Python POD API

Release v1.1.1

Thresa Kelly

Jun 21, 2023

CONTENTS:

CHAPTER

ONE

API_MODULES

1.1 BasicPodProtocol module

class BasicPodProtocol.**POD_Basics**(*port: str | int*, *baudrate: int = 9600*)

Bases: `object`

POD_Basics handles basic communication with a generic POD device, including reading and writing packets and packet interpretation.

_port

Instance-level COM_io object, which handles the COM port

Type
COM_io

_commands

Instance-level POD_Commands object, which stores information about the commands available to this POD device.

Type
POD_Commands

GetDeviceCommands() → dict[int, list[str | tuple[int] | bool]]

Gets the dictionary containing the class instance's available POD commands.

Returns
Dictionary containing the available commands and their information.Formatted as key(command number) : value([command name, number of argument ASCII bytes, number of return bytes, binary flag])

Return type
dict[int, list[str|tuple[int]|bool]]

static GetNumberOfPODDevices() → int

Gets the POD device counter (__numPod).

Returns
Number of POD_Basics instances.

Return type
int

GetPODpacket(*cmd: str | int*, *payload: int | bytes | tuple[int | bytes] | None = None*) → bytes

Builds a POD packet and writes it to a POD device via COM port. If an integer payload is give, the method will convert it into a bytes string of the length expected by the command. If a bytes payload is given, it must be the correct length.

Parameters

- **cmd** (*str | int*) – Command number.
- **payload** (*int | bytes | tuple[int | bytes], optional*) – None when there is no payload. If there is a payload, set to an integer value, bytes string, or tuple. Defaults to None.

Raises

- **Exception** – POD command does not exist.
- **Exception** – POD command requires a payload.

Returns
Bytes string of the POD packet.

Return type
bytes

ReadPODpacket(*validateChecksum: bool = True*) → bytes

Reads a complete POD packet, either in standard or binary format, beginning with STX and ending with ETX. Reads first STX and then starts recursion.

Parameters
validateChecksum (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Returns
Bytes string containing a POD packet beginning with STX and ending with ETX. This may be a standard packet, binary packet, or an unformatted packet (STX+something+ETX).

Return type
bytes

SetBaudrateOfDevice(*baudrate: int*) → bool

If the port is open, it will change the baud rate to the parameter's value.

Parameters
baudrate (*int*) – Baud rate to set for the open serial port.

Returns
True if successful at setting the baud rate, false otherwise.

Return type
bool

TranslatePODpacket(*msg: bytes*) → dict[str, int | bytes]

Determines if the packet is standard or binary, and translates accordingly.

Parameters
msg (*bytes*) – Bytes string containing either a standard or binary packet.

Returns
A dictionary containing the unpacked message in numbers.

Return type
dict[str,int|bytes]

static TranslatePODpacket_Binary(*msg: bytes*) → dict[str, int | bytes]

Unpacks the variable-length binary POD packet and converts the values of the ASCII-encoded bytes into integer values and leaves the binary-encoded bytes as is.

Parameters
msg (*bytes*) – Bytes message containing a variable-length POD packet.

Returns
A dictionary containing the 'Command Number' and 'Binary Packet Length' in integers, and 'Binary Data' in bytes.

Return type
dict[str,int|bytes]

TranslatePODpacket_Standard(*msg: bytes*) → dict[str, int]

Unpacks the standard POD packet and converts the ASCII-encoded bytes values into integer values.

Parameters
msg (*bytes*) – Bytes message containing a standard POD packet

Returns
A dictionary containing the POD packet's 'Command Number' and 'Payload' (if applicable) in integers.

Return type
dict[str,int]

UnpackPODpacket(*msg: bytes*) → dict[str, bytes]

Determines if the packet is standard or binary, and unpacks accordingly.

Parameters
msg (*bytes*) – Bytes string containing either a standard or binary packet

Returns
A dictionary containing the unpacked message in bytes

Return type
dict[str,bytes]

static UnpackPODpacket_Binary(*msg: bytes*) → dict[str, bytes]

Converts a variable-length binary packet into a dictionary containing the command number, binary packet length, and binary data in bytes.

Parameters
msg (*bytes*) – Bytes message containing a variable-length POD packet

Returns
A dictionary containing 'Command Number', 'Binary Packet Length', and 'Binary Data' keys with bytes values.

Return type
dict[str,bytes]

Raises
Exception –

(1) The msg does not have the minimum number of bytes in a standard pod packet,(2) does not begin with STX, (3) does not end with ETX, and (4) does not have an ETX after standard packet.

static UnpackPODpacket_Standard(*msg: bytes*) → dict[str, bytes]

Converts a standard POD packet into a dictionary containing the command number and payload (if applicable) in bytes.

Parameters
msg (*bytes*) – Bytes message containing a standard POD packet.

Returns
A dictionary containing the POD packet's 'Command Number' and 'Payload' (if applicable) in bytes.

Return type
dict[str,bytes]

Raises
Exception –

(1) The msg does not have the minimum number of bytes in a standard pod packet, (2) does not begin with STX, and (3) does not end with ETX.

WritePacket(*cmd: str | int*, *payload: int | bytes | tuple[int | bytes] | None = None*) → bytes

Builds a POD packet and writes it to the POD device.

Parameters

- **cmd** (*str | int*) – Command number.
- **payload** (*int | bytes | tuple[int | bytes], optional*) – None when there is no payload. If there is a payload, set to an integer value, bytes string, or tuple. Defaults to None.

Returns
Bytes string that was written to the POD device.

Return type
bytes

WriteRead(*cmd: str | int*, *payload: int | bytes | tuple[int | bytes] | None = None*, *validateChecksum: bool = True*) → bytes

Writes a command with optional payload to POD device, then reads (once) the device response.

Parameters

- **cmd** (*str | int*) – Command number.
- **payload** (*int | bytes | tuple[int|bytes], optional*) – None when there is no payload. If there is a payload, set to an integer value or a bytes string. Defaults to None.
- **validateChecksum** (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Returns
Bytes string containing a POD packet beginning with STX and ending with ETX. This may be a standard packet, binary packet, or an unformatted packet (STX+something+ETX).

Return type
bytes

_ReadPODpacket_Recursive(*validateChecksum: bool = True*) → bytes

Reads the command number. If the command number ends in ETX, the packet is returned. Next, it checks if the command is allowed. Then, it checks if the command is standard or binary and reads accordingly, then returns the packet.

Parameters
validateChecksum (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Raises
Exception – Cannot read an invalid command.

Returns
Bytes string containing a POD packet beginning with STX and ending with ETX. This may be a standard packet, binary packet, or an unformatted packet (STX+something+ETX).

Return type
bytes

_Read_Binary(*prePacket: bytes*, *validateChecksum: bool = True*) → bytes

Reads the remaining part of the variable-length binary packet. It first reads the standard packet (prePacket+payload+checksum+ETX). Then it determines how long the binary packet is from the payload of the standard POD packet and reads that many bytes. It then reads to ETX to get the checksum+ETX.

Parameters

- **prePacket** (*bytes*) – Bytes string containing the beginning of a POD packet: STX (1 byte) + command number (4 bytes)
- **validateChecksum** (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Raises
Exception – An exception is raised if the checksum is invalid (only if validateChecksum=True).

Returns
Bytes string for a variable-length binary POD packet.

Return type
bytes

_Read_GetCommand(*validateChecksum: bool = True*) → bytes

Reads one byte at a time up to 4 bytes to get the ASCII-encoded bytes command number. For each byte read, it can (1) start the recursion over if an STX is found, (2) returns if ETX is found, or (3) continue building the command number.

Parameters
validateChecksum (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Returns
4 byte long string containing the ASCII-encoded command number.

Return type
bytes

_Read_Standard(*prePacket: bytes*, *validateChecksum: bool = True*) → bytes

Reads the payload, checksum, and ETX. Then it builds the complete standard POD packet in bytes.

Parameters

- **prePacket** (*bytes*) – Bytes string containing the beginning of a POD packet: STX (1 byte) + command number (4 bytes).
- **validateChecksum** (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Raises
Exception – An exception is raised if the checksum is invalid (only if validateChecksum=True).

Returns
Bytes string for a complete standard POD packet.

Return type
bytes

_Read_ToETX(*validateChecksum: bool = True*) → bytes

Reads one byte at a time until an ETX is found. It will restart the recursive read if an STX is found anywhere.

Parameters
validateChecksum (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Returns
Bytes string ending with ETX.

Return type
bytes

static _ValidateChecksum(*msg: bytes*) → bool

Validates the checksum of a given POD packet. The checksum is valid if the calculated checksum from the data matches the checksum written in the packet.

