hachi Documentation

Release 0.3

Antoine Bertin

July 16, 2013

CONTENTS

Release v0.3

Hachi is a Python library to interact with XBees.

CHAPTER

ONE

SERIAL IMPLEMENTATION

Read XBeeResponse:

```
>>> import hachi
>>> from hachi.serial import XBeeSerial
>>> x = XBeeSerial('/dev/ttyUSB0')
>>> x.read_response()
<ZBIoSampleResponse(len=18)>
```

Send XBeeRequest:

```
>>> request = hachi.AtRequest('ID', 0xff)
>>> x.send(request)
>>> response = x.read_response()
>>> response
<AtResponse(len=xx)>
>>> response.status == hachi.COMMAND_STATUS_OK
True
```

CHAPTER

TWO

TWISTED IMPLEMENTATION

Use the XBeeProtocol:

```
>>> import hachi
>>> from hachi.twisted import XBeeProtocol
>>> from twisted.internet import reactor
>>> from twisted.internet.serialport import SerialPort
>>> class TestXBee(XBeeProtocol):
...     def responseReceived(self, response):
...         print(response)
...
>>> serial = SerialPort(TestXBee(), '/dev/ttyUSB0', reactor, baudrate=9600)
>>> reactor.run()
<ZBIoSampleResponse(len=18)>
<ZBIoSampleResponse(len=18)>
<ZBIoSampleResponse(len=18)>
```

CHAPTER

THREE

API DOCUMENTATION

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

3.1 Core

class `hachi.core.`**`XBee`**(*callback=None*)

Parser for incoming XBee communications

The `XBee` parses data through its `feed()` method. When a complete and valid response is found, the *response* attribute is set to the corresponding `XBeeResponse` and the callback is called. Any malformed response is silently discarded.

Parameters **callback** (*function*) – callback method called with a `XBeeResponse` as first positional argument

`reset`()

Reset the state of the parser

`feed`(*data*)

Feed the parser with data

Parameters **data** (*int or bytes or bytearray*) – byte(s) to add

`hachi.core.`**`escape`**(*byte*)

Escape a byte

Parameters **byte** (*int*) – the byte to escape

Raise ValueError if the byte is not a special byte

Returns the escaped byte

Return type int

`hachi.core.`**`escape_frame`**(*frame*)

Escape a frame

Parameters **frame** (*bytearray*) – the frame to escape, starting with `FRAME_DELIMITER`

Raise ValueError if the frame does not start with a `FRAME_DELIMITER`

Returns the escaped frame

Return type bytearray

`hachi.core.`**`unescape`**(*byte*)

Unescape a byte

Parameters **byte** (*int*) – the byte to unescape

Raise ValueError if the unescaped byte is not a special byte

Returns the unescaped byte

Return type int

3.2 Response

3.2.1 Responses

class `hachi.response.`**`XBeeResponse`**(*frame*)

Base class for all XBee responses

The `XBeeResponse` is a wrapper around the underlying raw API frame.

Parameters **frame** (*bytearray*) – unescaped raw API frame

`api_id` = **None**

API ID

Subclasses must implement this and return the corresponding API ID

`frame` = **None**

Unescaped raw API frame

`length`

Length, on 2 bytes starting right after the `FRAME_DELIMITER` of the `frame`

`id_data`

API ID-specific raw data, bytes between the `api_id` and the `checksum`

Subclasses may provide properties to access API ID-specific data

`checksum`

Checksum, last byte of the `frame`

`verify`()

Check if the response has a valid `checksum`

Returns *True* if the response has a valid `checksum`, *False* otherwise

Return type bool

class `hachi.response.`**`Rx64Response`**(*frame*)

Response to a `Tx64Request`

Frame example: 7E 00 10 80 00 13 A2 00 40 52 2B AA 16 03 F1 2E AA BD C9 FB

`source_address`

Source address, first 8 bytes of the `id_data`

`rssi`

RSSI, immediatly following the `source_address`

`options`

Options, immediatly following the `rssi`

`data`

Data, all bytes following the `options` until the end of the `id_data`

class `hachi.response.`**`Rx16Response`**(*frame*)

Response to a `Tx16Request`

Frame example: 7E 00 0A 81 52 1A 23 01 12 33 85 A1 F2 91'

