com/pyserial/pyserial/releases. Unpack the archive, enter the `pyserial-x.y` directory and run:

```
python setup.py install
```

Using the *python*/*python3* executable of the desired version (2.7/3.x).

Packages

There are also packaged versions for some Linux distributions:

- Debian/Ubuntu: "python-serial", "python3-serial"
- Fedora / RHEL / CentOS / EPEL: "pyserial"
- Arch Linux: "python-pyserial"
- Gentoo: "dev-python/pyserial"

Note that some distributions may package an older version of pySerial. These packages are created and maintained by developers working on these distributions.

References

- Python: http://www.python.org/
- Jython: http://www.jython.org/
- IronPython: http://www.codeplex.com/IronPython

Older Versions

Older versions are still available on the current download page or the old download page. The last version of pySerial's 2.x series was 2.7, compatible with Python 2.3 and newer and partially with early Python 3.x versions.

pySerial 1.21 is compatible with Python 2.0 on Windows, Linux and several un*x like systems, MacOSX and Jython.

On Windows, releases older than 2.5 will depend on pywin32 (previously known as win32all). WinXP is supported up to 3.0.1.

CHAPTER 2

Short introduction

Opening serial ports

Open port at “9600,8,N,1”, no timeout:

```
>>> import serial
>>> ser = serial.Serial('/dev/ttyUSB0')  # open serial port
>>> print(ser.name)          # check which port was really used
>>> ser.write(b'hello')      # write a string
>>> ser.close()              # close port
```

Open named port at “19200,8,N,1”, 1s timeout:

```
>>> with serial.Serial('/dev/ttyS1', 19200, timeout=1) as ser:
...     x = ser.read()          # read one byte
...     s = ser.read(10)        # read up to ten bytes (timeout)
...     line = ser.readline()   # read a '\n' terminated line
```

Open port at “38400,8,E,1”, non blocking HW handshaking:

```
>>> ser = serial.Serial('COM3', 38400, timeout=0,
...                     parity=serial.PARITY_EVEN, rtscts=1)
>>> s = ser.read(100)        # read up to one hundred bytes
...                          # or as much is in the buffer
```

Configuring ports later

Get a Serial instance and configure/open it later:

```
>>> ser = serial.Serial()
>>> ser.baudrate = 19200
>>> ser.port = 'COM1'
```

```
>>> ser
Serial<id=0xa81c10, open=False>(port='COM1', baudrate=19200, bytesize=8, parity='N',␣
↪stopbits=1, timeout=None, xonxoff=0, rtscts=0)
>>> ser.open()
>>> ser.is_open
True
>>> ser.close()
>>> ser.is_open
False
```

Also supported with *context manager*:

```
with serial.Serial() as ser:
    ser.baudrate = 19200
    ser.port = 'COM1'
    ser.open()
    ser.write(b'hello')
```

Readline

Be careful when using `readline()`. Do specify a timeout when opening the serial port otherwise it could block forever if no newline character is received. Also note that `readlines()` only works with a timeout. `readlines()` depends on having a timeout and interprets that as EOF (end of file). It raises an exception if the port is not opened correctly.

Do also have a look at the example files in the examples directory in the source distribution or online.

Note: The `eol` parameter for `readline()` is no longer supported when pySerial is run with newer Python versions (V2.6+) where the module `io` is available.

EOL

To specify the EOL character for `readline()` or to use universal newline mode, it is advised to use io.TextIOWrapper:

```
import serial
import io
ser = serial.serial_for_url('loop://', timeout=1)
sio = io.TextIOWrapper(io.BufferedRWPair(ser, ser))

sio.write(unicode("hello\n"))
sio.flush() # it is buffering. required to get the data out *now*
hello = sio.readline()
print(hello == unicode("hello\n"))
```

Testing ports

Listing ports

`python -m serial.tools.list_ports` will print a list of available ports. It is also possible to add a regexp as first argument and the list will only include entries that matched.

Note: The enumeration may not work on all operating systems. It may be incomplete, list unavailable ports or may lack detailed descriptions of the ports.

Accessing ports

pySerial includes a small console based terminal program called *serial.tools.miniterm*. It ca be started with `python -m serial.tools.miniterm <port_name>` (use option `-h` to get a listing of all options).

CHAPTER 3

pySerial API

Classes

Native ports

class `serial.`**`Serial`**

`__init__`(*port=None*, *baudrate=9600*, *bytesize=EIGHTBITS*, *parity=PARITY_NONE*, *stopbits=STOPBITS_ONE*, *timeout=None*, *xonxoff=False*, *rtscts=False*, *write_timeout=None*, *dsrdtr=False*, *inter_byte_timeout=None*, *exclusive=None*)

Parameters

- **`port`** – Device name or `None`.
- **`baudrate`** (*int*) – Baud rate such as 9600 or 115200 etc.
- **`bytesize`** – Number of data bits. Possible values: *FIVEBITS*, *SIXBITS*, *SEVENBITS*, *EIGHTBITS*
- **`parity`** – Enable parity checking. Possible values: *PARITY_NONE*, *PARITY_EVEN*, *PARITY_ODD PARITY_MARK*, *PARITY_SPACE*
- **`stopbits`** – Number of stop bits. Possible values: *STOPBITS_ONE*, *STOPBITS_ONE_POINT_FIVE*, *STOPBITS_TWO*
- **`timeout`** (*float*) – Set a read timeout value.
- **`xonxoff`** (*bool*) – Enable software flow control.
- **`rtscts`** (*bool*) – Enable hardware (RTS/CTS) flow control.
- **`dsrdtr`** (*bool*) – Enable hardware (DSR/DTR) flow control.
- **`write_timeout`** (*float*) – Set a write timeout value.
- **`inter_byte_timeout`** (*float*) – Inter-character timeout, `None` to disable (default).

- **exclusive** (*bool*) – Set exclusive access mode (POSIX only). A port cannot be opened in exclusive access mode if it is already open in exclusive access mode.

Raises

- **ValueError** – Will be raised when parameter are out of range, e.g. baud rate, data bits.
- ***SerialException*** – In case the device can not be found or can not be configured.

The port is immediately opened on object creation, when a *port* is given. It is not opened when *port* is `None` and a successive call to *open()* is required.

port is a device name: depending on operating system. e.g. `/dev/ttyUSB0` on GNU/Linux or `COM3` on Windows.

The parameter *baudrate* can be one of the standard values: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200. These are well supported on all platforms.

Standard values above 115200, such as: 230400, 460800, 500000, 576000, 921600, 1000000, 1152000, 1500000, 2000000, 2500000, 3000000, 3500000, 4000000 also work on many platforms and devices.

Non-standard values are also supported on some platforms (GNU/Linux, MAC OSX >= Tiger, Windows). Though, even on these platforms some serial ports may reject non-standard values.

Possible values for the parameter *timeout* which controls the behavior of *read()*:

- `timeout = None`: wait forever / until requested number of bytes are received
- `timeout = 0`: non-blocking mode, return immediately in any case, returning zero or more, up to the requested number of bytes
- `timeout = x`: set timeout to `x` seconds (float allowed) returns immediately when the requested number of bytes are available, otherwise wait until the timeout expires and return all bytes that were received until then.

write() is blocking by default, unless *write_timeout* is set. For possible values refer to the list for *timeout* above.

Note that enabling both flow control methods (*xonxoff* and *rtscts*) together may not be supported. It is common to use one of the methods at once, not both.

dsrdtr is not supported by all platforms (silently ignored). Setting it to `None` has the effect that its state follows *rtscts*.

Also consider using the function *serial_for_url()* instead of creating Serial instances directly.