Parameters
msg (*bytes*) – Bytes message containing a POD packet: STX (1 bytes) + data (? bytes) + checksum (2 bytes) + ETX (1 byte).

Returns
True if the checksum is correct, false otherwise.

Return type
bool

Raises
Exception – msg does not begin with STX or end with ETX.

__MINBINARYLENGTH: int = 15

Class-level integer representing the minimum length of a binary POD command packet. Format is STX (1 byte) + command number (4 bytes) + length of binary (4 bytes) + checksum (2 bytes) + ETX (1 bytes) + binary (LENGTH bytes) + checksum (2 bytes) + ETX (1 bytes)

__MINSTANDARDLENGTH: int = 8

Class-level integer representing the minimum length of a standard POD command packet. Format is STX (1 byte) + command number (4 bytes) + optional packet (? bytes) + checksum (2 bytes) + ETX (1 bytes)

__numPod: int = 0

Class-level integer counting the number of POD_Basics instances. Maintained by __init__ and __del__.

1.2 GetUserInput module

class GetUserInput.**UserInput**

Bases: object

UserInput contains several methods for getting user input for POD device setup.

static AskForFloat(*prompt: str*) → float

Asks user for float type input.

Parameters
prompt (*str*) – Statement requesting input from the user.

Returns
Float type input from user.

Return type
float

static AskForFloatInList(*prompt: str*, *goodInputs: list*, *badInputMessage: str | None = None*) → float

Asks the user for a float that exists in the list of valid options.

Parameters

- **prompt** (*str*) – Statement requesting input from the user.
- **goodInputs** (*list*) – List of valid input options.
- **badInputMessage** (*str | None, optional*) – Error message to be printed if invalid input is given. Defaults to None.

Returns
User's choice from the options list as a float.

Return type
float

static AskForFloatInRange(*prompt: str*, *minimum: float*, *maximum: float*, *thisIs: str = 'Input'*, *unit: str = ''*) → float

Asks the user for an float value that falls in a given range.

Parameters

- **prompt** (*str*) – Statement requesting input from the user.
- **minimum** (*float*) – Minimum value of range.
- **maximum** (*float*) – Maximum value of range.
- **thisIs** (*str, optional*) – Description of the input/what is being asked for. Used when printing the error message. Defaults to 'Input'.
- **unit** (*str, optional*) – Unit of the requested value. Use when printing the error message. Defaults to ''.

Returns
Float value given by the user that falls in the given range.

Return type
float

static AskForInput(*prompt: str*, *append: str = ': '*) → str

Asks user for input given a prompt. Will append a colon ':' to the end of prompt by default

Parameters

- **prompt** (*str*) – Statement requesting input from the user
- **append** (*str, optional*) – Appended to the end of the prompt. Defaults to ': '.

Returns
String of the user input

Return type
str

static **AskForInt**(*prompt: str*) → int

Asks user for int type input.

Parameters
prompt (*str*) – Statement requesting input from the user.

Returns
integer type input from user.

Return type
int

static **AskForIntInList**(*prompt: str*, *goodInputs: list*, *badInputMessage: str | None = None*) → int

Asks the user for an integer that exists in the list of valid options.

Parameters

- **prompt** (*str*) – Statement requesting input from the user
- **goodInputs** (*list*) – List of valid input options.
- **badInputMessage** (*str | None, optional*) – Error message to be printed if invalid input is given. Defaults to None.

Returns
User's choice from the options list as an integer.

Return type
int

static **AskForIntInRange**(*prompt: str*, *minimum: int*, *maximum: int*, *thisIs: str = 'Input'*, *unit: str = ''*) → int

Asks the user for an integer value that falls in a given range.

Parameters

- **prompt** (*str*) – Statement requesting input from the user.
- **minimum** (*int*) – Minimum value of range.
- **maximum** (*int*) – Maximum value of range.
- **thisIs** (*str, optional*) – Description of the input/what is being asked for. Used when printing the error message. Defaults to 'Input'.
- **unit** (*str, optional*) – Unit of the requested value. Use when printing the error message. Defaults to ''.

Returns
Integer value given by the user that falls in the given range.

Return type
int

static **AskForStrInList**(*prompt: str*, *goodInputs: list*, *badInputMessage: str | None = None*) → str

Asks the user for a string that exists in the list of valid options.

Parameters

- **prompt** (*str*) – Statement requesting input from the user.
- **goodInputs** (*list*) – List of valid input options
- **badInputMessage** (*str | None, optional*) – Error message to be printed if invalid input is given. Defaults to None.

Returns
User's choice from the options list as a string.

Return type
str

static AskForType(*typecast: function*, *prompt: str*) → int | float | str

Ask user for input of a specific data type. If invalid input is given, an error message will print and the user will be prompted again.

Parameters

- **typecast** (*function*) – Datatype to cast the user input (ex. _CastInt, _CastFloat, _CastStr)
- **prompt** (*str*) – Statement requesting input from the user

Returns
Input from user as the requested type.

Return type
int|float|str

static AskForTypeInList(*typecast: function*, *prompt: str*, *goodInputs: list*, *badInputMessage: str | None = None*) → int | float | str

Asks the user for a value of a given type that exists in the list of valid options. If invalid input is given, an error message will print and the user will be prompted again.

Parameters

- **typecast** (*function*) – Datatype to cast the user input (ex. _CastInt, _CastFloat, _CastStr).
- **prompt** (*str*) – Statement requesting input from the user.
- **goodInputs** (*list*) – List of valid input options.
- **badInputMessage** (*str | None, optional*) – Error message to be printed if invalid input is given. Defaults to None.

Returns
User's choice from the goodInputs list as the given datatype.

Return type
int|float|str

static AskForTypeInRange(*typecast: function*, *prompt: str*, *minimum: int | float*, *maximum: int | float*, *thisIs: str = 'Input'*, *unit: str = ''*) → int | float

Asks user for a numerical value that falls between two numbers. If invalid input is given, an error message will print and the user will be prompted again.

Parameters

- **typecast** (*function*) – Datatype to cast the user input (ex. _CastInt, _CastFloat, _CastStr).
- **prompt** (*str*) – Statement requesting input from the user.
- **minimum** (*int | float*) – Minimum value of range.
- **maximum** (*int | float*) – Maximum value of range.
- **thisIs** (*str, optional*) – Description of the input/what is being asked for. Used when printing the error message. Defaults to 'Input'.

- **unit** (*str, optional*) – Unit of the requested value. Use when printing the error message. Defaults to ‘’.

Returns
Numerical value given by the user that falls in the given range.

Return type
int|float

static AskYN(*question: str*, *append: str = ' (y/n): '*) → bool

Asks the user a yes or no question. If invalid input is given, an error message will print and the user will be prompted again.

Parameters

- **question** (*str*) – Statement requesting input from the user.
- **append** (*str, optional*) – Appended to the end of the question. Defaults to ‘ (y/n): ‘.

Returns
True for yes, false for no.

Return type
bool

static CastFloat(*value*) → float

Casts the argument as an float.

Parameters
value – Value to type casted.

Returns
Value type casted as a float.

Return type
float

static CastInt(*value*) → int

Casts the argument as an integer.

Parameters
value – Value to type casted.

Returns
Value type casted as an integer.

Return type
int

static CastStr(*value*) → str

Casts the argument as an string.

Parameters
value – Value to type casted.

Returns
Value type casted as a string.

Return type
str

1.3 PodCommands module

class PodCommands.**POD_Commands**

Bases: object

POD_Commands manages a dictionary containing available commands for a POD device.

__commands

Dictionary containing the available commands for a POD device. Each entry is formatted as { key(command number) : value([command name, number of argument ASCII bytes, number of return bytes, binary flag) }.

Type
dict[int,list[str|tuple[int]|bool]]

AddCommand(*commandNumber: int*, *commandName: str*, *argumentBytes: tuple[int]*, *returnBytes: tuple[int]*, *isBinary: bool*) → bool

Adds a command entry to the current commands dictionary (__commands) if the command does not exist.

Parameters

- **commandNumber** (*int*) – Integer of the command number.
- **commandName** (*str*) – String of the command's name.
- **argumentBytes** (*tuple[int]*) – Integer of the number of bytes in the argument.
- **returnBytes** (*tuple[int]*) – Integer of the number of bytes in the return.
- **isBinary** (*bool*) – Boolean flag to mark if the command is binary (True) or standard (False).

Returns
True if the command was successfully added, False if the command could not be added because it already exists.

Return type
bool

ArgumentHexChar(*cmd: int | str*) → tuple[int] | None

Gets the tuple for the number of hex characters in the argument for a given command.