`source_address`
Source address, first 2 bytes of the `id_data`

`rssi`
RSSI, immediatly following the `source_address`

`options`
Options, immediatly following the `rssi`

`data`
Data, all bytes following the `options` until the end of the `id_data`

class `hachi.response.`**`Rx64IoSampleResponse`**(*frame*)

IO sample response using 64-bits addressing

Frame example: 7E 00 14 82 00 13 A2 00 40 52 2B AA 23 01 02 14 88 00 80 00 8F 03 ED 00 08 02 4C 00 0C 3E

`source_address`
Source address, first 8 bytes of the `id_data`

`rssi`
RSSI, immediatly following the `source_address`

`options`
Options, immediatly following the `rssi`

`sample_count`
Sample count, immediatly following the `options`

`analog_mask`
Analog mask, immediatly following the `sample_count`

It is the whole byte of which first and last bit are set to 0 because useless in this context. Second bit (right to left) tells if analog channel 0 (A0) is enabled (bit to 1) or disabled (bit to 0), third bit is for A1 and so on until seventh bit which is for A5

`digital_mask`
Digital mask, on 2 bytes including the `analog_mask` byte

It is the whole first byte (right to left) and the second one of which all bits except the first one are set to 0 because useless in this context. First bit (right to left) tells if digital channel 0 (D0) is enabled (bit to 1) or disabled (bit to 0), second bit is for D1 and so on until ninth bit which is for D8

`contains_analog`
True if at least one analog channel is enabled, *False* otherwise

`contains_digital`
True if at least one digital channel is enabled, *False* otherwise

`is_analog_enabled`(*pin*)
Tells if a given analog pin is enabled or not

Parameters **pin** (*int*) – analog pin number

Returns *True* if the given analog pin is enabled, *False* otherwise

Return type boolean

is_digital_enabled(*pin*)
Tells if a given digital pin is enabled or not

Parameters **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is enabled, *False* otherwise

Return type boolean

is_digital_on(*index*, *pin*)
Tells if a given digital pin is on or not in a sample

Parameters

- **index** (*int*) – index of the sample
- **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is on in the sample, *False* otherwise

Return type boolean

get_analog(*index*, *pin*)
Gives the analog value of a pin in a sample

Parameters

- **index** (*int*) – index of the sample
- **pin** (*int*) – analog pin number

Returns analog pin value

Return type int

class hachi.response.**Rx16IoSampleResponse**(*frame*)
IO sample response using 16-bits addressing

Frame example: 7E 00 14 83 7D 84 23 01 02 14 88 00 80 00 8F 03 ED 00 08 02 4C 00 0C 58

source_address
Source address, first 2 bytes of the `id_data`

rssi
RSSI, immediatly following the `source_address`

options
Options, immediatly following the `rssi`

sample_count
Sample count, immediatly following the `options`

analog_mask
Analog mask, immediatly following the `sample_count`

It is the whole byte of which first and last bit are set to 0 because useless in this context. Second bit (right to left) tells if analog channel 0 (A0) is enabled (bit to 1) or disabled (bit to 0), third bit is for A1 and so on until seventh bit which is for A5

digital_mask
Digital mask, on 2 bytes including the `analog_mask` byte

It is the whole first byte (right to left) and the second one of which all bits except the first one are set to 0 because useless in this context. First bit (right to left) tells if digital channel 0 (D0) is enabled (bit to 1) or disabled (bit to 0), second bit is for D1 and so on until ninth bit which is for D8

`contains_analog`
True if at least one analog channel is enabled, *False* otherwise

`contains_digital`
True if at least one digital channel is enabled, *False* otherwise

`is_analog_enabled`(*pin*)
Tells if a given analog pin is enabled or not

Parameters **pin** (*int*) – analog pin number

Returns *True* if the given analog pin is enabled, *False* otherwise

Return type boolean

`is_digital_enabled`(*pin*)
Tells if a given digital pin is enabled or not

Parameters **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is enabled, *False* otherwise

Return type boolean

`is_digital_on`(*index*, *pin*)
Tells if a given digital pin is on or not in a sample

Parameters

- **index** (*int*) – index of the sample
- **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is on in the sample, *False* otherwise