Changed in version 2.5: *dsrdtr* now defaults to `False` (instead of *None*)

Changed in version 3.0: numbers as *port* argument are no longer supported

New in version 3.3: `exclusive` flag

open()

Open port. The state of *rts* and *dtr* is applied.

Note: Some OS and/or drivers may activate RTS and or DTR automatically, as soon as the port is opened. There may be a glitch on RTS/DTR when *rts* or *dtr* are set differently from their default value (`True` / active).

Note: For compatibility reasons, no error is reported when applying *rts* or *dtr* fails on POSIX due to EINVAL (22) or ENOTTY (25).

---

close()
Close port immediately.

__del__()
Destructor, close port when serial port instance is freed.

The following methods may raise *SerialException* when applied to a closed port.

read(*size=1*)

Parameters **size** – Number of bytes to read.

Returns Bytes read from the port.

Return type bytes

Read *size* bytes from the serial port. If a timeout is set it may return less characters as requested. With no timeout it will block until the requested number of bytes is read.

Changed in version 2.5: Returns an instance of `bytes` when available (Python 2.6 and newer) and `str` otherwise.

write(*data*)

Parameters **data** – Data to send.

Returns Number of bytes written.

Return type int

Raises ***SerialTimeoutException*** – In case a write timeout is configured for the port and the time is exceeded.

Write the bytes *data* to the port. This should be of type `bytes` (or compatible such as `bytearray` or `memoryview`). Unicode strings must be encoded (e.g. `'hello'.encode('utf-8')`).

Changed in version 2.5: Accepts instances of `bytes` and `bytearray` when available (Python 2.6 and newer) and `str` otherwise.

Changed in version 2.5: Write returned `None` in previous versions.

flush()
Flush of file like objects. In this case, wait until all data is written.

in_waiting

Getter Get the number of bytes in the input buffer

Type int

Return the number of bytes in the receive buffer.

Changed in version 3.0: changed to property from `inWaiting()`

out_waiting

Getter Get the number of bytes in the output buffer

Type int

Platform Posix

Platform Windows

Return the number of bytes in the output buffer.

Changed in version 2.7: (Posix support added)

Changed in version 3.0: changed to property from `outWaiting()`

reset_input_buffer()
Flush input buffer, discarding all its contents.

Changed in version 3.0: renamed from `flushInput()`

reset_output_buffer()
Clear output buffer, aborting the current output and discarding all that is in the buffer.

Note, for some USB serial adapters, this may only flush the buffer of the OS and not all the data that may be present in the USB part.

Changed in version 3.0: renamed from `flushOutput()`

send_break(*duration=0.25*)

Parameters **duration** (*float*) – Time to activate the BREAK condition.

Send break condition. Timed, returns to idle state after given duration.

break_condition

Getter Get the current BREAK state

Setter Control the BREAK state

Type bool

When set to `True` activate BREAK condition, else disable. Controls TXD. When active, no transmitting is possible.

rts

Setter Set the state of the RTS line

Getter Return the state of the RTS line

Type bool

Set RTS line to specified logic level. It is possible to assign this value before opening the serial port, then the value is applied uppon *open()* (with restrictions, see *open()*).

dtr

Setter Set the state of the DTR line

Getter Return the state of the DTR line

Type bool

Set DTR line to specified logic level. It is possible to assign this value before opening the serial port, then the value is applied uppon *open()* (with restrictions, see *open()*).

Read-only attributes:

name

Getter Device name.

Type str

New in version 2.5.

cts

Getter Get the state of the CTS line

Type bool

Return the state of the CTS line.

dsr

Getter Get the state of the DSR line

Type bool

Return the state of the DSR line.

ri

Getter Get the state of the RI line

Type bool

Return the state of the RI line.

cd

Getter Get the state of the CD line

Type bool

Return the state of the CD line

is_open

Getter Get the state of the serial port, whether it's open.

Type bool

New values can be assigned to the following attributes (properties), the port will be reconfigured, even if it's opened at that time:

port

Type str

Read or write port. When the port is already open, it will be closed and reopened with the new setting.

baudrate

Getter Get current baud rate

Setter Set new baud rate

Type int

Read or write current baud rate setting.

bytesize

Getter Get current byte size

Setter Set new byte size. Possible values: *FIVEBITS*, *SIXBITS*, *SEVENBITS*, *EIGHTBITS*

Type int

Read or write current data byte size setting.

parity

Getter Get current parity setting

Setter Set new parity mode. Possible values: *PARITY_NONE*, *PARITY_EVEN*, *PARITY_ODD* *PARITY_MARK*, *PARITY_SPACE*

Read or write current parity setting.

stopbits

Getter Get current stop bit setting

Setter Set new stop bit setting. Possible values: *STOPBITS_ONE*, *STOPBITS_ONE_POINT_FIVE*, *STOPBITS_TWO*

Read or write current stop bit width setting.

timeout

Getter Get current read timeout setting

Setter Set read timeout

Type float (seconds)

Read or write current read timeout setting.

write_timeout

Getter Get current write timeout setting

Setter Set write timeout

Type float (seconds)

Read or write current write timeout setting.

Changed in version 3.0: renamed from `writeTimeout`

inter_byte_timeout

Getter Get current inter byte timeout setting

Setter Disable (`None`) or enable the inter byte timeout

Type float or None

Read or write current inter byte timeout setting.

Changed in version 3.0: renamed from `interCharTimeout`

xonxoff

Getter Get current software flow control setting

Setter Enable or disable software flow control

Type bool

Read or write current software flow control rate setting.

rtscts

Getter Get current hardware flow control setting

Setter Enable or disable hardware flow control

Type bool

Read or write current hardware flow control setting.

dsrdtr

Getter Get current hardware flow control setting

Setter Enable or disable hardware flow control

Type bool

Read or write current hardware flow control setting.

rs485_mode

Getter Get the current RS485 settings

Setter Disable (`None`) or enable the RS485 settings

Type *rs485.RS485Settings* or `None`

Platform Posix (Linux, limited set of hardware)

Platform Windows (only RTS on TX possible)

Attribute to configure RS485 support. When set to an instance of *rs485.RS485Settings* and supported by OS, RTS will be active when data is sent and inactive otherwise (for reception). The *rs485.RS485Settings* class provides additional settings supported on some platforms.

New in version 3.0.

The following constants are also provided:

BAUDRATES
A list of valid baud rates. The list may be incomplete, such that higher and/or intermediate baud rates may also be supported by the device (Read Only).

BYTESIZES
A list of valid byte sizes for the device (Read Only).

PARITIES
A list of valid parities for the device (Read Only).

STOPBITS
A list of valid stop bit widths for the device (Read Only).

The following methods are for compatibility with the `io` library.

readable()

Returns True

New in version 2.5.

writable()

Returns True

New in version 2.5.

seekable()

Returns False

New in version 2.5.

readinto(*b*)

Parameters **b** – bytearray or array instance

Returns Number of byte read

Read up to len(b) bytes into `bytearray` *b* and return the number of bytes read.

New in version 2.5.

The port settings can be read and written as dictionary. The following keys are supported: `write_timeout`, `inter_byte_timeout`, `dsrdtr`, `baudrate`, `timeout`, `parity`, `bytesize`, `rtscts`, `stopbits`, `xonxoff`

get_settings()

Returns a dictionary with current port settings.

Return type dict

Get a dictionary with port settings. This is useful to backup the current settings so that a later point in time they can be restored using *apply_settings()*.

Note that the state of control lines (RTS/DTR) are not part of the settings.

New in version 2.5.

Changed in version 3.0: renamed from `getSettingsDict`

apply_settings(*d*)

Parameters **d** (*dict*) – a dictionary with port settings.

Applies a dictionary that was created by *get_settings()*. Only changes are applied and when a key is missing, it means that the setting stays unchanged.