Parameters
cmd (*int | str*) – integer command number or string command name.

Returns
Tuple representing the number of bytes in the argument for cmd. If the command could not be found, return None.

Return type
tuple[int]|None

CommandNumberFromName(*name: str*) → int | None

Gets the command number from the command dictionary using the command's name.

Parameters
name (*str*) – string of the command's name.

Returns
Integer representing the command number. If the command could not be found, return None.

Return type
int|None

DoesCommandExist(*cmd: int | str*) → bool

Checks if a command exists in the __commands dictionary.

Parameters
cmd (*int | str*) – integer command number or string command name.

Returns
True if the command exists, false otherwise.

Return type
bool

static GetBasicCommands() → dict[int, list[str | tuple[int] | bool]]

Creates a dictionary containing the basic POD command set (0,1,2,3,4,5,6,7,8,9,10,11,12).

Returns
Dictionary containing the available commands for this POD device. Each entry is formatted as { key(command number) : value([command name, number of argument ASCII bytes, number of return bytes, binary flag) }.

Return type
dict[int,list[str|tuple[int]|bool]]

GetCommands() → dict[int, list[str | tuple[int] | bool]]

Gets the contents of the current command dictionary (__commands).

Returns
Dictionary containing the available commands for a POD device. Each entry is formatted as { key(command number) : value([command name, number of argument ASCII bytes, number of return bytes, binary flag) }.

Return type
dict[int, list[str|tuple[int]|bool]]

IsCommandBinary(*cmd: int | str*) → bool | None

Gets the binary flag for a given command.

Parameters
cmd (*int | str*) – integer command number or string command name.

Returns
Boolean flag that is True if the command is binary and False if standard. If the command could not be found, return None.

Return type
bool|None

static NoValue() → int

Gets value of __NOVALUE.

Returns
Value of __NOVALUE.

Return type
int

RemoveCommand(*cmd: int | str*) → bool

Removes the entry for a given command in __commands dictionary.

Parameters
cmd (*int | str*) – integer command number or string command name.

Returns
True if the command was successfully removed, False if the command does not exist.

Return type
bool

RestoreBasicCommands() → None

Sets the current commands (__commands) to the basic POD command set.

ReturnHexChar(*cmd: int | str*) → tuple[int] | None

Gets the tuple for the number of hex characters in the return for a given command.

Parameters
cmd (*int | str*) – integer command number or string command name.

Returns
Tuple representing the number of hex characters in the return for cmd. If the command could not be found, return None.

Return type
tuple[int]|None

static U16() → int

Gets value of __U16.

Returns
Value of __U16.

Return type
int

static U8() → int

Gets value of __U8.

Returns
Value of __U8.

Return type
int

__ARGUMENTS: int = 1

Class-level integer representing the index key for the number of bytes in an argument for __commands list values.

__BINARY: int = 3

Class-level integer representing the index key for the binary flag for __commands list values.

__NAME: int = 0

Class-level integer representing the index key for the command name for __commands list values.

__NOVALUE: int = -1

Class-level integer used to mark when a list item in __commands means 'no value' or is undefined.

__RETURNS: int = 2

Class-level integer representing the index key for the number of bytes in the return for __commands list values.

__U16: int = 4

Class-level integer representing the number of hexadecimal characters for an unsigned 16-bit value.

__U8: int = 2

Class-level integer representing the number of hexadecimal characters for an unsigned 8-bit value.

1.4 PodDevice_8206HR module

class PodDevice_8206HR.**POD_8206HR**(*port: str | int*, *preampGain: int*, *baudrate: int = 9600*)

Bases: *POD_Basics*

POD_8206HR handles communication using an 8206HR POD device.

_preampGain

Instance-level integer (10 or 100) preamplifier gain.

Type
int

TranslatePODpacket(*msg: bytes*) → dict[str, int | dict[str, int]]

Overwrites the parent's method. Determines if the packet is standard or binary, and translates accordingly. Adds a check for the 'GET TTL PORT' command.

Parameters
msg (*bytes*) – Bytes string containing either a standard or binary packet.

Returns
A dictionary containing the unpacked message in numbers.

Return type
dict[str,int|dict[str,int]]

TranslatePODpacket_Binary(*msg: bytes*) → dict[str, int | float | dict[str, int]]

Overwrites the parent's method. Unpacks the binary4 POD packet and converts the values of the ASCII-encoded bytes into integer values and the values of binary-encoded bytes into integers. Channel values are given in Volts.

Parameters
msg (*bytes*) – Bytes string containing a complete binary4 Pod packet: STX (1 byte) + command (4 bytes) + packet number (1 bytes) + TTL (1 byte) + ch0 (2 bytes) + ch1 (2 bytes) + ch2 (2 bytes) + checksum (2 bytes) + ETX (1 byte).

Returns
A dictionary containing 'Command Number', 'Packet #', 'TTL', 'Ch0', 'Ch1', and 'Ch2' as numbers.

Return type
dict[str,int|float|dict[str,int]]

static UnpackPODpacket_Binary(*msg: bytes*) → dict[str, bytes]

Overwrites the parent's method. Separates the components of a binary4 packet into a dictionary.

Parameters
msg (*bytes*) – Bytes string containing a complete binary4 Pod packet: STX (1 byte) + command (4 bytes) + packet number (1 bytes) + TTL (1 byte) + ch0 (2 bytes) + ch1 (2 bytes) + ch2 (2 bytes) + checksum (2 bytes) + ETX (1 byte).

Raises
Exception –

(1) the packet does not have the minimum number of bytes, (2) does not begin with STX, or (3) does not end with ETX.

Returns
A dictionary containing 'Command Number', 'Packet #', 'TTL', 'Ch0', 'Ch1', and 'Ch2' in bytes.

Return type
dict[str,bytes]

_BinaryBytesToVoltage(*value: bytes*) → float

Converts a binary bytes value read from POD device and converts it to the real voltage value at the preamplifier input.

Parameters
value (*bytes*) – Bytes string containing voltage measurement.

Returns
A number containing the voltage in Volts [V].

Return type
float

_Read_Binary(*prePacket: bytes*, *validateChecksum: bool = True*) → bytes

After receiving the prePacket, it reads the 8 bytes(TTL+channels) and then reads to ETX (checksum+ETX).

Parameters

- **prePacket** (*bytes*) – Bytes string containing the beginning of a POD packet: STX (1 byte) + command number (4 bytes).
- **validateChecksum** (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Raises
Exception – Bad checksum for binary POD packet read.

Returns
Byte string for a binary4 POD packet.

Return type
bytes

static _TranslateTTLbyte_ASCII(*ttlByte: bytes*) → dict[str, int]

Separates the bits of each TTL (0-3) from a ASCII encoded byte.

Parameters
ttlByte (*bytes*) – One byte string for the TTL (ASCII encoded).

Returns
Dictionary of the TTLs. Values are 1 when input, 0 when output.

Return type
dict[str,int]

static _TranslateTTLbyte_Binary(*ttlByte: bytes*) → dict[str, int]

Separates the bits of each TTL (0-3) from a binary encoded byte.

Parameters
ttlByte (*bytes*) – One byte string for the TTL (binary encoded).

Returns
Dictionary of the TTLs. Values are 1 when input, 0 when output.

Return type
dict[str,int]

__B4BINARYLENGTH: int = 8

Class-level integer representing the number of binary bytes for a Binary 4 packet.

__B4LENGTH: int = 16

Class-level integer representing the number of bytes for a Binary 4 packet.

1.5 PodDevice_8401HR module

class PodDevice_8401HR.**POD_8401HR**(*port: str | int*, *ssGain: dict[str, int | None] = {'A': None, 'B': None, 'C': None, 'D': None}*, *preampGain: dict[str, int | None] = {'A': None, 'B': None, 'C': None, 'D': None}*, *baudrate: int = 9600*)

Bases: *POD_Basics*

POD_8401HR handles communication using an 8401-HR POD device.

_ssGain

Instance-level dictionary storing the second-stage gain for all four channels.

Type
dict[str,int|None]

_preampGain

Instance-level dictionary storing the pramplifier gain for all four channels.

Type
dict[str,int|None]

static CalculateBiasDAC_GetDACValue(*vout: int | float*) → int

Calculates the DAC value given the output voltage. Used for 'GET/SET BIAS' commands.

Parameters
vout (*int | float*) – Output voltage (+/- 2.048 V).

Returns
Integer of the DAC value (16 bit 2's complement).

Return type
int

static CalculateBiasDAC_GetVout(*value: int*) → float

Calculates the output voltage given the DAC value. Used for 'GET/SET BIAS' commands.

Parameters
value (*int*) – DAC value (16 bit 2's complement).