Return type boolean

`get_analog`(*index*, *pin*)
Gives the analog value of a pin in a sample

Parameters

- **index** (*int*) – index of the sample
- **pin** (*int*) – analog pin number

Returns analog pin value

Return type int

class `hachi.response.`**`AtResponse`**(*frame*)
Response to a `AtRequest`

Frame example: 7E 00 07 88 52 4D 59 00 00 00 7F

`frame_id`
Frame Id, first byte of the `id_data`

`command`
Command, on 2 bytes immediately following the `frame_id`

`status`
Status, immediately following the `command`

`value`
Value, all bytes following the `status` until the end of the `id_data`

class `hachi.response.`**`TxStatusResponse`**(*frame*)

Status response emitted by the module after a `Tx64Request` or a `Tx16Request`

Frame example: 7E 00 03 89 2A 74 D8

`frame_id`
Frame Id, first byte of the `id_data`

`status`
Status, immediately following the `frame_id`

class `hachi.response.`**`ModemStatusResponse`**(*frame*)

Modem status response

Frame example: 7E 00 02 8A 06 6F

`status`
Status, first byte of the `id_data`

class `hachi.response.`**`ZBTxStatusResponse`**(*frame*)

Status response emitted by the module after a `ZBTxRequest` or a `ZBExplicitTxRequest`

Frame example: 7E 00 07 8B 01 7D 84 00 00 01 71

`frame_id`
Frame Id, first byte of the `id_data`

`destination_address`
Destination address, on 2 bytes immediately following the `frame_id`

`retry_count`
Retry count, immediately following the `destination_address`

`delivery_status`
Delivery status, immediately following the `retry_count`

`discovery_status`
Discovery status, immediately following the `delivery_status`

class `hachi.response.`**`ZBRxResponse`**(*frame*)

Response to a `ZBTxRequest`

Frame example: 7E 00 12 90 00 00 13 A2 00 40 52 2B AA 7D 84 01 52 78 44 61 74 61 0D

`frame_id`
Frame Id, first byte of the `id_data`

`source_address_64`
64-bits source address, on 8 bytes immediately following the `frame_id`

`source_address_16`
16-bits source address, on 2 bytes immediately following the `source_address_64`

`options`
Options, immediately following the `source_address_16`

`data`
Data, all bytes following the `options` until the end of the `id_data`

class `hachi.response.`**`ZBExplicitRxResponse`**(*frame*)

Response to a `ZBExplicitTxRequest`

Frame example: 7E 00 18 91 00 13 A2 00 40 52 2B AA 7D 84 E0 E0 22 11 C1 05 02 52 78 44 61 74 61 52

`source_address_64`
64-bits source address, first 8 bytes of the `id_data`

`source_address_16`
16-bits source address, on 2 bytes immediately following the `source_address_64`

`source_endpoint`
Source endpoint, immediately following the `source_address_16`

`destination_endpoint`
Destination endpoint, immediately following the `source_endpoint`

`cluster_id`
Cluster id, on 2 bytes immediately following the `destination_endpoint`

`profile_id`
Profile id, on 2 bytes immediately following the `cluster_id`

`options`
Options, immediately following the `profile_id`

`data`
Data, all bytes following the `options` until the end of the `id_data`

class `hachi.response.`**`ZBIoSampleResponse`**(*frame*)
IO sample response

Frame example: 7E 00 14 92 00 13 A2 00 40 52 2B AA 7D 84 01 01 00 1C 02 00 14 02 25 F5

`source_address_64`
64-bits source address, first 8 bytes of the `id_data`

`source_address_16`
16-bits source address, on 2 bytes immediately following the `source_address_64`

`options`
Options, immediately following the `source_address_16`

`sample_count`
Sample count, immediately following the `options`

`digital_mask`
Digital mask, on 2 bytes, immediately following the `sample_count`

`analog_mask`
Analog mask, immediately following the `digital_mask`

`contains_digital`
True if at least one digital channel is enabled, *False* otherwise

`contains_analog`
True if at least one analog channel is enabled, *False* otherwise

`is_digital_enabled`(*pin*)
Tells if a given digital pin is enabled or not

Parameters **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is enabled, *False* otherwise

