**class LoRa**

This class provides a driver for the LoRa network processor in the **LoPy**. Example usage:

**from** network **import** LoRa

**import** socket

**import** binascii

**import** struct

*# Initialize LoRa in LORAWAN mode.*

lora **=** LoRa(mode**=**LoRa**.**LORAWAN)

*# create an ABP authentication params*

dev\_addr **=** struct**.**unpack(">l", binascii**.**unhexlify('00 00 00 05'**.**replace(' ','')))[0]

nwk\_swkey **=** binascii**.**unhexlify('2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C'**.**replace(' ',''))

app\_swkey **=** binascii**.**unhexlify('2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C'**.**replace(' ',''))

*# join a network using ABP (Activation By Personalization)*

lora**.**join(activation**=**LoRa**.**ABP, auth**=**(dev\_addr, nwk\_swkey, app\_swkey))

*# create a LoRa socket*

s **=** socket**.**socket(socket**.**AF\_LORA, socket**.**SOCK\_RAW)

*# set the LoRaWAN data rate*

s**.**setsockopt(socket**.**SOL\_LORA, socket**.**SO\_DR, 5)

*# make the socket non-blocking*

s**.**setblocking(**False**)

*# send some data*

s**.**send(bytes([0x01, 0x02, 0x03]))

*# get any data received...*

data **=** s**.**recv(64)

print(data)

**Additional examples**

Check here for [aditional examples](https://docs.pycom.io/pycom_esp32/pycom_esp32/tutorial/lopy.html#lora-examples).

**Constructors**

***class*network.LoRa(*id=0*, *...*)**

Create and configure a LoRa object. See **init** for params of configuration.

**Methods**

**lora.init(*mode*, *\**, *frequency=868000000*, *tx\_power=14*, *bandwidth=LoRa.868000000*, *sf=7*, *preamble=8*, *coding\_rate=LoRa.CODING\_4\_5*, *power\_mode=LoRa.ALWAYS\_ON*, *tx\_iq=false*, *rx\_iq=false*, *adr=false*, *public=true*, *tx\_retries=1*)**

Set the LoRa subsystem configuration

The arguments are:

* **mode** can be either **LoRa.LORA** or **LoRa.LORAWAN**.
* **frequency** accepts values between 863000000 and 870000000 in the 868 band, or between 902000000 and 928000000 in the 915 band.
* **tx\_power** is the transmit power in dBm. It accepts between 2 and 14 for the 868 band, and between 5 and 20 in the 915 band.
* **bandwidth** is the channel bandwidth in KHz. In the 868 band the accepted values are **LoRa.BW\_125KHZ** and **LoRa.BW\_250KHZ**. In the 915 band the accepted values are **LoRa.BW\_125KHZ** and **LoRa.BW\_500KHZ**.
* **sf** sets the desired spreading factor. Accepts values between 7 and 12.
* **preamble** configures the number of pre-amble symbols. The default value is 8.
* **coding\_rate** can take the following values: **LoRa.CODING\_4\_5**, **LoRa.CODING\_4\_6**,**LoRa.CODING\_4\_7** or **LoRa.CODING\_4\_8**.
* **power\_mode** can be either **LoRa.ALWAYS\_ON**, **LoRa.TX\_ONLY** or **LoRa.SLEEP**. In **ALWAYS\_ON** mode, the radio is always listening for incoming packets whenever a transmission is not taking place. In **TX\_ONLY** the radio goes to sleep as soon as the transmission completes. In **SLEEP** mode the radio is sent to sleep permanently and won’t accept any commands until the power mode is changed.
* **tx\_iq** enables TX IQ inversion.
* **rx\_iq** enables RX IQ inversion.
* **adr** enables Adaptive Data Rate.
* **public** selects between the public and private sync word.
* **tx\_retries** sets the number of TX retries in **LoRa.LORAWAN** mode.

**Note**

In **LoRa.LORAWAN** mode, only **adr**, **public** and **tx\_retries** are used. All the other params will be ignored as they are handled by the LoRaWAN stack directly. On the other hand, in **LoRa.LORA** mode from those 3 arguments, only the **public** one is important in order to program the sync word. In **LoRa.LORA** mode **adr** and **tx\_retries** are ignored since they are only relevant to the LoRaWAN stack.