Returns
Float of the output bias voltage [V].

Return type
float

static GetChannelMapForPreampDevice(*preampName: str*) → dict[str, str] | None

Get the channel mapping (channel labels for A,B,C,D) for a given device.

Parameters
preampName (*str*) – String for the device/sensor name.

Returns
Dictionary with keys A,B,C,D with values of the channel names. Returns None if the device name does not exist.

Return type
dict[str,str]|None

static GetSSConfigBitmask_int(*gain: int*, *highpass: float*) → int

Gets a bitmask, represented by an unsigned integer, used for 'SET SS CONFIG' command.

Parameters

- **gain** (*int*) – 1 for 1x gain. else for 5x gain.
- **highpass** (*float*) – 0 for DC highpass, else for 0.5Hz highpass.

Returns
Integer representing a bitmask.

Return type
int

static GetSupportedPreampDevices() → list[str]

Gets a list of device/sensor names used for channel mapping.

Returns
List of string names of all supported sensors.

Return type
list[str]

static GetTTLbitmask_Int(*ext0: bool = 0*, *ext1: bool = 0*, *ttl4: bool = 0*, *ttl3: bool = 0*, *ttl2: bool = 0*, *ttl1: bool = 0*) → int

Builds an integer, which represents a binary mask, that can be used for TTL command arguments.

Parameters

- **ext0** (*bool, optional*) – boolean bit for ext0. Defaults to 0.
- **ext1** (*bool, optional*) – boolean bit for ext1. Defaults to 0.
- **ttl4** (*bool, optional*) – boolean bit for ttl4. Defaults to 0.
- **ttl3** (*bool, optional*) – boolean bit for ttl3. Defaults to 0.
- **ttl2** (*bool, optional*) – boolean bit for ttl2. Defaults to 0.
- **ttl1** (*bool, optional*) – boolean bit for ttl1. Defaults to 0.

Returns
Integer number to be used as a bit mask.

Return type
int

static IsPreampDeviceSupported(*name: str*) → bool

Checks if the argument exists in channel map for all preamp sensors.

Parameters
name (*str*) – name of the device

Returns
True if the name exists in __CHANNELMAPALL, false otherwise.

Return type
bool

TranslatePODpacket(*msg: bytes*) → dict[str, int | dict[str, int]]

Overwrites the parent's method. Unpacks the binary5 POD packet and converts the values of the ASCII-encoded bytes into integer values and the values of binary-encoded bytes into integers. The channels and analogs are converted to volts (V).

Parameters
msg (*bytes*) – Bytes string containing a complete binary 5 Pod packet: STX (1 byte) + command (4) + packet number (1) + status (1) + channels (9) + analog inputs (12) + checksum (2) + ETX (1).

Returns
A dictionary containing 'Command Number', 'Packet #', 'Status', 'D', 'C', 'B', 'A', 'Analog EXT0', 'Analog EXT1', 'Analog TTL1', 'Analog TTL2', 'Analog TTL3', 'Analog TTL4', as numbers.

Return type
dict[str,int|dict[str,int]]

TranslatePODpacket_Binary(*msg: bytes*) → dict[str, int | float]

Unpacks the variable-length binary POD packet and converts the values of the ASCII-encoded bytes into integer values and leaves the binary-encoded bytes as is.

Parameters
msg (*bytes*) – Bytes message containing a variable-length POD packet.

Returns
A dictionary containing the 'Command Number' and 'Binary Packet Length' in integers, and 'Binary Data' in bytes.

Return type
dict[str,int|bytes]

static UnpackPODpacket_Binary(*msg: bytes*) → dict[str, bytes]

Overwrites the parent's method. Separates the components of a binary5 packet into a dictionary.

Parameters
msg (*bytes*) – Bytes string containing a complete binary5 Pod packet: STX (1 byte) + command (4) + packet number (1) + status (1) + channels (9) + analog inputs (12) + checksum (2) + ETX (1)

Raises
Exception –

(1) The packet does not have the minimum number of bytes, (2) does not begin with STX, or (3) does not end with ETX.

Returns
A dictionary containing 'Command Number', 'Packet #', 'Status', 'Channels', 'Analog EXT0', 'Analog EXT1', 'Analog TTL1', 'Analog TTL2', 'Analog TTL3', 'Analog TTL4', in bytes.

Return type
dict[str,bytes]

_Read_Binary(*prePacket: bytes*, *validateChecksum: bool = True*)

After receiving the prePacket, it reads the 23 bytes (binary data) and then reads to ETX.

Parameters

- **prePacket** (*bytes*) – Bytes string containing the beginning of a POD packet: STX (1 byte) + command number (4 bytes).
- **validateChecksum** (*bool, optional*) – Set to True to validate the checksum. Set to False to skip validation. Defaults to True.

Raises

Exception – Bad checksum for binary POD packet read.

static _TranslateTTLByte(*ttlByte: bytes*) → dict[str, int]

Converts the TTL bytes argument into a dictionary of integer TTL values.

Parameters

ttlByte (*bytes*) – One byte containing the TTL bitmask.

Returns

Dictinoary with TTL name keys and integer TTL values.

Return type

dict[str,int]

static _Voltage_PrimaryChannels(*value: int*, *ssGain: int | None = None*, *PreampGain: int | None = None*) → float

Converts a value to a voltage for a primary channel.

Parameters

- **value** (*int*) – Value to be converted to voltage.
- **ssGain** (*int | None, optional*) – Second stage gain. Defaults to None.
- **PreampGain** (*int | None, optional*) – Preamplifier gain. Defaults to None.

Returns

Number of the voltage in volts [V]. Returns value if no gain is given (no-connect).

Return type

float

static _Voltage_PrimaryChannels_Biosensor(*value: int*, *ssGain: int*) → float

Converts a value to a voltage for a biosensor primary channel.

Parameters

- **value** (*int*) – Value to be converted to voltage.
- **ssGain** (*int*) – Second stage gain.

Returns

Number of the voltage in volts [V].

Return type

float

static _Voltage_PrimaryChannels_EEGEMG(*value: int*, *ssGain: int*, *PreampGain: int*) → float

Converts a value to a voltage for an EEG/EMG primary channel.

Parameters

- **value** (*int*) – Value to be converted to voltage.

- **ssGain** (*int*) – Second stage gain.
- **PreampGain** (*int*) – Preamplifier gain.

Returns
Number of the voltage in volts [V].

Return type
float

static _Voltage_SecondaryChannels(*value: int*) → float

Converts a value to a voltage for a secondary channel.

Parameters
value (*int*) – Value to be converted to voltage.

Returns
Number of the voltage in volts [V].

Return type
float

__B5BINARYLENGTH: int = 23

Class-level integer representing the number of binary bytes for a Binary 5 packet.

__B5LENGTH: int = 31

Class-level integer representing the number of bytes for a Binary 5 packet.

__CHANNELMAPALL: dict[str, dict[str, str]] = {'8406-2BIO': {'A': 'Bio1', 'B': 'Bio2', 'C': 'NC', 'D': 'NC'}, '8406-BIO': {'A': 'Bio', 'B': 'NC', 'C': 'NC', 'D': 'NC'}, '8406-EEG2BIO': {'A': 'Bio1', 'B': 'EEG1', 'C': 'EMG', 'D': 'Bio2'}, '8406-SE': {'A': 'Bio', 'B': 'EEG1', 'C': 'EMG', 'D': 'EEG2'}, '8406-SE3': {'A': 'Bio', 'B': 'EEG1', 'C': 'EEG3', 'D': 'EEG2'}, '8406-SE31M': {'A': 'EMG', 'B': 'EEG1', 'C': 'EEG3', 'D': 'EEG2'}, '8406-SE4': {'A': 'EEG4', 'B': 'EEG1', 'C': 'EEG3', 'D': 'EEG2'}, '8406-SL': {'A': 'Bio', 'B': 'EEG1', 'C': 'EMG', 'D': 'EEG2'}, '8407-SE': {'A': 'Bio', 'B': 'EEG1', 'C': 'EMG', 'D': 'EEG2'}, '8407-SE-2BIO': {'A': 'Bio1', 'B': 'Bio2', 'C': 'EMG', 'D': 'EEG2'}, '8407-SE3': {'A': 'Bio', 'B': 'EEG1', 'C': 'EEG3', 'D': 'EEG2'}, '8407-SE31M': {'A': 'EEG3', 'B': 'EEG1', 'C': 'EMG', 'D': 'EEG2'}, '8407-SE4': {'A': 'EEG4', 'B': 'EEG1', 'C': 'EEG3', 'D': 'EEG2'}, '8407-SL': {'A': 'Bio', 'B': 'EEG1', 'C': 'EMG', 'D': 'EEG2'}, '8407-SL-2BIO': {'A': 'Bio1', 'B': 'Bio2', 'C': 'EMG', 'D': 'EEG2'}}

Class-level dictionary containing the channel map for all preamplifier devices.