Return type boolean

`is_analog_enabled`(*pin*)
Tells if a given analog pin is enabled or not

Parameters **pin** (*int*) – analog pin number

Returns *True* if the given analog pin is enabled, *False* otherwise

Return type boolean

`is_digital_on`(*pin*)
Tells if a given digital pin is on or not

Parameters **pin** (*int*) – digital pin number

Returns *True* if the given digital pin is on, *False* otherwise

Return type boolean

`get_analog`(*pin*)
Gives the analog value of a pin

Parameters **pin** (*int*) – analog pin number

Returns analog pin value

Return type int

class `hachi.response.`**`RemoteAtResponse`**(*frame*)
Response to a `RemoteAtRequest`

Frame example: 7E 00 13 97 55 00 13 A2 00 40 52 2B AA 7D 84 53 4C 00 40 52 2B AA F0

`frame_id`
Frame Id, first byte of the `id_data`

`source_address_64`
64-bits source address, on 8 bytes immediately following the `frame_id`

`source_address_16`
16-bits source address, on 2 bytes immediately following the `source_address_64`

`command`
Command, on two bytes immediately following the `source_address_16`

`status`
Status, immediately following the `command`

`data`
Data, all bytes following the `status` until the end of the `id_data` if any *None* otherwise

3.2.2 Map

`hachi.response.`**`RESPONSE_MAP`**
Mapping from *Response API IDs* to `XBeeResponse`

3.2.3 Utilities

`hachi.response.`**`bitcount`**(*number*)
Count the number of bits to 1 in a number

For example:

```
>>> bitcount(0b10110010)
4
>>> bitcount(0b0100)
1
```

Parameters **number** (*int*) – number on which to count the positive bits

Returns the number of positive bits

Return type int

3.3 Request

3.3.1 Requests

hachi.request.**FRAME_ID_DEFAULT = 1**
Frame id used by default. It is non-zero to trigger a status response

hachi.request.**TRANSMIT_OPTION_DEFAULT = 0**
Transmit option used by default.

class hachi.request.**XBeeRequest**
Base class for all XBee requests

The `XBeeRequest` provides helpers to access to the raw API frame. Unless specified, the type of attributes is `int`.

api_id = None
API ID

Subclasses must implement this and return the corresponding API ID

length
Computed length

id_data
API ID-specific raw data, bytes between the `api_id` and the `checksum`

Subclasses must implement this

Type bytearray

checksum
Computed checksum

frame
Computed frame

class hachi.request.**Tx64Request** (*data*, *destination_address=0*, *options=0*, *frame_id=1*)
Tx Request using 64-bit addressing

frame_id = None
Frame id

destination_address = None
64-bit destination address

options = None
Options

`data` = **None**
Data

Type bytearray or bytes

class `hachi.request.`**`Tx16Request`**(*data*, *destination_address*, *options=0*, *frame_id=1*)
Tx Request using 16-bit addressing

`frame_id` = **None**
Frame id

`destination_address` = **None**
16-bit destination address

`options` = **None**
Options

`data` = **None**
Data

Type bytearray or bytes

class `hachi.request.`**`AtRequest`**(*command*, *parameter=None*, *frame_id=1*)
At Request

`frame_id` = **None**
Frame id

`command` = **None**
Command

Type bytes

`parameter` = **None**
Parameter value

Set to *None* to query the register

Type None or bytearray or bytes

class `hachi.request.`**`AtQueueRequest`**(*command*, *parameter=None*, *frame_id=1*)
At Queue Request

`frame_id` = **None**
Frame id

`command` = **None**
Command

Type bytes

`parameter` = **None**
Parameter value

Set to *None* to query the register

Type None or bytearray or bytes

class `hachi.request.`**`ZBTxRequest`**(*data*, *destination_address_64=0*, *destination_address_16=65534*, *broadcast_radius=0*, *options=0*, *frame_id=1*)
ZB Tx Request

`frame_id` = **None**
Frame id

destination_address_64 = None
64-bit destination address

destination_address_16 = None
16-bit destination address

broadcast_radius = None
Broadcast radius

options = None
Options

data = None
Data

Type bytearray or bytes

class hachi.request.**ZBExplicitTxRequest**(*data*, *destination_address_64*, *source_endpoint*, *destination_endpoint*, *cluster_id*, *profile_id*, *destination_address_16=65534*, *broadcast_radius=0*, *options=0*, *frame_id=1*)
ZB Explicit Tx Request