Note that control lines (RTS/DTR) are not changed.

New in version 2.5.

Changed in version 3.0: renamed from `applySettingsDict`

This class can be used as context manager. The serial port is closed when the context is left.

__enter__()

Returns Serial instance

Returns the instance that was used in the `with` statement.

Example:

```
>>> with serial.serial_for_url(port) as s:
...     s.write(b'hello')
```

Here no port argument is given, so it is not opened automatically:

```
>>> with serial.Serial() as s:
...     s.port = ...
...     s.open()
...     s.write(b'hello')
```

__exit__(*exc_type*, *exc_val*, *exc_tb*)
Closes serial port (exceptions are not handled by `__exit__`).

Platform specific methods.

Warning: Programs using the following methods and attributes are not portable to other platforms!

nonblocking()

Platform Posix

Deprecated since version 3.2: The serial port is already opened in this mode. This method is not needed and going away.

fileno()

Platform Posix

Returns File descriptor.

Return file descriptor number for the port that is opened by this object. It is useful when serial ports are used with `select`.

set_input_flow_control(*enable*)

Platform Posix

Parameters **enable** (*bool*) – Set flow control state.

Manually control flow - when software flow control is enabled.

This will send XON (true) and XOFF (false) to the other device.

New in version 2.7: (Posix support added)

Changed in version 3.0: renamed from `flowControlOut`

set_output_flow_control(*enable*)

Platform Posix (HW and SW flow control)

Platform Windows (SW flow control only)

Parameters **enable** (*bool*) – Set flow control state.

Manually control flow of outgoing data - when hardware or software flow control is enabled.

Sending will be suspended when called with `False` and enabled when called with `True`.

Changed in version 2.7: (renamed on Posix, function was called `flowControl`)

Changed in version 3.0: renamed from `setXON`

cancel_read()

Platform Posix

Platform Windows

Cancel a pending read operation from another thread. A blocking *read()* call is aborted immediately. *read()* will not report any error but return all data received up to that point (similar to a timeout).

On Posix a call to *cancel_read()* may cancel a future *read()* call.

New in version 3.1.

cancel_write()

Platform Posix

Platform Windows

Cancel a pending write operation from another thread. The *write()* method will return immediately (no error indicated). However the OS may still be sending from the buffer, a separate call to *reset_output_buffer()* may be needed.

On Posix a call to *cancel_write()* may cancel a future *write()* call.

New in version 3.1.

Note: The following members are deprecated and will be removed in a future release.

---

portstr
Deprecated since version 2.5: use *name* instead

inWaiting()
Deprecated since version 3.0: see *in_waiting*

isOpen()
Deprecated since version 3.0: see *is_open*

writeTimeout
Deprecated since version 3.0: see *write_timeout*

interCharTimeout
Deprecated since version 3.0: see *inter_byte_timeout*

sendBreak(*duration=0.25*)
Deprecated since version 3.0: see *send_break()*

flushInput()
Deprecated since version 3.0: see *reset_input_buffer()*

flushOutput()
Deprecated since version 3.0: see *reset_output_buffer()*

setBreak(*level=True*)
Deprecated since version 3.0: see *break_condition*

setRTS(*level=True*)
Deprecated since version 3.0: see *rts*

setDTR(*level=True*)
Deprecated since version 3.0: see *dtr*

getCTS()
Deprecated since version 3.0: see *cts*

getDSR()
Deprecated since version 3.0: see *dsr*

getRI()
Deprecated since version 3.0: see *ri*

getCD()
Deprecated since version 3.0: see *cd*

getSettingsDict()
Deprecated since version 3.0: see *get_settings()*

applySettingsDict(*d*)
Deprecated since version 3.0: see *apply_settings()*

outWaiting()
Deprecated since version 3.0: see *out_waiting*

setXON(*level=True*)
Deprecated since version 3.0: see *set_output_flow_control()*

flowControlOut(*enable*)
Deprecated since version 3.0: see *set_input_flow_control()*

rtsToggle

Platform Windows

Attribute to configure RTS toggle control setting. When enabled and supported by OS, RTS will be active when data is available and inactive if no data is available.

New in version 2.6.

Changed in version 3.0: (removed, see *rs485_mode* instead)

Implementation detail: some attributes and functions are provided by the class `SerialBase` and some by the platform specific class and others by the base class mentioned above.

RS485 support

The *Serial* class has a *Serial.rs485_mode* attribute which allows to enable RS485 specific support on some platforms. Currently Windows and Linux (only a small number of devices) are supported.

Serial.rs485_mode needs to be set to an instance of *rs485.RS485Settings* to enable or to `None` to disable this feature.

Usage:

```
import serial
import serial.rs485
ser = serial.Serial(...)
ser.rs485_mode = serial.rs485.RS485Settings(...)
ser.write(b'hello')
```

There is a subclass *rs485.RS485* available to emulate the RS485 support on regular serial ports (`serial.rs485` needs to be imported).

class `rs485.`**RS485Settings**
A class that holds RS485 specific settings which are supported on some platforms.

New in version 3.0.

__init__(rts_level_for_tx=True, rts_level_for_rx=False, loopback=False, delay_before_t

Parameters

- **rts_level_for_tx** (*bool*) – RTS level for transmission
- **rts_level_for_rx** (*bool*) – RTS level for reception
- **loopback** (*bool*) – When set to `True` transmitted data is also received.
- **delay_before_tx** (*float*) – Delay after setting RTS but before transmission starts
- **delay_before_rx** (*float*) – Delay after transmission ends and resetting RTS

rts_level_for_tx
RTS level for transmission.

rts_level_for_rx
RTS level for reception.

loopback
When set to `True` transmitted data is also received.

delay_before_tx
Delay after setting RTS but before transmission starts (seconds as float).

delay_before_rx
Delay after transmission ends and resetting RTS (seconds as float).

class `rs485.`**RS485**
A subclass that replaces the *Serial.write()* method with one that toggles RTS according to the RS485 settings.

Usage:

```
ser = serial.rs485.RS485(...)
ser.rs485_mode = serial.rs485.RS485Settings(...)
ser.write(b'hello')
```

Warning: This may work unreliably on some serial ports (control signals not synchronized or delayed compared to data). Using delays may be unreliable (varying times, larger than expected) as the OS may not support very fine grained delays (no smaller than in the order of tens of milliseconds).

Note: Some implementations support this natively in the class *Serial*. Better performance can be expected when the native version is used.

Note: The loopback property is ignored by this implementation. The actual behavior depends on the used hardware.

RFC 2217 Network ports

Warning: This implementation is currently in an experimental state. Use at your own risk.

class rfc2217.**Serial**

This implements a **RFC 2217** compatible client. Port names are *URL* in the form: `rfc2217://<host>:<port>[?<option>[&<option>]]`

This class API is compatible to *Serial* with a few exceptions:

- `write_timeout` is not implemented
- The current implementation starts a thread that keeps reading from the (internal) socket. The thread is managed automatically by the *rfc2217.Serial* port object on `open()`/`close()`. However it may be a problem for user applications that like to use select instead of threads.

Due to the nature of the network and protocol involved there are a few extra points to keep in mind:

- All operations have an additional latency time.
- Setting control lines (RTS/CTS) needs more time.
- Reading the status lines (DSR/DTR etc.) returns a cached value. When that cache is updated depends entirely on the server. The server itself may implement a polling at a certain rate and quick changes may be invisible.
- The network layer also has buffers. This means that `flush()`, `reset_input_buffer()` and `reset_output_buffer()` may work with additional delay. Likewise `in_waiting` returns the size of the data arrived at the objects internal buffer and excludes any bytes in the network buffers or any server side buffer.
- Closing and immediately reopening the same port may fail due to time needed by the server to get ready again.