1.6 PodPacketHandling module

class PodPacketHandling.**POD_Packets**

Bases: object

POD_Packets is a collection of methods for creating and interpreting POD packets.

static ASCIIbytesToInt_Split(*msg: bytes*, *keepTopBits: int*, *cutBottomBits: int*) → int

Converts a specific bit range in an ASCII-encoded bytes object to an integer.

Parameters

- **msg** (*bytes*) – Bytes message holding binary information to be converted into an integer.
- **keepTopBits** (*int*) – Integer position of the msb of desired bit range.

- **cutBottomBits** (*int*) – Integer number of lsb to remove.

Returns
Integer result from the ASCII-encoded bytes message in a given bit range.

Return type
int

static AsciiBytesToInt(*msg_b: bytes*, *signed: bool = False*) → int

Converts a ASCII-encoded bytes message into an integer. It does this using a base-16 conversion. If the message is signed and the msb is '1', the integer will be converted to it's negative 2's complement.

Parameters

- **msg_b** (*bytes*) – Bytes message to be converted to an integer. The bytes must be base-16 or the conversion will fail.
- **signed** (*bool, optional*) – True if the message is signed, false if unsigned. Defaults to False.

Returns
Integer result from the ASCII-encoded byte conversion.

Return type
int

static BinaryBytesToInt(*msg: bytes*, *byteorder: str = 'big'*, *signed: bool = False*) → int

Converts binary-encoded bytes into an integer.

Parameters

- **msg** (*bytes*) – Bytes message holding binary information to be converted into an integer.
- **byteorder** (*str, optional*) – Ordering of bytes. 'big' for big endian and 'little' for little endian. Defaults to 'big'.
- **signed** (*bool, optional*) – Boolean flag to mark if the msg is signed (True) or unsigned (False). Defaults to False.

Returns
Integer result from the binary-encoded bytes message.

Return type
int

static BinaryBytesToInt_Split(*msg: bytes*, *keepTopBits: int*, *cutBottomBits: int*, *byteorder: str = 'big'*, *signed: bool = False*) → int

Converts a specific bit range in a binary-encoded bytes object to an integer.

Parameters

- **msg** (*bytes*) – Bytes message holding binary information to be converted into an integer.
- **keepTopBits** (*int*) – Integer position of the msb of desired bit range.
- **cutBottomBits** (*int*) – Integer number of lsb to remove.
- **byteorder** (*str, optional*) – Ordering of bytes. 'big' for big endian and 'little' for little endian. Defaults to 'big'.
- **signed** (*bool, optional*) – Boolean flag to mark if the msg is signed (True) or unsigned (False). Defaults to False.

Returns
Integer result from the binary-encoded bytes message in a given bit range.

Return type
int

static BuildPODpacket_Standard(*commandNumber: int*, *payload: bytes | None = None*) → bytes

Builds a standard POD packet as bytes: STX (1 byte) + command number (4 bytes) + optional packet (? bytes) + checksum (2 bytes)+ ETX (1 bytes).

Parameters

- **commandNumber** (*int*) – Integer representing the command number. This will be converted into a 4 byte long ASCII-encoded bytes string.
- **payload** (*bytes | None, optional*) – bytes string containing the payload. Defaults to None.

Returns
Bytes string of a complete standard POD packet.

Return type
bytes

static Checksum(*bytesIn: bytes*) → bytes

Calculates the checksum of a given bytes message. This is achieved by summing each byte in the message, inverting, and taking the last byte.

Parameters
bytesIn (*bytes*) – Bytes message containing POD packet data.

Returns
Two ASCII-encoded bytes containing the checksum for bytesIn.

Return type
bytes

static ETX() → bytes

Get end-of-transmission (ETX) character in bytes. ETX marks the end byte of a POD Packet.

Returns
Bytes for ETX(0x03).

Return type
bytes

static IntToAsciiBytes(*value: int*, *numChars: int*) → bytes

Converts an integer value into ASCII-encoded bytes.

First, it converts the integer value into a usable uppercase hexadecimal string. Then it converts the ASCII code for each character into bytes. Lastly, it ensures that the final message is the desired length.

Example: if value=2 and numBytes=4, the returned ASCII will show b'0002', which is '0x30 0x30 0x30 0x32' in bytes. Uses the 2's complement if the val is negative.

Parameters

- **value** (*int*) – Integer value to be converted into ASCII-encoded bytes.
- **numChars** (*int*) – Number characters to be the length of the ASCII-encoded message.

Returns
Bytes that are ASCII-encoded conversions of the value parameter.

Return type
bytes

static PayloadToBytes(*payload: int | bytes | tuple[int | bytes]*, *argSizes: tuple[int]*) → bytes

Converts a payload into a bytes string.

Parameters

- **payload** (*int | bytes | tuple[int | bytes]*) – Integer, bytes, or tuple containing the payload.
- **argSizes** (*tuple[int]*) – Tuple of the argument sizes.

Raises

- **Exception** – Payload requires multiple arguments, use a tuple.
- **Exception** – Payload is the wrong size.
- **Exception** – Payload has an incorrect number of items.
- **Exception** – Payload has invalid values.
- **Exception** – Payload is an invalid type.

Returns
Bytes string of the payload.

Return type
bytes

static STX() → bytes

Get start-of-transmission (STX) character in bytes. STX marks the starting byte of a POD Packet.

Returns
Bytes for STX (0x02).

Return type
bytes

static TwosComplement(*val: int*, *nbits: int*) → int

Gets the 2's complement of the argument value.

Parameters

- **val** (*int*) – Value to be complemented.
- **nbits** (*int*) – Number of bits in the value.

Returns
Integer of the 2's complement for the val.

Return type
int

1.7 SerialCommunication module

class SerialCommunication.**COM_io**(*port: str | int*, *baudrate: int = 9600*)

Bases: object

COM_io handles serial communication (read/write) using COM ports.

__serialInst

Instance-level serial COM port.

Type
Serial

CloseSerialPort() → None

Closes the instance serial port if it is open.

static GetCOMportsList() → list[str]

Finds all the available COM ports on the user's computer and appends them to an accessible list.

Returns
List containing the names of available COM ports.

Return type
list[str]

GetPortName() → str | None

Gets the name of the open port.

Returns
If the serial port is open, it will return a string of the port's name. If the port is closed, it will return None.

Return type
str|None

IsSerialClosed() → bool

Returns False if the serial instance port is open, True otherwise.

Returns
True if the COM port is closed, False otherwise.

Return type
bool

IsSerialOpen() → bool

Returns True if the serial instance port is open, false otherwise.

Returns
True if the COM port is open, False otherwise.

Return type
bool

OpenSerialPort(*port: str | int*, *baudrate: int = 9600*) → None

First, it closes the serial port if it is open. Then, it opens a serial port with a set baud rate.

Parameters

- **port** (*str | int*) – String of the serial port to be opened.
- **baudrate** (*int, optional*) – Integer baud rate of the opened serial port. Defaults to 9600.

Raises
Exception – Port does not exist.

Read(*numBytes: int*) → bytes | None

Reads a specified number of bytes from the open serial port.

Parameters
numBytes (*int*) – Integer number of bytes to read.

Returns
If the serial port is open, it will return a set number of read bytes. If it is closed, it will return None.

Return type
bytes|None

ReadLine() → bytes | None

Reads until a new line is read from the open serial port.

Returns
If the serial port is open, it will return a complete read line. If closed, it will return None.

Return type
bytes|None

ReadUntil(*eol: bytes*) → bytes | None

Reads until a set character from the open serial port.

Parameters
eol (*bytes*) – end-of-line character.

Returns
If the serial port is open, it will return a read line ending in eol. If closed, it will return None.

Return type
bytes|None

SetBaudrate(*baudrate: int*) → bool

Write(*message: bytes*) → None

Write a set message to the open serial port.

Parameters
message (*bytes*) – byte string containing the message to write.

__BuildPortName(*port: str | int*) → str

Converts the port parameter into the "COM"+<number> format.

Parameters
port (*str | int*) – Name of a COM port. Can be an integer or string.

Returns
Name of the COM port.

Return type
str

1.8 Setup_8206HR module

class Setup_8206HR.**Setup_8206HR**

Bases: *Setup_Interface*

Setup_8206HR provides the setup functions for an 8206-HR POD device.

static GetDeviceName() → str

Returns the name of the POD device.

Returns
String of _NAME.