frame_id = None
Frame id

destination_address_64 = None
64-bit destination address

destination_address_16 = None
16-bit destination address

source_endpoint = None
Source endpoint

destination_endpoint = None
Destination endpoint

cluster_id = None
Cluster id

profile_id = None
Profile id

broadcast_radius = None
Broadcast radius

options = None
Options

data = None
Data

Type bytearray or bytes

class hachi.request.**RemoteAtRequest**(*command*, *destination_address_64*, *parameter=None*, *destination_address_16=65534*, *options=2*, *frame_id=1*)
Remote At Request

frame_id = None
Frame id

destination_address_64 = None
64-bit destination address

`destination_address_16` = **None**
    16-bit destination address

`options` = **None**
    Options

`command` = **None**
    Command

    Type bytes

`parameter` = **None**
    Parameter value

    Set to *None* to query the register

    Type None or bytearray or bytes

3.3.2 Map

`hachi.request.`**`REQUEST_MAP`**
    Mapping from *Request API IDs* to `XBeeRequest`

3.4 Constants

3.4.1 Special bytes

`hachi.const.`**`FRAME_DELIMITER`** = **126**
    Frame delimiter byte

`hachi.const.`**`ESCAPE`** = **125**
    Escape byte

`hachi.const.`**`XON`** = **17**
    XON byte

`hachi.const.`**`XOFF`** = **19**
    XOFF byte

3.4.2 API IDs

Request API IDs

`hachi.const.`**`TX_64_REQUEST`** = **0**
    API ID for `Tx64Request`

`hachi.const.`**`TX_16_REQUEST`** = **1**
    API ID for `Tx16Request`

`hachi.const.`**`AT_REQUEST`** = **8**
    API ID for `AtRequest`

`hachi.const.`**`AT_QUEUE_REQUEST`** = **9**
    API ID for `AtQueueRequest`

`hachi.const.`**`ZB_TX_REQUEST`** = **16**
    API ID for `ZBTxRequest`

`hachi.const.`**`ZB_EXPLICIT_TX_REQUEST`** = 17
    API ID for `ZBExplicitTxRequest`

`hachi.const.`**`REMOTE_AT_REQUEST`** = 23
    API ID for `RemoteAtRequest`

Response API IDs

`hachi.const.`**`RX_64_RESPONSE`** = 128
    API ID for `Rx64Response`

`hachi.const.`**`RX_16_RESPONSE`** = 129
    API ID for `Rx16Response`

`hachi.const.`**`RX_64_IO_RESPONSE`** = 130
    API ID for `Rx64IoSampleResponse`

`hachi.const.`**`RX_16_IO_RESPONSE`** = 131
    API ID for `Rx16IoSampleResponse`

`hachi.const.`**`AT_RESPONSE`** = 136
    API ID for `AtResponse`

`hachi.const.`**`TX_STATUS_RESPONSE`** = 137
    API ID for `TxStatusResponse`

`hachi.const.`**`MODEM_STATUS_RESPONSE`** = 138
    API ID for `ModemStatusResponse`

`hachi.const.`**`ZB_TX_STATUS_RESPONSE`** = 139
    API ID for `ZBTxStatusResponse`

`hachi.const.`**`ZB_RX_RESPONSE`** = 144
    API ID for `ZBRxResponse`

`hachi.const.`**`ZB_EXPLICIT_RX_RESPONSE`** = 145
    API ID for `ZBExplicitRxResponse`

`hachi.const.`**`ZB_IO_SAMPLE_RESPONSE`** = 146
    API ID for `ZBIoSampleResponse`

`hachi.const.`**`REMOTE_AT_RESPONSE`** = 151
    API ID for `RemoteAtResponse`

3.4.3 Special frame ids

`hachi.const.`**`FRAME_ID_NO_RESPONSE`** = 0
    Frame id that disables status responses

3.4.4 Special addresses

`hachi.const.`**`ADDRESS_16_USE_64_BIT_ADDRESSING`** = 65534
    Use 64-bit addressing 16-bit address. Applies to `Tx16Request`

`hachi.const.`**`ADDRESS_16_BROADCAST`** = 65535
    Broadcast 16-bit address

`hachi.const.`**`ADDRESS_64_COORDINATOR`** = 0
    Coordinator 64-bit address

`hachi.const.`**`ADDRESS_64_BROADCAST`** = 65535
  Broadcast 64-bit address

`hachi.const.`**`ADDRESS_64_UNKNOWN`** = 18446744073709551615L
  Unknown 64-bit address