Not implemented yet / Possible problems with the implementation:

- RFC 2217 flow control between client and server (objects internal buffer may eat all your memory when never read).
- No authentication support (servers may not prompt for a password etc.)
- No encryption.

Due to lack of authentication and encryption it is not suitable to use this client for connections across the internet and should only be used in controlled environments.

New in version 2.5.

class `rfc2217.`**`PortManager`**

This class provides helper functions for implementing RFC 2217 compatible servers.

Basically, it implements everything needed for the RFC 2217 protocol. It just does not open sockets and read/write to serial ports (though it changes other port settings). The user of this class must take care of the data transmission itself. The reason for that is, that this way, this class supports all programming models such as threads and select.

Usage examples can be found in the examples where two TCP/IP - serial converters are shown, one using threads (the single port server) and an other using select (the multi port server).

Note: Each new client connection must create a new instance as this object (and the RFC 2217 protocol) has internal state.

`__init__`(*serial_port*, *connection*, *debug_output=False*)

Parameters

- **`serial_port`** – a *`Serial`* instance that is managed.
- **`connection`** – an object implementing `write()`, used to write to the network.
- **`debug_output`** – enables debug messages: a `logging.Logger` instance or None.

Initializes the Manager and starts negotiating with client in Telnet and RFC 2217 protocol. The negotiation starts immediately so that the class should be instantiated in the moment the client connects.

The *serial_port* can be controlled by RFC 2217 commands. This object will modify the port settings (baud rate etc.) and control lines (RTS/DTR) send BREAK etc. when the corresponding commands are found by the *`filter()`* method.

The *connection* object must implement a `write()` function. This function must ensure that *data* is written at once (no user data mixed in, i.e. it must be thread-safe). All data must be sent in its raw form (*`escape()`* must not be used) as it is used to send Telnet and RFC 2217 control commands.

For diagnostics of the connection or the implementation, *debug_output* can be set to an instance of a `logging.Logger` (e.g. `logging.getLogger('rfc2217.server')`). The caller should configure the logger using `setLevel` for the desired detail level of the logs.

`escape`(*data*)

Parameters **`data`** – data to be sent over the network.

Returns data, escaped for Telnet/RFC 2217

A generator that escapes all data to be compatible with RFC 2217. Implementors of servers should use this function to process all data sent over the network.

The function returns a generator which can be used in `for` loops. It can be converted to bytes using *`serial.to_bytes()`*.

filter(*data*)

Parameters **data** – data read from the network, including Telnet and RFC 2217 controls.

Returns data, free from Telnet and RFC 2217 controls.

A generator that filters and processes all data related to RFC 2217. Implementors of servers should use this function to process all data received from the network.

The function returns a generator which can be used in `for` loops. It can be converted to bytes using *`serial.to_bytes()`*.

check_modem_lines(*force_notification=False*)

Parameters **force_notification** – Set to false. Parameter is for internal use.

This function needs to be called periodically (e.g. every second) when the server wants to send NOTIFY_MODEMSTATE messages. This is required to support the client for reading CTS/DSR/RI/CD status lines.

The function reads the status line and issues the notifications automatically.

New in version 2.5.

See also:

RFC 2217 - Telnet Com Port Control Option

Exceptions

exception `serial.`**SerialException**
Base class for serial port exceptions.

Changed in version 2.5: Now derives from `IOError` instead of `Exception`

exception `serial.`**SerialTimeoutException**
Exception that is raised on write timeouts.

Constants

Parity

`serial.`**PARITY_NONE**

`serial.`**PARITY_EVEN**

`serial.`**PARITY_ODD**

`serial.`**PARITY_MARK**

`serial.`**PARITY_SPACE**

Stop bits

`serial.`**STOPBITS_ONE**

`serial.`**STOPBITS_ONE_POINT_FIVE**

`serial.`**STOPBITS_TWO**

Note that 1.5 stop bits are not supported on POSIX. It will fall back to 2 stop bits.

Byte size

serial.**FIVEBITS**

serial.**SIXBITS**

serial.**SEVENBITS**

serial.**EIGHTBITS**

Others

Default control characters (instances of `bytes` for Python 3.0+) for software flow control:

serial.**XON**

serial.**XOFF**

Module version:

serial.**VERSION**
A string indicating the pySerial version, such as `3.0`.

New in version 2.3.

Module functions and attributes

serial.**device**(*number*)
Changed in version 3.0: removed, use `serial.tools.list_ports` instead

serial.**serial_for_url**(*url*, **args*, ***kwargs*)

Parameters

- **url** – Device name, number or *URL*
- **do_not_open** – When set to true, the serial port is not opened.

Returns an instance of *Serial* or a compatible object.

Get a native or a **RFC 2217** implementation of the Serial class, depending on port/url. This factory function is useful when an application wants to support both, local ports and remote ports. There is also support for other types, see *URL* section.

The port is not opened when a keyword parameter called *do_not_open* is given and true, by default it is opened.

New in version 2.5.

serial.**protocol_handler_packages**
This attribute is a list of package names (strings) that is searched for protocol handlers.

e.g. we want to support a URL `foobar://`. A module `my_handlers.protocol_foobar` is provided by the user:

```
serial.protocol_handler_packages.append("my_handlers")
s = serial.serial_for_url("foobar://")
```

For an URL starting with `XY://` is the function *serial_for_url()* attempts to import `PACKAGE.protocol_XY` with each candidate for `PACKAGE` from this list.

New in version 2.6.

serial.**to_bytes**(*sequence*)

Parameters **sequence** – bytes, bytearray or memoryview

Returns an instance of `bytes`

Convert a sequence to a `bytes` type. This is used to write code that is compatible to Python 2.x and 3.x.

In Python versions prior 3.x, `bytes` is a subclass of str. They convert `str([17])` to `'[17]'` instead of `'\x11'` so a simple `bytes(sequence)` doesn't work for all versions of Python.

This function is used internally and in the unit tests.

New in version 2.5.

`serial.`**`iterbytes`**(*sequence*)

Parameters **`sequence`** – bytes, bytearray or memoryview

Returns a generator that yields bytes

Some versions of Python (3.x) would return integers instead of bytes when looping over an instance of `bytes`. This helper function ensures that bytes are returned.

New in version 3.0.

Threading

New in version 3.0.

Warning: This implementation is currently in an experimental state. Use at your own risk.

This module provides classes to simplify working with threads and protocols.

class `serial.threaded.`**`Protocol`**
Protocol as used by the *`ReaderThread`*. This base class provides empty implementations of all methods.

`connection_made`(*transport*)

Parameters **`transport`** – instance used to write to serial port.

Called when reader thread is started.

`data_received`(*data*)

Parameters **`data`** (*`bytes`*) – received bytes

Called with snippets received from the serial port.

`connection_lost`(*exc*)

Parameters **`exc`** – Exception if connection was terminated by error else `None`

Called when the serial port is closed or the reader loop terminated otherwise.

class `serial.threaded.`**`Packetizer`**(*Protocol*)
Read binary packets from serial port. Packets are expected to be terminated with a `TERMINATOR` byte (null byte by default).

The class also keeps track of the transport.

`TERMINATOR = b'\0'`

`__init__`()

`connection_made`(*transport*)
Stores transport.

connection_lost(*exc*)
Forgets transport.

data_received(*data*)

Parameters **data** (*bytes*) – partial received data

Buffer received data and search for TERMINATOR, when found, call *handle_packet()*.

handle_packet(*packet*)

Parameters **packet** (*bytes*) – a packet as defined by TERMINATOR

Process packets - to be overridden by subclassing.

class serial.threaded.**LineReader**(*Packetizer*)
Read and write (Unicode) lines from/to serial port. The encoding is applied.

