python-xbee Documentation

Release 2.3.1

Paul Malmsten

Jul 05, 2017

Contents

CHAPTER 1

Introduction

The purpose of XBee is to allow one easy access to the advanced features of an XBee device from a Python application. It provides a semi-complete implementation of the XBee binary API protocol and allows a developer to send and receive the information they desire without dealing with the raw communication details.

Note: This library is compatible with both XBee 802.15.4 (Series 1) and XBee ZigBee (Series 2) modules, normal and PRO. The following examples are applicable to XBee 802.15.4 modules; to follow these examples with a XBee ZigBee device or a Series 1 device loaded with Digimesh firmware, change the line:

```
xbee = XBee(serial_port)
```

to:

```
xbee = ZigBee(serial_port)
```

CHAPTER 2

Usage

Note: In order to use an XBee device with this library, API mode must be enabled. For instructions about how to do this, see the documentation for your XBee device.

Note: The default implementation of the python-xbee library is to use threads. There is an implementation that utilizes the Tornado IOLoop, if imported.

Threaded Synchonous Mode

The following code demonstrates a minimal use-case for the xbee package:

```
import serial
from xbee import XBee

serial_port = serial.Serial('/dev/ttyUSB0', 9600)
xbee = XBee(serial_port)

while True:
    try:
        print xbee.wait_read_frame()
    except KeyboardInterrupt:
        break

serial_port.close()
```

This example will perpetually read from the serial port and print out any data frames which arrive from a connected XBee device. Be aware that wait_read_frame() will block until a valid frame is received from the associated XBee device.

Threaded Asynchronous Mode

The xbee package is used only slightly differently when asynchonous notification of received data is needed:

```
import serial
import time
from xbee import XBee

serial_port = serial.Serial('/dev/ttyUSB0', 9600)

def print_data(data):
    """
    This method is called whenever data is received
    from the associated XBee device. Its first and
    only argument is the data contained within the
    frame.
    """
    print data

xbee = XBee(serial_port, callback=print_data)

while True:
    try:
        time.sleep(0.001)
    except KeyboardInterrupt:
        break

xbee.halt()
serial_port.close()
```

Warning: When asychonous mode is enabled, the provided callback method is called by a background thread managed by the xbee package. Make sure that updates to external state are thread-safe.

Note that a background thread is automatically started to handle receiving and processing incoming data from an XBee device. This example is functionally equivalent to the non-asyncronous example above.

Tornado IOLoop

Tornado provides a simple and easy to use IOLoop for asynchronous listening for XBee communications. The library usage is seemingly identical to the threaded implementation, excepting importing and yielding. The example highlights the key differences:

```
import serial
from tornado import ioloop, gen
from xbee.tornado import XBee

serial_port = serial.Serial('/dev/ttyUSB0', 9600)

def print_data(data):
    """
    This method is called whenever data is received
    from the associated XBee device. Its first and
    only argument is the data contained within the
```

```
    frame.
    """
    print data

@gen.coroutine
def main():
    xbee = XBee(serial_port, callback=print_data)

    try:
        while True:
            yield gen.sleep(0.001)
    except KeyboardInterrupt:
        ioloop.IOLoop.current().stop()
    finally
        xbee.halt()
        serial_port.close()

ioloop.IOLoop.current().spawn_callback(main)
ioloop.IOLoop.current().start()
ioloop.IOLoop.current().close()
```

Additional Examples

For additional examples, look in the examples/ directory contained within the xbee package source code archive or source control.

CHAPTER 3

API

Frame Data Format

Information returned from this library is a dictionary in the following format:

```
{'id':str,
 'param':binary data,
 ...}
```

The id field is always a human-readable name of the packet type received. All following fields, shown above with the key 'param', map binary data to each of the possible fields contained within the received data frame.

Note: A listing of all supported data frames and their respective fields may be found in xbee.ieee.XBee (or xbee.zigbee.ZigBee for XBee ZB devices) defined as api_responses.

Sample Data Format

Sample data is returned in the following format:

```
[ {"dio-0":True,
   "dio-1":False,
   "adc-0":100"}, ...]
```

This format is a list of dictionaries. Each dictionary represents one sample, listed in chronological order. Each sample dictionary can contain any number of digital (dio) and analog (adc) pin samples. The keys of a sample dictionary will always follow this pattern:

```
(dio|adc)-[0-9]+
```

The number of dio and adc values returned depends upon the type and configuration of the XBee device used with this library.

Response Parsing

As of version 2.1.0

For XBee devices, sample data within I/O sample messages is automatically parsed into the above format. In addition, when an "IS", Force Sample, AT command is issued, either to a local device or a remote device, the value of the "parameter" field in the response will be automatically parsed as I/O sample data.

ZigBee devices extend this behavior to include automatic parsing of "ND", Node Discover, AT command responses. The parameter field of a ND AT response will assume the following format:

```
{"source_addr":      two bytes,
 "source_addr_long": eight bytes,
 "node_identifier":  string,
 "parent_address":   two bytes,
 "device_type":      one byte,
 "status":           one byte,
 "profile_id":       two bytes,
 "manufacturer":     two bytes,
}
```

Sending Data to an XBee Device

In order to send data to an XBee device, use the send() method:

```
xbee.send("at", frame='A', command='MY', parameter=None)
```

This example will request the 16-bit address of the connected XBee device with an API frame marker of 'A'.

For your convenience, some optional data fields are not required. For example, the 'parameter' field may be ommitted if empty:

```
xbee.send("at", frame='A', command='MY')
```

Additionally, an alternate syntax is provided for all supported commands::

```
xbee.at(frame='A', command='MY')
```

This example is functionally equivalent to the line(s) above.

Lastly, if an API frame identifier is not needed, the command may be reduced to:

```
xbee.at(command='MY')
```

Note: A listing of all supported commands and their associated data fields may be found in xbee.ieee.XBee (xbee.zigbee.ZigBee for XBee ZB devices) defined as api_commands.

API Reference

class xbee.**XBee**(**args*, ***kwargs*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

x

Index

X