Return type
str

StopStream() → None

Write a command to stop streaming data to all POD devices.

static _ChoosePreampGain() → int

Asks user for the preamplifier gain of their POD device.

Returns
Integer 10 or 100 for the preamplifier gain.

Return type
int

_ConnectPODdevice(*deviceNum: int*, *deviceParams: dict[str, str | int | dict[str, int]]*) → bool

Creates a POD_8206HR object and write the setup parameters to it.

Parameters

- **deviceNum** (*int*) – Integer of the device's number.
- **deviceParams** (*dict[str,*) – Dictionary of the device's parameters

Returns
True if connection was successful, false otherwise.

Return type
bool

_GetPODdeviceParameterTable() → Texttable

Builds a table containing the parameters for all POD devices.

Returns
Texttable containing all parameters.

Return type
Texttable

_GetParam_onePODdevice(*forbiddenNames: list[str]*) → dict[str, str | int | dict[str, int]]

Asks the user to input all the device parameters.

Parameters
forbiddenNames (*list[str]*) – List of port names already used by other devices.

Returns
Dictionary of device parameters.

Return type
dict[str,(str|int|dict[str,int])]

_IsOneDeviceValid(*paramDict: dict*) → bool

Raises an exception if the parameters dictionary is incorrectly formatted.

Parameters
paramDict (*dict*) – Dictionary of the parameters for one device.

Raises

- **Exception** – Invalid parameters.
- **Exception** – Invalid parameter value types.
- **Exception** – Invalid low-pass parameters.
- **Exception** – Invalid low-pass value types.

Returns
True if no exceptions are raised.

Return type
bool

_LOWPASSKEYS: list[str] = ['EEG1', 'EEG2', 'EEG3/EMG']

Class-level list containing the keys of the 'Low-pass' parameter dict value.

_NAME: str = '8206-HR'

Class-level string containing the POD device name.

_OpenSaveFile_EDF(*fname: str*, *devNum: int*) → EdfWriter

Opens EDF file and write header.

Parameters

- **fname** (*str*) – String filename.
- **devNum** (*int*) – Integer device number.

Returns
Opened file.

Return type
EdfWriter

_OpenSaveFile_TXT(*fname: str*) → IOBase

Opens a text file and writes the column names. Writes the current date/time at the top of the txt file.

Parameters
fname (*str*) – String filename.

Returns
Opened file.

Return type
IOBase

_PARAMKEYS: list[str] = ['Port', 'Sample Rate', 'Preamplifier Gain', 'Low-pass']

Class-level list containing the device parameter dict keys.

_PHYSICAL_BOUND_uV: int = 2046

Class-level integer representing the max/-min physical value in uV. Used for EDF files.

_StreamThreading() → dict[int, threading.Thread]

Opens a save file, then creates a thread for each device to stream and write data from.

Returns
Dictionary with keys as the device number and values as the started Thread.

Return type
dict[int,Thread]

_StreamUntilStop(*pod:* POD_8206HR, *file: IOBase | EdfWriter*, *sampleRate: int*) → None

Streams data from a POD device and saves data to file. Stops looking when a stop stream command is read. Calculates average time difference across multiple packets to collect a continuous time series data.

Parameters

- **pod** (POD_8206HR) – POD device to read from.
- **file** (*IOBase | EdfWriter*) – open file.
- **sampleRate** (*int*) – Integer sample rate in Hz.

static _WriteDataToFile_EDF(*file: EdfWriter*, *data: list[numpy.ndarray]*)

Writes data to an open EDF file.

Parameters

- **file** (*EdfWriter*) – opened EDF file.
- **data** (*list[np.ndarray]*) – List of 3 items, one for each channel.

static _WriteDataToFile_TXT(*file: IOBase*, *data: list[numpy.ndarray]*, *t: ndarray*)

Writes data to an open text file.

Parameters

- **file** (*IOBase*) – opened write file.
- **data** (*list[np.ndarray]*) – List of 3 items, one for each channel.
- **t** (*np.ndarray*) – list with the time stamps (in seconds).

1.9 Setup_8401HR module

class Setup_8401HR.**Setup_8401HR**

Bases: *Setup_Interface*

Setup_8401HR provides the setup functions for an 8206-HR POD device. REQUIRES FIRMWARE 1.0.2 OR HIGHER.

static GetDeviceName() → str

returns the name of the POD device.

Returns
String of _NAME.

Return type
str

StopStream() → None

Write a command to stop streaming data to all POD devices.

_CHANNELKEYS = ['A', 'B', 'C', 'D']

Class-level list containing the keys of 'Preamplifier Gain','Second Stage Gain', 'High-pass','Low-pass','Bias','DC Mode' parameters.

static _CodeDCmode(*dcMode: str*) → int

gets the integer payload to use for 'SET DC MODE' commands given the mode.

Parameters
dcMode (*str*) – DC mode VBIAS or AGND.

Raises
Exception – DC Mode value is not supported.

Returns
Integer code representing the DC mode.

Return type
int

static _CodeHighpass(*highpass: float*) → int

Gets the integer payload to use for 'SET HIGHPASS' given a highpass value.

Parameters
highpass (*float*) – Highpass value in Hz.

Raises
Exception – High-pass value is not supported.

Returns
Integer code representing the highpass value.

Return type
int

_ConnectPODdevice(*deviceNum: int*, *deviceParams: dict[str, str | int | dict]*) → bool

Creates a POD_8206HR object and write the setup parameters to it.

Parameters

- **deviceNum** (*int*) – Integer of the device's number.
- **deviceParams** (*dict[str,]*) – Dictionary of the device's parameters

Returns
True if connection was successful, false otherwise.

Return type
bool

_GetPODdeviceParameterTable() → Texttable

Builds a table containing the parameters for all POD devices.

Returns
Texttable containing all parameters.

Return type
Texttable

_GetParam_onePODdevice(*forbiddenNames: list[str]*) → dict[str, str | int | dict]

Asks the user to input all the device parameters.

Parameters
forbiddenNames (*list[str]*) – List of port names already used by other devices.

Returns
Dictionary of device parameters.

Return type
dict[str,(str|int|dict[str,int])]

_GetPreampDeviceName() → str

Asks the user to select a mouse/rat preamplifier.

Returns
String of the chosen preamplifier.

Return type
str

_IsChannelTypeValid(*chdict: dict*, *isType*) → bool

Checks that the keys and values for a given channel are valid.

Parameters

- **chdict** (*dict*) – dictionary with ABCD keys and isType type values.
- **isType** (*bool*) – data type.

Raises

- **Exception** – Channel dictionary is empty.
- **Exception** – Invalid channel keys.
- **Exception** – Invalid channel value.

Returns
True for valid parameters.

Return type
bool

_IsOneDeviceValid(*paramDict: dict*) → bool

Checks if the parameters for one device are valid.

Parameters
paramDict (*dict*) – Dictionary of the parameters for one device

Raises

- **Exception** – Invalid parameters.
- **Exception** – Invalid parameter value types.
- **Exception** – Preamplifier is not supported.

Returns
True for valid parameters.

Return type
bool

_NAME: str = '8401-HR'

Class-level string containing the POD device name.

_NiceABCDtableText(*abcdValueDict: dict[str, int | str | None]*, *channelMap: dict[str, str]*) → str

Builds a string that formats the channel values to be input into the parameter table.

Parameters

- **abcdValueDict** (*dict[str,int | str | None]*) – Dictionary with ABCD keys.
- **channelMap** (*dict[str,str]*) – Maps the ABCD channels to the sensor's channel name.

Returns
String with "channel name: value newline…" for each channel.

Return type
str

_OpenSaveFile_EDF(*fname: str*, *devNum: int*) → EdfWriter

Opens EDF file and write header.

Parameters

- **fname** (*str*) – String filename.
- **devNum** (*int*) – Integer device number.

Returns
Opened file.

Return type
EdfWriter

_OpenSaveFile_TXT(*fname: str*) → IOBase

Opens a text file and writes the column names. Writes the current date/time at the top of the txt file.

Parameters
fname (*str*) – String filename.

Returns
Opened file.

Return type
IOBase

_PARAMKEYS = ['Port', 'Preamplifier Device', 'Sample Rate', 'Mux Mode', 'Preamplifier Gain', 'Second Stage Gain', 'High-pass', 'Low-pass', 'Bias', 'DC Mode']

Class-level list containing the device parameter dict keys.

_PHYSICAL_BOUND_uV: int = 2046

Class-level integer representing the max/-min physical value in uV. Used for EDF files.

static _SetBias(*channelName: str*) → float

Asks the user for the bias voltage in V (+/-2.048V).

Parameters
channelName (*str*) – Name of the channel.