TERMINATOR = b'\r\n'
Line ending.

ENCODING = 'utf-8'
Encoding of the send and received data.

UNICODE_HANDLING = 'replace'
Unicode error handly policy.

handle_packet(*packet*)

Parameters **packet** (*bytes*) – a packet as defined by TERMINATOR

In this case it will be a line, calls *handle_line()* after applying the ENCODING.

handle_line(*line*)

Parameters **line** (*str*) – Unicode string with one line (excluding line terminator)

Process one line - to be overridden by subclassing.

write_line(*text*)

Parameters **text** (*str*) – Unicode string with one line (excluding line terminator)

Write *text* to the transport. *text* is expected to be a Unicode string and the encoding is applied before sending and also the TERMINATOR (new line) is appended.

class serial.threaded.**ReaderThread**(*threading.Thread*)
Implement a serial port read loop and dispatch to a Protocol instance (like the asyncio.Protocol) but do it with threads.

Calls to *close()* will close the serial port but it is also possible to just *stop()* this thread and continue to use the serial port instance otherwise.

__init__(*serial_instance*, *protocol_factory*)

Parameters

- **serial_instance** – serial port instance (opened) to be used.
- **protocol_factory** – a callable that returns a Protocol instance

Initialize thread.

Note that the serial_instance 's timeout is set to one second! Other settings are not changed.

stop()
Stop the reader thread.

run()
The actual reader loop driven by the thread. It calls *Protocol.connection_made()*, reads from the serial port calling *Protocol.data_received()* and finally calls *Protocol.connection_lost()* when *close()* is called or an error occurs.

write(*data*)

Parameters **data** (*bytes*) – data to write

Thread safe writing (uses lock).

close()
Close the serial port and exit reader thread, calls *stop()* (uses lock).

connect()
Wait until connection is set up and return the transport and protocol instances.

This class can be used as context manager, in this case it starts the thread and connects automatically. The serial port is closed when the context is left.

__enter__()

Returns protocol

Connect and return protocol instance.

__exit__(*exc_type*, *exc_val*, *exc_tb*)
Closes serial port.

Example:

```
class PrintLines(LineReader):
    def connection_made(self, transport):
        super(PrintLines, self).connection_made(transport)
        sys.stdout.write('port opened\n')
        self.write_line('hello world')

    def handle_line(self, data):
        sys.stdout.write('line received: {}\n'.format(repr(data)))

    def connection_lost(self, exc):
        if exc:
            traceback.print_exc(exc)
        sys.stdout.write('port closed\n')

ser = serial.serial_for_url('loop://', baudrate=115200, timeout=1)
with ReaderThread(ser, PrintLines) as protocol:
    protocol.write_line('hello')
    time.sleep(2)
```

asyncio

`asyncio` was introduced with Python 3.4. Experimental support for pySerial is provided via a separate distribution pyserial-asyncio.

It is currently under development, see:

- http://pyserial-asyncio.readthedocs.io/
- https://github.com/pyserial/pyserial-asyncio

CHAPTER 4

Tools

serial.tools.list_ports

This module can be executed to get a list of ports (`python -m serial.tools.list_ports`). It also contains the following functions.

`serial.tools.list_ports.`**`comports`**(*include_links=False*)

Parameters **`include_links`** (*bool*) – include symlinks under `/dev` when they point to a serial port

Returns a list containing *`ListPortInfo`* objects.

The function returns a list of *`ListPortInfo`* objects.

Items are returned in no particular order. It may make sense to sort the items. Also note that the reported strings are different across platforms and operating systems, even for the same device.

Note: Support is limited to a number of operating systems. On some systems description and hardware ID will not be available (`None`).

Under Linux, OSX and Windows, extended information will be available for USB devices (e.g. the *`ListPortInfo.hwid`* string contains *VID:PID*, *SER* (serial number), *LOCATION* (hierarchy), which makes them searchable via *`grep()`*. The USB info is also available as attributes of *`ListPortInfo`*.

If *include_links* is true, all devices under `/dev` are inspected and tested if they are a link to a known serial port device. These entries will include `LINK` in their `hwid` string. This implies that the same device listed twice, once under its original name and once under linked name.

Platform Posix (/dev files)

Platform Linux (/dev files, sysfs)

Platform OSX (iokit)

Platform Windows (setupapi, registry)

serial.tools.list_ports.**grep**(*regexp*, *include_links=False*)

Parameters

- **regexp** – regular expression (see stdlib re)
- **include_links** (*bool*) – include symlinks under /dev when they point to a serial port

Returns an iterable that yields *ListPortInfo* objects, see also *comports()*.

Search for ports using a regular expression. Port name, description and hwid are searched (case insensitive). The function returns an iterable that contains the same data that *comports()* generates, but includes only those entries that match the regexp.

class serial.tools.list_ports.**ListPortInfo**

This object holds information about a serial port. It supports indexed access for backwards compatibility, as in port, desc, hwid = info.

device
Full device name/path, e.g. /dev/ttyUSB0. This is also the information returned as first element when accessed by index.

name
Short device name, e.g. ttyUSB0.

description
Human readable description or n/a. This is also the information returned as second element when accessed by index.

hwid
Technical description or n/a. This is also the information returned as third element when accessed by index.

USB specific data, these are all None if it is not an USB device (or the platform does not support extended info).

vid
USB Vendor ID (integer, 0...65535).

pid
USB product ID (integer, 0...65535).

serial_number
USB serial number as a string.

location
USB device location string ("<bus>-<port>[-<port>]...")

manufacturer
USB manufacturer string, as reported by device.

product
USB product string, as reported by device.

interface
Interface specific description, e.g. used in compound USB devices.

Comparison operators are implemented such that the *ListPortInfo* objects can be sorted by device. Strings are split into groups of numbers and text so that the order is "natural" (i.e. com1 < com2 < com10).

Command line usage

Help for python -m serial.tools.list_ports:

```
usage: list_ports.py [-h] [-v] [-q] [-n N] [-s] [regexp]

Serial port enumeration

positional arguments:
  regexp                only show ports that match this regex

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         show more messages
  -q, --quiet           suppress all messages
  -n N                  only output the N-th entry
  -s, --include-links   include entries that are symlinks to real devices
```

Examples:

- List all ports with details:

```
$ python -m serial.tools.list_ports -v
/dev/ttyS0
    desc: ttyS0
    hwid: PNP0501
/dev/ttyUSB0
    desc: CP2102 USB to UART Bridge Controller
    hwid: USB VID:PID=10C4:EA60 SER=0001 LOCATION=2-1.6
2 ports found
```

- List the 2nd port matching a USB VID:PID pattern:

```
$ python -m serial.tools.list_ports 1234:5678 -q -n 2
/dev/ttyUSB1
```

New in version 2.6.

Changed in version 3.0: returning `ListPortInfo` objects instead of a tuple

serial.tools.miniterm

This is a console application that provides a small terminal application.

Miniterm itself does not implement any terminal features such as VT102 compatibility. However it may inherit these features from the terminal it is run. For example on GNU/Linux running from an xterm it will support the escape sequences of the xterm. On Windows the typical console window is dumb and does not support any escapes. When ANSI.sys is loaded it supports some escapes.

The default is to filter terminal control characters, see `--filter` for different options.

Miniterm:

```
--- Miniterm on /dev/ttyS0: 9600,8,N,1 ---
--- Quit: Ctrl+]  |  Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
```

Command line options can be given so that binary data including escapes for terminals are escaped or output as hex.

Miniterm supports **RFC 2217** remote serial ports and raw sockets using *URL Handlers* such as `rfc2217://<host>:<port>` respectively `socket://<host>:<port>` as *port* argument when invoking.