3.4.5 Special broadcast radius

`hachi.const.`**`BROADCAST_RADIUS_MAX_HOPS`** = 0
  Maximum hops

3.4.6 Transmit options

`hachi.const.`**`TRANSMIT_OPTION_DISABLE_ACKNOWLEDGEMENT`** = 1
  Disable acknowledgement. Applies to `Tx64Request` and `Tx16Request`

`hachi.const.`**`TRANSMIT_OPTION_BROADCAST_PACKET`** = 4
  Send packet with broadcast pan id. Applies to `Tx64Request` and `Tx16Request`

`hachi.const.`**`TRANSMIT_OPTION_APPLY_CHANGES`** = 2
  Apply changes. Applies to `RemoteAtRequest`

`hachi.const.`**`TRANSMIT_OPTION_DISABLE_RETRIES_AND_ROUTE_REPAIR`** = 1
  Disable retries and route repair. Applies to `ZBTxRequest` and `ZBExplicitTxRequest`

`hachi.const.`**`TRANSMIT_OPTION_ENABLE_APS_ENCRYPTION`** = 32
  Enable APS encryption. Applies to `ZBTxRequest` and `ZBExplicitTxRequest`

`hachi.const.`**`TRANSMIT_OPTION_USE_EXTENDED_TRANSMISSION_TIMEOUT`** = 64
  Use extended transmission timeout. Applies to `ZBTxRequest` and `ZBExplicitTxRequest`

3.4.7 Receive options

`hachi.const.`**`RECEIVE_OPTION_ADDRESS_BROADCAST`** = 1
  Address broadcast. Applies to `Rx64Response` and `Rx16Response`

`hachi.const.`**`RECEIVE_OPTION_PAN_BROADCAST`** = 2
  PAN broadcast. Applies to `Rx64Response` and `Rx16Response`

`hachi.const.`**`RECEIVE_OPTION_PACKET_ACKNOWLEDGED`** = 1
  Packet acknowledged. Applies to `ZBRxResponse` and `ZBExplicitRxResponse`

`hachi.const.`**`RECEIVE_OPTION_PACKET_BROADCAST`** = 2
  Packet was a broadcast packet. Applies to `ZBRxResponse` and `ZBExplicitRxResponse`

`hachi.const.`**`RECEIVE_OPTION_PACKET_ENCRYPTED_WITH_APS`** = 32
  Packet encrypted with APS encryption. Applies to `ZBRxResponse` and `ZBExplicitRxResponse`

`hachi.const.`**`RECEIVE_OPTION_PACKET_FROM_END_DEVICE`** = 64
  Packet was sent from an end device (if known). Applies to `ZBRxResponse` and `ZBExplicitRxResponse`

3.4.8 Transmit statuses

`hachi.const.`**`STATUS_SUCCESS`** = 0
  Success. Applies to `TxStatusResponse` and `ZBTxStatusResponse`

`hachi.const.`**`STATUS_MAC_ACK_FAILURE`** = 1
MAC ACK failure. Applies to `TxStatusResponse` and `ZBTxStatusResponse`

`hachi.const.`**`STATUS_CCA_FAILURE`** = 2
CCA failure. Applies to `TxStatusResponse` and `ZBTxStatusResponse`

`hachi.const.`**`STATUS_PURGED`** = 3
Purged. Applies to `TxStatusResponse`

`hachi.const.`**`STATUS_INVALID_DESTINATION_ENDPOINT`** = 21
Invalid destination endpoint. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_NETWORK_ACK_FAILURE`** = 33
Network ACK failure. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_NOT_JOINED_TO_NETWORK`** = 34
Not joined to network. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_SELF_ADDRESSED`** = 35
Self-addressed. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_ADDRESS_NOT_FOUND`** = 36
Address not found. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_ROUTE_NOT_FOUND`** = 37
Route not found. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_NEIGHBOR_FAILURE`** = 38
Broadcast source failed to hear a neighbor relay relay the message. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_INVALID_BINDING_TABLE_INDEX`** = 43
Invalid binding table index. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_RESOURCE_ERROR`** = 44
Resource error lack of free buffers, timers, etc. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_ATTEMPTED_BROADCAST_WITH_APS`** = 45
Attempted broadcast with APS transmission. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_ATTEMPTED_UNICAST_APS`** = 46
Attempted unicast with APS transmission but EE=0. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_RESOURCE_ERROR_2`** = 50
Resource error lack of free buffers, timers, etc. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_DATA_PAYLOAD_TOO_LARGE`** = 116
Data payload too large. Applies to `ZBTxStatusResponse`