Returns
A float for thebias voltage in V.

Return type
float

static _SetDCMode(*channelName: str*) → str

Asks the user for the DC mode (VBIAS or AGND).

Parameters
channelName (*str*) – Name of the channel.

Returns
String for the DC mode.

Return type
str

_SetForMappedChannels(*message: str*, *channelMap: dict[str, str]*, *func: function*) → dict[str, int | None]

Asks the user to input values for all channels (excluding no-connects).

Parameters

- **message** (*str*) – Message to ask the user.
- **channelMap** (*dict[str,str]*) – Maps the ABCD channels to the sensor's channel name.
- **func** (*function*) – a function that asks the user for an input. takes one string parameter and returns one value.

Returns
Dictionary with ABCD keys and user inputs for values.

Return type
dict[str,int|None]

static _SetHighpass(*channelName: str*) → float | None

Asks the user for the high-pass in Hz (0.5,1,10Hz, or DC).

Parameters
channelName (*str*) – Name of the channel.

Returns
A float for the high-pass frequency in Hz, or None if DC.

Return type
float|None

static _SetLowpass(*channelName: str*) → int | None

Asks the user for the low-pass in Hz (21-15000Hz).

Parameters
channelName (*str*) – Name of the channel.

Returns
An integer for the low-pass frequency in Hz.

Return type
int|None

static _SetPreampGain(*channelName: str*) → int | None

Asks the user for the preamplifier gain.

Parameters
channelName (*str*) – Name of the channel.

Returns
An integer for the gain, or None if no gain.

Return type
int|None

static _SetSSGain(*channelName: str*) → int

Asks the user for the second stage gain.

Parameters
channelName (*str*) – Name of the channel.

Returns
An integer for the gain.

Return type
int

_StreamThreading() → dict[int, threading.Thread]

Opens a save file, then creates a thread for each device to stream and write data from.

Returns
Dictionary with keys as the device number and values as the started Thread.

Return type
dict[int,Thread]

_StreamUntilStop(*pod: POD_8401HR*, *file: IOBase | EdfWriter*, *sampleRate: int*, *devNum: int*) → None

Streams data from a POD device and saves data to file. Stops looking when a stop stream command is read. Calculates average time difference across multiple packets to collect a continuous time series data.

Parameters

- **pod** (*POD_8206HR*) – POD device to read from.
- **file** (*IOBase | EdfWriter*) – open file.
- **sampleRate** (*int*) – Integer sample rate in Hz.

static _WriteDataToFile_EDF(*file: EdfWriter*, *data: list[numpy.ndarray]*)

Writes data to an open EDF file.

Parameters

- **file** (*EdfWriter*) – opened EDF file.
- **data** (*list[np.ndarray]*) – List with one item for each channel.

static _WriteDataToFile_TXT(*file: IOBase*, *data: list[numpy.ndarray]*, *t: ndarray*)

Writes data to an open text file.

Parameters

- **file** (*IOBase*) – opened write file.
- **data** (*list[np.ndarray]*) – List with one item for each channel.
- **t** (*np.ndarray*) – list with the time stamps (in seconds).

1.10 Setup_PodDevices module

class Setup_PodDevices.**Setup_PodDevices**(*saveFile: str | None = None*, *podParametersDict: dict[str, dict | None] | None = None*)

Bases: object

Setup_PodDevices allows a user to set up and stream from any number of POD devices. The streamed data is saved to a file.

REQUIRES FIRMWARE 1.0.2 OR HIGHER.

_Setup_PodDevices

Dictionary containing the Setup_Interface subclasses for each POD device.

Type
dict[str,*Setup_Interface*]

_saveFileName

String containing the path, filename, and file extension to a file to save streaming data to. The filename will be extended with "_<DEVICE NAME>_<DEVICE NUMBER>" for each device.

Type
str

_options

Dictionary listing the different options for the user to complete.

Type
dict[int,str]

GetOptions() → dict[int, str]

Gets the dictionary of setup options.

Returns
Dictionary listing the different options for the user to complete (_options).

Return type
dict[int,str]

GetPODparametersDict() → dict[str, dict[int, dict]]

Sets up each POD device type. Used in initialization.

Returns
Dictionary of all POD devices initialization. The keys are the device name and the entries are the initialization dictionaries.

Return type
dict[str, dict[int, dict]]

GetSaveFileName() → str

Gets the name of the class object's save file.

Returns
String of the save file name and path (_saveFileName).

Return type
str

Run() → None

Prints the options and askes the user what to do. Loops until 'Quit" is chosen.

SetupPODparameters(*podParametersDict: dict[str, dict | None]*) → None

Sets up each POD device type. Used in initialization.

Parameters
podParametersDict (*dict[str,dict | None]*) – Dictionary of all POD devices initialization. The keys are the device name and the entries are the initialization dictionaries.

SetupSaveFile(*saveFile: str | None = None*) → None

Gets the path/file name from the user and stores it. Used in initialization.

Parameters
saveFile (*str | None, optional*) – String of the save file, which includes the directory path, filename, and file extension. Defaults to None.

_AskOption() → int

Asks user which option to do.

Returns
Integer number representing an option key.

Return type
int

_AskToStopStream() → None

Asks user to press enter to stop streaming. The program will then prompt all POD devices to end stream.

static _AskUserForDevices()

Asks the user what POD devices they want to use.

static _CheckFileExt(*f: str*, *fIsExt: bool = True*, *goodExt: list[str] = ['.csv', '.txt', '.edf']*, *printErr: bool = True*) → bool

summary

Parameters

- **f** (*str*) – file name or extension
- **fIsExt** (*bool, optional*) – Boolean flag that is true if f is an extension, false otherwise. Defaults to True.
- **goodExt** (*list[str], optional*) – List of valid file extensions. Defaults to [‘.csv’,’.txt’,’.edf’].
- **printErr** (*bool, optional*) – Boolean flag that, when true, will print an error statement. Defaults to True.

Returns
True if extension is in goodExt list, False otherwise.

Return type
bool

_CheckForValidParams(*podParametersDict: dict[str, None | dict]*) → bool

Checks if the parameters are correctly formatted.

Parameters
podParametersDict (*dict[str,None | dict]*) – Dictionary with keys as the device names and values as None or the respective parameter dictionary.

Raises

- **Exception** – Parameters must be dictionary type.
- **Exception** – Parameters dictionary is empty.
- **Exception** – Invalid device name in paramater dictionary.

Returns
True if the parameters are correctly formatted.

Return type
bool

_ConnectNewDevice() → None

Asks the user for the POD device type, then it sets up that device.

_DoOption(*choice: int*) → None

Performs the methods associated with the user selected option.

Parameters
choice (*int*) – Integer number representing an option key.

_EditCheckConnect() → None

Displays the POD devices parameters, asks the user to edit the device, and then reconnects the device for each POD device type.

_EditSaveFilePath() → None

Asks the user for the POD device type, then asks the user for a new file name and path, then sets the value to the POD devices.

_GetChosenDeviceType(*question: str*) → str

Asks the user which type of POD device they want.

Parameters
question (*str*) – Question to ask the user.

Returns
String of the user input (may be invalid POD device type).

Return type
str

static _GetFileName() → str

Asks the user for a filename.

Returns
String of the file name and extension.

Return type
str

static _GetFilePath() → str

Asks user for a path and filename to save streaming data to.

Returns
String of the file path, name, and extension.

Return type
str

_GetParams(*podParametersDict: None | dict[str, None]*) → dict[str, dict | None]

If no parameters are give, this asks user which types of POD devices they want to use. Then it checks if the parameters are valid.

Parameters
podParametersDict (*None | dict[str,None]*) – Dictionary of all POD devices initialization. The keys are the device name and the entries are the initialization dictionaries.

Returns
Dictionary whose keys are the POD device name, and value the setup dictionary.

Return type
dict[str,dict|None]

_PrintInitCode() → None

Prints code that can be used to initialize and run SetupPodDevices with the current parameters.

_PrintOptions() → None

Prints options available for user.

_PrintSaveFile() → None

Prints the file path and name that data is saved to. Note that the device name and number will be appended to the end of the filename,

_Reconnect() → bool

Reconnects all POD devices.

Returns
Bool that is true if all devices were successfully connected. False otherwise.

Return type
bool

_RemoveDevice() → None

Displays the POD devices parameters, asks the user which device ro remove, and then deletes that POD device.

_SetFilenameToDevices() → None

Sets the filename to each POD device type.

_Set_Setup_PodDevices(*podParametersDict: dict[str, dict | None]*) → None

Sets the _Setup_PodDevices varaible to have keys as the POD device name and values as the setup class.