Command line options `python -m serial.tools.miniterm -h`:

```
usage: miniterm.py [-h] [--parity {N,E,O,S,M}] [--rtscts] [--xonxoff]
                   [--rts RTS] [--dtr DTR] [-e] [--encoding CODEC] [-f NAME]
                   [--eol {CR,LF,CRLF}] [--raw] [--exit-char NUM]
                   [--menu-char NUM] [-q] [--develop]
                   [port] [baudrate]

Miniterm - A simple terminal program for the serial port.

positional arguments:
  port                  serial port name
  baudrate              set baud rate, default: 9600

optional arguments:
  -h, --help            show this help message and exit

port settings:
  --parity {N,E,O,S,M}  set parity, one of {N E O S M}, default: N
  --rtscts              enable RTS/CTS flow control (default off)
  --xonxoff             enable software flow control (default off)
  --rts RTS             set initial RTS line state (possible values: 0, 1)
  --dtr DTR             set initial DTR line state (possible values: 0, 1)
  --ask                 ask again for port when open fails

data handling:
  -e, --echo            enable local echo (default off)
  --encoding CODEC      set the encoding for the serial port (e.g. hexlify,
                        Latin1, UTF-8), default: UTF-8
  -f NAME, --filter NAME
                        add text transformation
  --eol {CR,LF,CRLF}    end of line mode
  --raw                 Do no apply any encodings/transformations

hotkeys:
  --exit-char NUM       Unicode of special character that is used to exit the
                        application, default: 29
  --menu-char NUM       Unicode code of special character that is used to
                        control miniterm (menu), default: 20

diagnostics:
  -q, --quiet           suppress non-error messages
  --develop             show Python traceback on error
```

Available filters (`--filter` option):

- `colorize`: Apply different colors for received and echo
- `debug`: Print what is sent and received
- `default`: remove typical terminal control codes from input
- `direct`: do-nothing: forward all data unchanged
- `nocontrol`: Remove all control codes, incl. `CR+LF`
- `printable`: Show decimal code for all non-ASCII characters and replace most control codes

Miniterm supports some control functions while being connected. Typing `Ctrl+T Ctrl+H` when it is running shows the help text:

```
--- pySerial (3.0a) - miniterm - help
---
--- Ctrl+]   Exit program
--- Ctrl+T   Menu escape key, followed by:
--- Menu keys:
---    Ctrl+T  Send the menu character itself to remote
---    Ctrl+]  Send the exit character itself to remote
---    Ctrl+I  Show info
---    Ctrl+U  Upload file (prompt will be shown)
---    Ctrl+A  encoding
---    Ctrl+F  edit filters
--- Toggles:
---    Ctrl+R  RTS   Ctrl+D  DTR   Ctrl+B  BREAK
---    Ctrl+E  echo  Ctrl+L  EOL
---
--- Port settings (Ctrl+T followed by the following):
---    p          change port
---    7 8        set data bits
---    N E O S M  change parity (None, Even, Odd, Space, Mark)
---    1 2 3      set stop bits (1, 2, 1.5)
---    b          change baud rate
---    x X        disable/enable software flow control
---    r R        disable/enable hardware flow control
```

Changed in version 2.5: Added `Ctrl+T` menu and added support for opening URLs.

Changed in version 2.6: File moved from the examples to *`serial.tools.miniterm`*.

Changed in version 3.0: Apply encoding on serial port, convert to Unicode for console. Added new filters, default to stripping terminal control sequences. Added `--ask` option.

CHAPTER 5

URL Handlers

Overview

The function *serial_for_url()* accepts the following types of URLs:

- `rfc2217://<host>:<port>[?<option>[&<option>...]]`
- `socket://<host>:<port>[?logging={debug|info|warning|error}]`
- `loop://[?logging={debug|info|warning|error}]`
- `hwgrep://<regexp>[&skip_busy][&n=N]`
- `spy://port[?option[=value][&option[=value]]]`
- `alt://port?class=<classname>`

Changed in version 3.0: Options are specified with `?` and `&` instead of `/`

Device names are also supported, e.g.:

- `/dev/ttyUSB0` (Linux)
- `COM3` (Windows)

Future releases of pySerial might add more types. Since pySerial 2.6 it is also possible for the user to add protocol handlers using *protocol_handler_packages*.

rfc2217://

Used to connect to **RFC 2217** compatible servers. All serial port functions are supported. Implemented by *rfc2217.Serial*.

Supported options in the URL are:

- `ign_set_control` does not wait for acknowledges to SET_CONTROL. This option can be used for non compliant servers (i.e. when getting an `remote rejected value for option 'control'` error when connecting).
- `poll_modem`: The client issues NOTIFY_MODEMSTATE requests when status lines are read (CTS/DTR/RI/CD). Without this option it relies on the server sending the notifications automatically (that's what the RFC suggests and most servers do). Enable this option when `cts` does not work as expected, i.e. for servers that do not send notifications.
- `timeout=<value>`: Change network timeout (default 3 seconds). This is useful when the server takes a little more time to send its answers. The timeout applies to the initial Telnet / **RFC 2271** negotiation as well as changing port settings or control line change commands.
- `logging={debug|info|warning|error}`: Prints diagnostic messages (not useful for end users). It uses the logging module and a logger called `pySerial.rfc2217` so that the application can setup up logging handlers etc. It will call `logging.basicConfig()` which initializes for output on `sys.stderr` (if no logging was set up already).

Warning: The connection is not encrypted and no authentication is supported! Only use it in trusted environments.

socket://

The purpose of this connection type is that applications using pySerial can connect to TCP/IP to serial port converters that do not support **RFC 2217**.

Uses a TCP/IP socket. All serial port settings, control and status lines are ignored. Only data is transmitted and received.

Supported options in the URL are:

- `logging={debug|info|warning|error}`: Prints diagnostic messages (not useful for end users). It uses the logging module and a logger called `pySerial.socket` so that the application can setup up logging handlers etc. It will call `logging.basicConfig()` which initializes for output on `sys.stderr` (if no logging was set up already).

Warning: The connection is not encrypted and no authentication is supported! Only use it in trusted environments.

loop://

The least useful type. It simulates a loop back connection (`RX<->TX RTS<->CTS DTR<->DSR`). It could be used to test applications or run the unit tests.

Supported options in the URL are:

- `logging={debug|info|warning|error}`: Prints diagnostic messages (not useful for end users). It uses the logging module and a logger called `pySerial.loop` so that the application can setup up logging handlers etc. It will call `logging.basicConfig()` which initializes for output on `sys.stderr` (if no logging was set up already).

hwgrep://

This type uses *serial.tools.list_ports* to obtain a list of ports and searches the list for matches by a regexp that follows the slashes (see Pythons `re` module for detailed syntax information).

Note that options are separated using the character `&`, this also applies to the first, where URLs usually use `?`. This exception is made as the question mark is used in regexp itself.

Depending on the capabilities of the `list_ports` module on the system, it is possible to search for the description or hardware ID of a device, e.g. USB VID:PID or texts.

Unfortunately, on some systems `list_ports` only lists a subset of the port names with no additional information. Currently, on Windows and Linux and OSX it should find additional information.

Supported options in the URL are:

- `n=N`: pick the N'th entry instead of the first
- `skip_busy`: skip ports that can not be opened, e.g. because they are already in use. This may not work as expected on platforms where the file is not locked automatically (e.g. Posix).

spy://

Wrapping the native serial port, this protocol makes it possible to intercept the data received and transmitted as well as the access to the control lines, break and flush commands. It is mainly used to debug applications.