`hachi.const.`**`STATUS_INDIRECT_MESSAGE_UNREQUESTED`** = 117
Indirect message unrequested. Applies to `ZBTxStatusResponse`

3.4.9 Discovery statuses

`hachi.const.`**`DISCOVERY_STATUS_NO_OVERHEAD`** = 0
No overhead discovery. Applies to `ZBTxStatusResponse`

`hachi.const.`**`DISCOVERY_STATUS_ADDRESS`** = 1
Address discovery. Applies to `ZBTxStatusResponse`

`hachi.const.`**`DISCOVERY_STATUS_ROUTE`** = 2
Route discovery. Applies to `ZBTxStatusResponse`

`hachi.const.`**`DISCOVERY_STATUS_ADDRESS_AND_ROUTE`** = 3
Address and route discovery. Applies to `ZBTxStatusResponse`

`hachi.const.`**`DISCOVERY_STATUS_EXTENDED_TIMEOUT`** = 64
Extended timeout discovery. Applies to `ZBTxStatusResponse`

3.4.10 Command statuses

`hachi.const.`**`COMMAND_STATUS_OK`** = 0
OK. Applies to `AtResponse` and `RemoteAtResponse`

`hachi.const.`**`COMMAND_STATUS_ERROR`** = 1
Error. Applies to `AtResponse` and `RemoteAtResponse`

`hachi.const.`**`COMMAND_STATUS_INVALID_COMMAND`** = 2
Invalid command. Applies to `AtResponse` and `RemoteAtResponse`

`hachi.const.`**`COMMAND_STATUS_INVALID_PARAMETER`** = 3
Invalid parameter. Applies to `AtResponse` and `RemoteAtResponse`

`hachi.const.`**`COMMAND_STATUS_NO_RESPONSE`** = 4
No response. Applies to `RemoteAtResponse`

3.4.11 Modem statuses

`hachi.const.`**`MODEM_STATUS_HARDWARE_RESET`** = 0
Hardware reset

`hachi.const.`**`MODEM_STATUS_WATCHDOG_TIMER_RESET`** = 1
Watchdog timer reset

`hachi.const.`**`MODEM_STATUS_ASSOCIATED`** = 2
Associated

`hachi.const.`**`MODEM_STATUS_DISASSOCIATED`** = 3
Disassociated

`hachi.const.`**`MODEM_STATUS_SYNCHRONIZATION_LOST`** = 4
Synchronization lost

`hachi.const.`**`MODEM_STATUS_COORDINATOR_REALIGNMENT`** = 5
Coordinator realignment

`hachi.const.`**`MODEM_STATUS_COORDINATOR_STARTED`** = 6
Coordinator started

`hachi.const.`**`MODEM_STATUS_NETWORK_SECURITY_KEY_UPDATED`** = 7
Network security key updated

`hachi.const.`**`MODEM_STATUS_VOLTAGE_SUPPLY_LIMIT_EXCEEDED`** = 13
Voltage supply limit exceeded

`hachi.const.`**`MODEM_STATUS_MODEM_CONFIGURATION_CHANGED_WHILE_JOINING`** = 17
Modem configuration changed while join in progress

`hachi.const.`**`MODEM_STATUS_STACK_ERROR_MIN`** = 128
Stack error minimum

3.5 Exceptions

class `hachi.exceptions.`**`HachiError`**
  Base class for all exceptions in hachi

class `hachi.exceptions.`**`Timeout`**
  Timeout

3.6 Serial

3.7 Twisted

CHAPTER

FOUR

HISTORY

4.1 0.3

release date: 2013-07-16

- Python 2.7.3 fixes

4.2 0.2

release date: 2013-07-16

- Python 3 fixes

4.3 0.1

release date: 2013-07-16

- First release

PYTHON MODULE INDEX

h

hachi.const, **??**
hachi.core, **??**
hachi.exceptions, **??**
hachi.request, **??**
hachi.response, **??**
hachi.serial, **??**
hachi.twisted, **??**