Parameters
podParametersDict (*dict[str,dict | None]*) – Dictionary with keys as the device names and values as None or the respective parameter dictionary.

_ShowCurrentSettings() → None

Displays the POD device settings for all devices, and then prints the save file name.

_Stream() → float

Streams data from all POD devices and prints the execution time.

Returns
Float of the execution time in seconds.

Return type
float

_StreamAllDevices() → None

Streams data from all the devices. User is asked to click enter to stop streaming. Data is saved to file. Uses threading.

static _TimeFunc(*func: function*) → float

Runs a function and gets the calculated execultion time.

Parameters
func (*function*) – Function/method name.

Returns
Float of the execution time in seconds rounded to 3 decimal places.

Return type
float

1.11 Setup_PodInterface module

class Setup_PodInterface.**Setup_Interface**

Bases: object

Setup_Interface provides the basic interface of required methods for subclasses to implement. SetupPodDevices.py is designed to handle any of these children.

_podDevices

Instance-level dictionary of pod device objects. MUST have keys as device number.

Type
dict[int,*POD_Basics*]

_podParametersDict

Instance-level dictionary of device information. MUST have keys as device number, and each value must have {'_PORTKEY': str, ...other values...}.

Type
dict[int,dict]

_saveFileName

Instance-level string filename: <path>/file.ext. The device name and number will be appended to the filename.

Type
str

AreDeviceParamsValid(*paramDict: None | dict[int, dict]*)

Checks if the parameters dictionary is valid.

Parameters
paramDict (*None | dict[int,dict]*) – Dictionary of parameters for all POD devices.

Raises

- **Exception** – Parameters must be contained in a dictionary.
- **Exception** – Device keys must be integer type.
- **Exception** – Device parameters must be dictionary type.
- **Exception** – Device parameters dictionary is empty.

ConnectAllPODdevices() → bool

Connects all setup POD devices.

Returns
True if all devices are successfully connected, false otherwise.

Return type
bool

DisplayPODdeviceParameters() → None

Display all the pod device parameters in a table.

static GetDeviceName() → str

returns the name of the POD device.

Returns
String of _NAME.

Return type
str

GetPODparametersDict() → dict[int, dict]

Gets a dictionary whose keys are the device number and the value is the device parameters dict.

Returns
Dictionary of POD device parameters. The keys are the device number.

Return type
dict[int,dict]

SetFileName(*fileName: str*) → None

Sets the filename to save data to. Note that the device name and number will be appended to the end.

Parameters
fileName (*str*) – String file name.

SetupPODparameters(*podParametersDict: dict[int, dict] | None = None*) → None

Sets the parameters for the POD devices.

Parameters
podParametersDict (*dict[int,dict] | None, optional*) – dictionary of the device parameters for all devices. Defaults to None.

StopStream() → None

Write a command to stop streaming data to all POD devices.

Stream() → dict[int, threading.Thread]

Tests that all devices are connected then starts streaming data.

Raises
Exception – Test connection failed.

Returns
Dictionary with integer device number keys and Thread values.

Return type
dict[int,Thread]

_AddPODdevice() → None

Asks the user for the parameters for the new device. A new device number is generated.

_BuildFileName(*devNum: int*) → str

Appends the device name and number to the end of the file name.

Parameters
devNum (*int*) – Integer of the device number.

Returns
String file name.

Return type
str

static _ChoosePort(*forbidden: list[str] = []*) → str

Asks the user to select a COM port.

Parameters
forbidden (*list[str], optional*) – List of port names that are already used. Defaults to [].

Returns
String name of the port.

Return type
str

_ConnectPODdevice(*deviceNum: int*, *deviceParams: dict[str, str | int | dict]*) → bool

Creates a POD device object and write the setup parameters to it.

Parameters

- **deviceNum** (*int*) – Integer of the device's number.
- **deviceParams** (*dict[str,*) – Dictionary of the device's parameters

Returns
True if connection was successful, false otherwise.

Return type
bool

_DisconnectAllPODdevices() → None

Disconnects all POD devices by deleted all POD obejcts.

_EditParams() → None

Asks the user which device to edit, and then asks them to re-input the device parameters.

_GetForbiddenNames(*key: str = 'Port'*, *exclude: str | None = None*) → list[str]

Generates a list of port names used by the active pod devices. There is an option to exclude an additional name from the list.

Parameters

- **key** (*str, optional*) – String key to access the _podParametersDict. Defaults to 'Port'.
- **exclude** (*str | None, optional*) – String port name to exclude from the returned list. Defaults to None.

Returns
List of string names of ports in use.

Return type
list[str]

_GetPODdeviceParameterTable() → Texttable

Builds a table containing the parameters for all POD devices.

Returns
Texttable containing all parameters.

Return type
Texttable

_GetParam_onePODdevice(*forbiddenNames: list[str]*) → dict[str, str | int | dict]

Asks the user to input all the device parameters.

Parameters
forbiddenNames (*list[str]*) – List of port names already used by other devices.

Returns
Dictionary of device parameters.

Return type
dict[str,(str|int|dict[str,int])]

static _GetPortsList(*forbidden: list[str] = []*) → list[str]

Gets the names of all available ports.

Parameters
forbidden (*list[str], optional*) – List of port names that are already used. Defaults to [].

Returns
List of port names.

Return type
list[str]

static _GetTimeHeader_forTXT() → str

Builds a string containing the current date and time to be written to the text file header.

Returns
String containing the date and time. Each line begins with '#' and ends with a newline.

Return type
str

_IsOneDeviceValid(*paramDict: dict*) → bool

Checks if the parameters for one device are valid.

Parameters
paramDict (*dict*) – Dictionary of the parameters for one device.

Raises

- **Exception** – Invalid parameters.
- **Exception** – Invalid parameter value types.
- **Exception** – Preamplifier is not supported.

Returns
True for valid parameters.

Return type
bool

_NAME: str = 'GENERIC'

Class-level string for the Device name. This should be overwritten by child subclasses.

_OpenSaveFile(*devNum: int*) → IOBase | EdfWriter

Opens a save file for a given device.

Parameters
devNum (*int*) – Integer of the device number.

Returns
Opened file. IOBase for a text file, or EdfWriter for EDF file.

Return type
IOBase | EdfWriter

_OpenSaveFile_EDF(*fname: str*, *devNum: int*) → EdfWriter

Opens EDF file and write header.

Parameters

- **fname** (*str*) – String filename.

- **devNum** (*int*) – Integer device number.

Returns
Opened file.

Return type
EdfWriter

_OpenSaveFile_TXT(*fname: str*) → IOBase

Opens a text file and writes the column names. Writes the current date/time at the top of the txt file.

Parameters
fname (*str*) – String filename.

Returns
Opened file.

Return type
IOBase

_PORTKEY: str = 'Port'

Class-level string that is the parameter's dictionary key for the COM port.

static _PrintDeviceNumber(*num: int*) → None

Prints a title with the device number.

Parameters
num (*int*) – Integer of the device number.

_RemoveDevice() → None

Asks the user for a device number to remove, then deletes that device. This will only remove a device if there are more than one options.

_SelectDeviceFromDict(*action: str*) → int

Asks the user to select a valid device number. The input must be an integer number of an existing device.

Parameters
action (*str*) – Description of the action to be performed on the device.

Returns
Integer for the device number.

Return type
int

static _SetNumberOfDevices(*name: str*) → int

Asks the user for how many devices they want to setup.

Parameters
name (*str*) – Name of the POD device type.

Returns
Integer number of POD devices desired by the user.

Return type
int

_SetParam_allPODdevices() → None

First gets the number of POD devices, then asks the user for the information for each device.

_StreamThreading() → dict[int, threading.Thread]

Opens a save file, then creates a thread for each device to stream and write data from.

Returns
Dictionary with keys as the device number and values as the started Thread.

Return type
dict[int,Thread]

static _TestDeviceConnection(*pod:* POD_Basics) → bool

Tests if a POD device can be read from or written. Sends a PING command.

Parameters
pod (POD_Basics) – POD device to read to and write from.

Returns
True for successful connection, false otherwise.

Return type
bool

_TestDeviceConnection_All() → bool

Tests the connection of all setup POD devices.

Returns
True when all devices are successfully connected, false otherwise

Return type
bool

_ValidateParams() → None

Displays a table of the parameters of all devices, then asks the user if everything is correct. The user can then edit the parameters of a device.

static _uV(*voltage: float | int*) → float

Converts volts to microVolts, rounded to 6 decimal places.

Parameters
voltage (*float | int*) – number of volts.

Returns
voltage in of uV.

Return type
float

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

G

I

M

N

O

P

R

S

T

U

W