Supported options in the URL are:

- `file=FILENAME` output to given file or device instead of stderr
- `color` enable ANSI escape sequences to colorize output
- `raw` output the read and written data directly (default is to create a hex dump). In this mode, no control line and other commands are logged.
- `all` also show `in_waiting` and empty `read()` calls (hidden by default because of high traffic).

Example:

```
import serial

with serial.serial_for_url('spy:///dev/ttyUSB0?file=test.txt', timeout=1) as s:
    s.dtr = False
    s.write('hello world')
    s.read(20)
    s.dtr = True
    s.write(serial.to_bytes(range(256)))
    s.read(400)
    s.send_break()

with open('test.txt') as f:
    print(f.read())
```

Outputs:

```
000000.002 Q-RX reset_input_buffer
000000.002 DTR  inactive
000000.002 TX   0000  68 65 6C 6C 6F 20 77 6F  72 6C 64                hello world
000001.015 RX   0000  68 65 6C 6C 6F 20 77 6F  72 6C 64                hello world
```

```
000001.015 DTR  active
000001.015 TX   0000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
000001.015 TX   0010  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
000001.015 TX   0020  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
000001.015 TX   0030  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
000001.015 TX   0040  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
000001.016 TX   0050  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
000001.016 TX   0060  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
000001.016 TX   0070  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
000001.016 TX   0080  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
000001.016 TX   0090  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
000001.016 TX   00A0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
000001.016 TX   00B0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
000001.016 TX   00C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
000001.016 TX   00D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ................
000001.016 TX   00E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ................
000001.016 TX   00F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ................
000002.284 RX   0000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ................
000002.284 RX   0010  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F  ................
000002.284 RX   0020  20 21 22 23 24 25 26 27  28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
000002.284 RX   0030  30 31 32 33 34 35 36 37  38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
000002.284 RX   0040  40 41 42 43 44 45 46 47  48 49 4A 4B 4C 4D 4E 4F  @ABCDEFGHIJKLMNO
000002.284 RX   0050  50 51 52 53 54 55 56 57  58 59 5A 5B 5C 5D 5E 5F  PQRSTUVWXYZ[\]^_
000002.284 RX   0060  60 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 6F  `abcdefghijklmno
000002.284 RX   0070  70 71 72 73 74 75 76 77  78 79 7A 7B 7C 7D 7E 7F  pqrstuvwxyz{|}~.
000002.284 RX   0080  80 81 82 83 84 85 86 87  88 89 8A 8B 8C 8D 8E 8F  ................
000002.284 RX   0090  90 91 92 93 94 95 96 97  98 99 9A 9B 9C 9D 9E 9F  ................
000002.284 RX   00A0  A0 A1 A2 A3 A4 A5 A6 A7  A8 A9 AA AB AC AD AE AF  ................
000002.284 RX   00B0  B0 B1 B2 B3 B4 B5 B6 B7  B8 B9 BA BB BC BD BE BF  ................
000002.284 RX   00C0  C0 C1 C2 C3 C4 C5 C6 C7  C8 C9 CA CB CC CD CE CF  ................
```

```
000002.284 RX   00D0  D0 D1 D2 D3 D4 D5 D6 D7  D8 D9 DA DB DC DD DE DF  ..............
↪..
000002.284 RX   00E0  E0 E1 E2 E3 E4 E5 E6 E7  E8 E9 EA EB EC ED EE EF  ..............
↪..
000002.284 RX   00F0  F0 F1 F2 F3 F4 F5 F6 F7  F8 F9 FA FB FC FD FE FF  ..............
↪..
000002.284 BRK  send_break 0.25
```

Another example, on POSIX, open a second terminal window and find out it's device (e.g. with the `ps` command in the TTY column), assumed to be `/dev/pts/2` here, double quotes are used so that the ampersand in the URL is not interpreted by the shell:

```
python -m serial.tools.miniterm "spy:///dev/ttyUSB0?file=/dev/pts/2&color" 115200
```

The spy output will be live in the second terminal window.

New in version 3.0.

alt://

This handler allows to select alternate implementations of the native serial port.

Currently only the POSIX platform provides alternative implementations.

PosixPollSerial Poll based read implementation. Not all systems support poll properly. However this one has better handling of errors, such as a device disconnecting while it's in use (e.g. USB-serial unplugged).

VTIMESerial Implement timeout using `VTIME`/`VMIN` of TTY device instead of using `select`. This means that inter character timeout and overall timeout can not be used at the same time. Overall timeout is disabled when inter-character timeout is used. The error handling is degraded.

Examples:

```
alt:///dev/ttyUSB0?class=PosixPollSerial
alt:///dev/ttyUSB0?class=VTIMESerial
```

New in version 3.0.

Examples

- `rfc2217://localhost:7000`
- `rfc2217://localhost:7000?poll_modem`
- `rfc2217://localhost:7000?ign_set_control&timeout=5.5`
- `socket://localhost:7777`
- `loop://?logging=debug`
- `hwgrep://0451:f432` (USB VID:PID)
- `spy://COM54?file=log.txt`
- `alt:///dev/ttyUSB0?class=PosixPollSerial`

CHAPTER 6

Examples

Miniterm

Miniterm is now available as module instead of example. see *serial.tools.miniterm* for details.

miniterm.py The miniterm program.

setup-miniterm-py2exe.py This is a py2exe setup script for Windows. It can be used to create a standalone `miniterm.exe`.

TCP/IP - serial bridge

This program opens a TCP/IP port. When a connection is made to that port (e.g. with telnet) it forwards all data to the serial port and vice versa.

This example only exports a raw socket connection. The next example below gives the client much more control over the remote serial port.

- The serial port settings are set on the command line when starting the program.
- There is no possibility to change settings from remote.
- All data is passed through as-is.

```
usage: tcp_serial_redirect.py [-h] [-q] [--parity {N,E,O,S,M}] [--rtscts]
                              [--xonxoff] [--rts RTS] [--dtr DTR]
                              [-P LOCALPORT]
                              SERIALPORT [BAUDRATE]

Simple Serial to Network (TCP/IP) redirector.

positional arguments:
  SERIALPORT            serial port name
  BAUDRATE              set baud rate, default: 9600
```

```
optional arguments:
  -h, --help            show this help message and exit
  -q, --quiet           suppress non error messages

serial port:
  --parity {N,E,O,S,M}  set parity, one of {N E O S M}, default: N
  --rtscts              enable RTS/CTS flow control (default off)
  --xonxoff             enable software flow control (default off)
  --rts RTS             set initial RTS line state (possible values: 0, 1)
  --dtr DTR             set initial DTR line state (possible values: 0, 1)

network settings:
  -P LOCALPORT, --localport LOCALPORT
                        local TCP port

NOTE: no security measures are implemented. Anyone can remotely connect to
this service over the network. Only one connection at once is supported. When
the connection is terminated it waits for the next connect.
```

tcp_serial_redirect.py Main program.

Single-port TCP/IP - serial bridge (RFC 2217)

Simple cross platform **RFC 2217** serial port server. It uses threads and is portable (runs on POSIX, Windows, etc).

- The port settings and control lines (RTS/DTR) can be changed at any time using **RFC 2217** requests. The status lines (DSR/CTS/RI/CD) are polled every second and notifications are sent to the client.
- Telnet character IAC (0xff) needs to be doubled in data stream. IAC followed by another value is interpreted as Telnet command sequence.
- Telnet negotiation commands are sent when connecting to the server.
- RTS/DTR are activated on client connect and deactivated on disconnect.
- Default port settings are set again when client disconnects.

```
usage: rfc2217_server.py [-h] [-p TCPPORT] [-v] SERIALPORT

RFC 2217 Serial to Network (TCP/IP) redirector.

positional arguments:
  SERIALPORT

optional arguments:
  -h, --help            show this help message and exit
  -p TCPPORT, --localport TCPPORT
                        local TCP port, default: 2217
  -v, --verbose         print more diagnostic messages (option can be given
                        multiple times)

NOTE: no security measures are implemented. Anyone can remotely connect to
this service over the network. Only one connection at once is supported. When
the connection is terminated it waits for the next connect.
```

New in version 2.5.

rfc2217_server.py Main program.

setup-rfc2217_server-py2exe.py This is a py2exe setup script for Windows. It can be used to create a standalone `rfc2217_server.exe`.

