# Pygate Driver

This driver is designed to compliment the 'Pygate for P2P'[1] firmware modification to enable easy transmission and receipt of packets using a Pycom Pygate. The driver reference assumes that the firmware modification has been flashed onto the Pycom Pygate.

## Initialisation

The Pygate driver is simply initialised by instantiating the Pygate class as follows.

```
Pygate = Pygate()
```

It is important to note that a Gateway Configuration file, 'config.json' must be placed in the directory '/flash/config.json'. This file configures each of the eight Pygate channels. An example is provided in the 'gw-config' directory.

Note: if the frequencies are changed then the Pygate Driver must be modified. The frequencies in the 'set_pygate_channels' function should match that in the 'config.json'.

## Sending Packets

Two functions are provided for sending LoRa packets.

### send

'send' is the basic transmit function in the driver.

```
send(sf, radio, freq, payload)
```

'sf'—Sets which Spreading Factor the packet should be transmitted with.

'radio'—Sets which transceiver module the packet should be transmitted from (0 or 1).

'freq'—Sets the frequency at which the packet should be transmitted. This must match a frequency specified in the gateway configuration file.

'payload'—The packet payload to transmit.

### send_ch

'send_ch' is a function provided for ease of use which simply allows the user to specify which of the predefined channels the packet will be transmitted on.

```
send_ch(sf, channel, payload)
```

'sf'—Sets which Spreading Factor the packet should be transmitted with.

'channel'—Which of the 8 predefined channels to send on. 0 to 7 inclusive.

[1] Pygate for P2P, https://github.com/RupertHSmith/Pygate-for-P2P

‘payload’—Packet payload to transmit.

*Table 1 | Default channel configurations.*

```
        self.pygate_channels = []
        self.pygate_channels.append((0,867.1))
        self.pygate_channels.append((0,867.3))
        self.pygate_channels.append((0,867.5))
        self.pygate_channels.append((0,867.7))
        self.pygate_channels.append((0,867.9))
        self.pygate_channels.append((0,868.1))
        self.pygate_channels.append((0,868.3))
        self.pygate_channels.append((0,868.5))
```

## Receiving LoRa Packets

Pygates are received by calling the function ‘receive_lora’ which is a non-blocking function.

```
receive_lora()
```

If no LoRa packets have been received then the tuple `(None, None)` is returned. Otherwise a tuple `(header, payload)` is returned. ‘header’ is the packet metadata detailed in Table 2. ‘payload’ is the payload as an array of Bytes.

*Table 2 | Fields of Pygate Header JSON object.*

| ***Name*** | ***Description*** |
|---|---|
| *rssi* | RSSI of packet in dBm |
| *snr* | SNR of packet in dB |
| *chan* | Channel received on (0-7) |
| *rfch* | Radio module received on (0 or 1) |
| *freq* | Frequency received on (MHz) |
| *datr* | SF and Bandwidth combined as SF<SF>BW<Bandwidth in MHz> |
| *codr* | Code Rate (default: ‘4/5’) |

## Receiving LoRa TOA of Last Transmission

In order to retrieve the Time On Air (TOA) of the last transmission, a subsequent call to ‘receive_stat’ should be made which is a non-blocking function.

```
receive_lora()
```

If no ‘Time On Air’ information packets have been received from the concentrator then the function will return None. Otherwise the TOA of the last packet is returned in milliseconds.