For example, you can do:

*# initialize in raw LoRa mode*

lora**.**init(mode**=**LoRa**.**LORA, tx\_power**=**14, sf**=**12)

or:

*# initialize in LoRaWAN mode*

lora**.**init(mode**=**LoRa**.**LORAWAN)

**lora.join(*activation*, *auth*, *\**, *timeout=None*)**

Join a LoRaWAN network. The parameters are:

* **activation**: can be either **LoRa.OTAA** or **LoRa.ABP**.
* **auth**: is a tuple with the authentication data.

In the case of **LoRa.OTAA** the authentication tuple is: **(app\_eui, app\_key)**. Example:

**from** network **import** LoRa

**import** socket

**import** time

**import** binascii

*# Initialize LoRa in LORAWAN mode.*

lora **=** LoRa(mode**=**LoRa**.**LORAWAN)

*# create an OTAA authentication parameters*

app\_eui **=** binascii**.**unhexlify('AD A4 DA E3 AC 12 67 6B'**.**replace(' ',''))

app\_key **=** binascii**.**unhexlify('11 B0 28 2A 18 9B 75 B0 B4 D2 D8 C7 FA 38 54 8B'**.**replace(' ',''))

*# join a network using OTAA (Over the Air Activation)*

lora**.**join(activation**=**LoRa**.**OTAA, auth**=**(app\_eui, app\_key), timeout**=**0)

*# wait until the module has joined the network*

**while** **not** lora**.**has\_joined():

time**.**sleep(2.5)

print('Not yet joined...')

In the case of **LoRa.ABP** the authentication tuple is: **(dev\_addr, nwk\_swkey, app\_swkey)**. Example:

**from** network **import** LoRa

**import** socket

**import** binascii

**import** struct

*# Initialize LoRa in LORAWAN mode.*

lora **=** LoRa(mode**=**LoRa**.**LORAWAN)

*# create an ABP authentication params*

dev\_addr **=** struct**.**unpack(">l", binascii**.**unhexlify('00 00 00 05'**.**replace(' ','')))[0]

nwk\_swkey **=** binascii**.**unhexlify('2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C'**.**replace(' ',''))

app\_swkey **=** binascii**.**unhexlify('2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C'**.**replace(' ',''))

*# join a network using ABP (Activation By Personalization)*

lora**.**join(activation**=**LoRa**.**ABP, auth**=**(dev\_addr, nwk\_swkey, app\_swkey))

**lora.bandwidth([*bandwidth*])**

Get or set the bandwidth in raw LoRa mode (**LoRa.LORA**). Can be either **LoRa.BW\_125KHZ**, **LoRa.BW\_250KHZ** or **LoRa.BW\_500KHZ**.

**lora.frequency([*frequency*])**

Get or set the frequency in raw LoRa mode (**LoRa.LORA**). The allowed range is between 863000000 and 870000000 Hz for the 868MHz band version or between 902000000 and 928000000 Hz for the 915MHz abdn version.

**lora.coding\_rate([*coding\_rate*])**

Get or set the coding rate in raw LoRa mode (**LoRa.LORA**). The allowed values are: **LoRa.CODING\_4\_5**, **LoRa.CODING\_4\_6**, **LoRa.CODING\_4\_7** and **LoRa.CODING\_4\_8**.

**lora.preamble([*preamble*])**

Get or set the number of preamble symbols in raw LoRa mode (**LoRa.LORA**).

**lora.sf([*sf*])**

Get or set the spreading factor value in raw LoRa mode (**LoRa.LORA**). The minimmum value is 7 and the maximum is 12.

**lora.power\_mode([*power\_mode*])**

Get or set the power mode in raw LoRa mode (**LoRa.LORA**). The accepted values are: **LoRa.ALWAYS\_ON**, **LoRa.TX\_ONLY** and **LoRa.SLEEP**.

**lora.stats()**

Return a named tuple with usefel information from the last received LoRa or LoRaWAN packet. The named tuple has the following form:

**(timestamp, rssi, snr, sf)**

Where:

* **timestamp** is an internat timestamp with microseconds presicion.
* **rssi** hold the received signal strength in dBm.
* **snr** contains the signal to noise ratio id dB.
* **sf** tells the spreading factor of the packet received.

**lora.has\_joined()**

Returns **True** if a LoRaWAN network has been joined. **False** otherwise.

**lora.add\_channel(*index*, *\**, *frequency*, *dr\_min*, *dr\_max*)**

Add a LoRaWAN channel on the specified index. If there’s already a channel with that index it will be replaced with the new one.