Multi-port TCP/IP - serial bridge (RFC 2217)

This example implements a TCP/IP to serial port service that works with multiple ports at once. It uses select, no threads, for the serial ports and the network sockets and therefore runs on POSIX systems only.

- Full control over the serial port with **RFC 2217**.
- Check existence of `/tty/USB0...8`. This is done every 5 seconds using `os.path.exists`.
- Send zeroconf announcements when port appears or disappears (uses python-avahi and dbus). Service name: `_serial_port._tcp`.
- Each serial port becomes available as one TCP/IP server. e.g. `/dev/ttyUSB0` is reachable at `<host>:7000`.
- Single process for all ports and sockets (not per port).
- The script can be started as daemon.
- Logging to stdout or when run as daemon to syslog.
- Default port settings are set again when client disconnects.
- modem status lines (CTS/DSR/RI/CD) are not polled periodically and the server therefore does not send NOTIFY_MODEMSTATE on its own. However it responds to request from the client (i.e. use the `poll_modem` option in the URL when using a pySerial client.)

```
usage: port_publisher.py [options]

Announce the existence of devices using zeroconf and provide
a TCP/IP <-> serial port gateway (implements RFC 2217).

If running as daemon, write to syslog. Otherwise write to stdout.

optional arguments:
  -h, --help            show this help message and exit

serial port settings:
  --ports-regex REGEX   specify a regex to search against the serial devices
                        and their descriptions (default: /dev/ttyUSB[0-9]+)

network settings:
  --tcp-port PORT       specify lowest TCP port number (default: 7000)

daemon:
  -d, --daemon          start as daemon
  --pidfile FILE        specify a name for the PID file

diagnostics:
  -o FILE, --logfile FILE
                        write messages file instead of stdout
  -q, --quiet           suppress most diagnostic messages
  -v, --verbose         increase diagnostic messages

NOTE: no security measures are implemented. Anyone can remotely connect to
```

```
this service over the network. Only one connection at once, per port, is
supported. When the connection is terminated, it waits for the next connect.
```

Requirements:

- Python (>= 2.4)
- python-avahi
- python-dbus
- python-serial (>= 2.5)

Installation as daemon:

- Copy the script `port_publisher.py` to `/usr/local/bin`.
- Copy the script `port_publisher.sh` to `/etc/init.d`.
- Add links to the runlevels using `update-rc.d port_publisher.sh defaults 99`
- That's it :-) the service will be started on next reboot. Alternatively run `invoke-rc.d port_publisher.sh start` as root.

New in version 2.5: new example

port_publisher.py Multi-port TCP/IP-serial converter (RFC 2217) for POSIX environments.

port_publisher.sh Example init.d script.

wxPython examples

A simple terminal application for wxPython and a flexible serial port configuration dialog are shown here.

wxTerminal.py A simple terminal application. Note that the length of the buffer is limited by wx and it may suddenly stop displaying new input.

wxTerminal.wxg A wxGlade design file for the terminal application.

wxSerialConfigDialog.py A flexible serial port configuration dialog.

wxSerialConfigDialog.wxg The wxGlade design file for the configuration dialog.

setup-wxTerminal-py2exe.py A py2exe setup script to package the terminal application.

Unit tests

The project uses a number of unit test to verify the functionality. They all need a loop back connector. The scripts itself contain more information. All test scripts are contained in the directory `test`.

The unit tests are performed on port `loop://` unless a different device name or URL is given on the command line (`sys.argv[1]`). e.g. to run the test on an attached USB-serial converter `hwgrep://USB` could be used or the actual name such as `/dev/ttyUSB0` or `COM1` (depending on platform).

run_all_tests.py Collect all tests from all `test*` files and run them. By default, the `loop://` device is used.

test.py Basic tests (binary capabilities, timeout, control lines).

test_advanced.py Test more advanced features (properties).

test_high_load.py Tests involving sending a lot of data.

test_readline.py Tests involving `readline`.

test_iolib.py Tests involving the `io` library. Only available for Python 2.6 and newer.

test_url.py Tests involving the *URL* feature.

CHAPTER 7

Appendix

How To

Enable RFC 2217 (and other URL handlers) in programs using pySerial. Patch the code where the *serial.Serial* is instantiated. E.g. replace:

```
s = serial.Serial(...)
```

it with:

```
s = serial.serial_for_url(...)
```

or for backwards compatibility to old pySerial installations:

```
try:
    s = serial.serial_for_url(...)
except AttributeError:
    s = serial.Serial(...)
```

Assuming the application already stores port names as strings that's all that is required. The user just needs a way to change the port setting of your application to an `rfc2217://` *URL* (e.g. by editing a configuration file, GUI dialog etc.).

Please note that this enables all *URL* types supported by pySerial and that those involving the network are unencrypted and not protected against eavesdropping.

Test your setup. Is the device not working as expected? Maybe it's time to check the connection before proceeding. *serial.tools.miniterm* from the *Examples* can be used to open the serial port and do some basic tests.

To test cables, connecting RX to TX (loop back) and typing some characters in *serial.tools.miniterm* is a simple test. When the characters are displayed on the screen, then at least RX and TX work (they still could be swapped though).

There is also a `spy://` URL handler. It prints all calls (read/write, control lines) to the serial port to a file or stderr. See *spy://* for details.

FAQ

Example works in *serial.tools.miniterm* but not in script. The RTS and DTR lines are switched when the port is opened. This may cause some processing or reset on the connected device. In such a cases an immediately following call to `write()` may not be received by the device.

A delay after opening the port, before the first `write()`, is recommended in this situation. E.g. a `time.sleep(1)`

Application works when .py file is run, but fails when packaged (py2exe etc.) py2exe and similar packaging programs scan the sources for import statements and create a list of modules that they package. pySerial may create two issues with that:

- implementations for other modules are found. On Windows, it's safe to exclude 'serialposix', 'serialjava' and 'serialcli' as these are not used.
- *`serial.serial_for_url()`* does a dynamic lookup of protocol handlers at runtime. If this function is used, the desired handlers have to be included manually (e.g. 'serial.urlhandler.protocol_socket', 'serial.urlhandler.protocol_rfc2217', etc.). This can be done either with the "includes" option in `setup.py` or by a dummy import in one of the packaged modules.

User supplied URL handlers *`serial.serial_for_url()`* can be used to access "virtual" serial ports identified by an *URL* scheme. E.g. for the **RFC 2217**: `rfc2217://`.

Custom *URL* handlers can be added by extending the module search path in *`serial.protocol_handler_packages`*. This is possible starting from pySerial V2.6.

`Permission denied` errors On POSIX based systems, the user usually needs to be in a special group to have access to serial ports.

On Debian based systems, serial ports are usually in the group `dialout`, so running `sudo adduser $USER dialout` (and logging-out and -in) enables the user to access the port.

Support for Python 2.6 or earlier Support for older Python releases than 2.7 will not return to pySerial 3.x. Python 2.7 is now many years old (released 2010). If you insist on using Python 2.6 or earlier, it is recommend to use pySerial 2.7 (or any 2.x version).

Related software

com0com - http://com0com.sourceforge.net/ Provides virtual serial ports for Windows.

License

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

H

V

W

X