The arguments are:

* **index**: Index of the channel to add. Accepts values between 0 and 15 for EU and between 0 and 71 for US.
* **frequency**: Center frequency in Hz of the channel.
* **dr\_min**: Minimum data rate of the channel (0-7).
* **dr\_max**: Maximum data rate of the channel (0-7).

**lora.remove\_channel(*index*)**

Removes the channel from the specified index. On the 868MHz band the channels 0 to 2 cannot be removed, they can only be replaced by other channels using the **lora.add\_channel** method. A way to remove all channels except for one is to add the same channel 3 times on indexes 0, 1 and 2.

On the 915MHz band there are no restrictions around this.

**lora.mac()**

Returns a byte object with the 8-Byte MAC address of the LoRa radio.

**lora.callback(*trigger*, *handler=None*, *arg=None*)**

Specify a callback handler for the LoRa radio. The trigger types are **LoRa.RX\_PACKET\_EVENT**and **LoRa.TX\_PACKET\_EVENT**

**lora.events()**

This method returns a value with bits sets (if any) indicating the events that have triggered the callback. Please note that by calling this function the internal events registry is cleared automatically, therefore calling it immediaelly for a second time will most likely return a value of 0. Example:

**def** **lora\_cb**(lora):

events **=** lora**.**events()

**if** events **&** LoRa**.**RX\_PACKET\_EVENT:

print('Lora packet received')

**if** events **&** LoRa**.**TX\_PACKET\_EVENT:

print('Lora packet sent')

lora**.**callback(trigger**=**(LoRa**.**RX\_PACKET\_EVENT **|** LoRa**.**TX\_PACKET\_EVENT), handler**=**lora\_cb)

**Constants**

**LoRa.LORA**

**LoRa.LORAWAN**

LoRa mode

**LoRa.OTAA**

**LoRa.ABP**

LoRaWAN join procedure

**LoRa.ALWAYS\_ON**

**LoRa.TX\_ONLY**

**LoRa.SLEEP**

Raw LoRa power mode

**LoRa.BW\_125KHZ**

**LoRa.BW\_250KHZ**

**LoRa.BW\_500KHZ**

Raw LoRa bandwidth

**LoRa.CODING\_4\_5**

**LoRa.CODING\_4\_6**

**LoRa.CODING\_4\_7**

**LoRa.CODING\_4\_8**

Raw LoRa coding rate

**LoRa.RX\_PACKET\_EVENT**

**LoRa.TX\_PACKET\_EVENT**

Callback trigger types (may be ORed)

**Working with LoRa and LoRaWAN sockets**

LoRa sockets are created in the following way:

**import** socket

s **=** socket**.**socket(socket**.**AF\_LORA, socket**.**SOCK\_RAW)

And they **must** be created after initializing the LoRa network card.

LoRa sockets support the following standard methods from the **socket** module:

**socket.close()**

Usage: **s.close()**

**socket.bind(*port\_number*)**

Usage: **s.bind(1)**

**Note**

The **.bind()** method is only applicable when the radio is configured in **LoRa.LORAWAN** mode.

**socket.send(*bytes*)**

Usage: **s.send(bytes([1, 2, 3]))** or: **s.send('Hello')**

**socket.recv(*bufsize*)**

Usage: **s.recv(128)**

**socket.setsockopt(*level*, *optname*, *value*)**

Set the value of the given socket option. The needed symbolic constants are defined in the socket module (SO\_\* etc.). In the case of LoRa the values are always integers. Examples:

*# configuring the data rate*

s**.**setsockopt(socket**.**SOL\_LORA, socket**.**SO\_DR, 5)

*# selecting non-confirmed type of messages*

s**.**setsockopt(socket**.**SOL\_LORA, socket**.**SO\_CONFIRMED, **False**)

*# selecting confirmed type of messages*

s**.**setsockopt(socket**.**SOL\_LORA, socket**.**SO\_CONFIRMED, **True**)

**Note**

Socket options are only applicable when the LoRa radio is used in **LoRa.LORAWAN** mode. When using the radio in **LoRa.LORA** mode, use the class methods to change the spreading factor, bandwidth and coding rate to the desired values.

**socket.settimeout(*value*)**

Sets the socket timeout value in seconds. Accepts floating point values.

Usage: **s.settimeout(5.5)**

**socket.setblocking(*flag*)**

Usage: **s.setblocking(True)**