pyXCP Documentation

Release 0.9

Christoph Schueler

Jun 15, 2022

Contents:

CHAPTER 1

Installation and Getting Started

Pythons: *Python* >= 3.4 (*PyPy* not tested yet).

Platforms: No platform-specific restrictions besides availability of communication (CAN-bus) drivers.

Documentation: get latest.

1.1 Prerequisites

CHAPTER 2

Tutorial

CHAPTER 3

Configuration

Parameters live in *JSON* or *TOML* `pyxcp/examples` contains some example configurations.

3.1 General pyXCP Parameters

- ***LOGLEVEL*: str, False, "WARN" – "ERROR | "WARN" | "INFO" | "DEBUG"** Verbosity of logger.
- *DISABLE_ERROR_HANDLING*: bool, False, False – Bypass error-handling for performance reasons (expert option!!!).
- *CREATE_DAQ_TIMESTAMPS*: bool, False, False – Generate DAQ records with timestamp.
- *TIMEOUT*: float, False, 2.0 – General XCP timeout in seconds.
- *ALIGNMENT*: int, False, 1 – 1 | 2 | 4, byte alignment.

3.1.1 eth

These parameters are rather self-explanatory.

- *HOST*: str, "localhost"
- *PORT*: int, 5555
- *PROTOCOL*: str, "TCP", "TCP" | "UDP"
- *IPV6*: bool, False
- *TCP_NODELAY*: bool, False

3.1.2 sxi

Again, obvious parameters.

PORT: str, "COM1" *BITRATE*: int, 38400 *BYTESIZE*: int, 8 *PARITY*: str, "N" *STOPBITS*: int, 1

3.2 General CAN Parameters

- ***CAN_DRIVER*: str, REQUIRED – "Canalystii" | "IsCAN" | "Ixxat" | "Kvaser" | "Neovi" | "NiCan" |** "PCan" | "Serial" | "SlCan" | "SocketCAN" | "Systec" | "Usb2Can" | "Vector" (the driver names reflect the correspondending class names).
- *CHANNEL*: str, "" – Highly driver specific value, see documentation.
- *MAX_DLC_REQUIRED*: bool, False – if True, DLC is set to MAX_DLC, e.g. 8 on CAN Classic, unused bytes are set to zero.
- ***CAN_USE_DEFAULT_LISTENER*: bool, True – if True, the default listener thread is used.** If the canInterface implements a listener service, this parameter can be set to False, and the default listener thread won't be started.
- *CAN_ID_MASTER*: int, REQUIRED
- *CAN_ID_SLAVE*: int, REQUIRED
- *CAN_ID_BROADCAST*: int, REQUIRED
- *BITRATE*: int, 250000
- *RECEIVE_OWN_MESSAGES*: bool, False

3.3 Specific CAN Drivers

Every driver has some additional parameters, not further explained here, please refer to the python-can documentation.

3.3.1 canalystii

- *BAUD*: int, None – Uses *BAUD* instead of *BITRATE*.
- *TIMING0*: int, None
- *TIMING1*: int, None

3.3.2 iscan

- *POLL_INTERVAL*: float, 0.01

3.3.3 ixxat

- *UNIQUE_HARDWARE_ID*: str, None
- *RX_FIFO_SIZE*: int, 16
- *TX_FIFO_SIZE*: int, 16

3.3.4 kvaser

- *ACCEPT_VIRTUAL*: bool, True
- *DRIVER_MODE*: bool, True

- *NO_SAMP*: int, 1
- *SJW*: int, 2
- *TSEG1*: int, 5
- *TSEG2*: int, 2
- *SINGLE_HANDLÈ*: bool, True
- *FD*: bool, False
- *DATA_BITRATE*: int, None

3.3.5 neovi

- *FD*: bool, False
- *DATA_BITRATE*: int, None
- *USE_SYSTEM_TIMESTAMP*: bool, False
- *SERIAL*: str, None
- *OVERRIDE_LIBRARY_NAME*: str, None

3.3.6 nican

- *LOG_ERRORS*: bool, False

3.3.7 pcan

- *STATE*: str, "ACTIVE"

3.3.8 serial

- *BAUDRATE* int, 115200 – Uses *BAUDRATE* instead of *BITRATE*.
- *TIMEOUT*: float, 0.1
- *RTSCTS*: bool, False

3.3.9 slcan

- *TTY_BAUDRATE*: int, 115200
- *POLL_INTERVAL*: float, 0.01
- *SLEEP_AFTER_OPEN*: float, 2.0
- *RTSCTS*: bool, False

3.3.10 socketcan

- *FD*: bool, False

3.3.11 systec

- *DEVICE_NUMBER*: int, 255
- *RX_BUFFER_ENTRIES*: int, 4096
- *TX_BUFFER_ENTRIES*: int, 4096
- *STATE*: str, “ACTIVE”

3.3.12 usb2can

FLAGS: int, 0

3.3.13 vector

- *POLL_INTERVAL*: float, 0.01
- *APP_NAME*: str, “”
- *SERIAL*: int, None
- *RX_QUEUE_SIZE*: int, 16384
- *FD*: bool, False
- *DATA_BITRATE*: int, None

CHAPTER 4

HOW-TOs

4.1 How-to write your own command-line tools

4.2 How-to build your own CAN drivers

CHAPTER 5

API Documentation

5.1 pyxcp package

5.1.1 Subpackages

pyxcp.asam package

Submodules

pyxcp.asam.compumethod module

pyxcp.asam.types module

class pyxcp.asam.types.**A_Float32**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_FLOAT32 codec.

FMT = 'f'

class pyxcp.asam.types.**A_Float64**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_FLOAT64 codec.

FMT = 'd'

class pyxcp.asam.types.**A_Int16**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_INT16 codec.

FMT = 'h'

class pyxcp.asam.types.**A_Int32**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_INT32 codec.

FMT = 'i'

class pyxcp.asam.types.**A_Int64**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_INT64 codec.

FMT = 'q'

class pyxcp.asam.types.**A_Int8**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_INT8 codec.

FMT = 'b'

class pyxcp.asam.types.**A_Uint16**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_UINT16 codec.

FMT = 'H'

class pyxcp.asam.types.**A_Uint32**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_UINT32 codec.

FMT = 'I'

class pyxcp.asam.types.**A_Uint64**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_UINT64 codec.

FMT = 'Q'

class pyxcp.asam.types.**A_Uint8**(*byteorder*)
Bases: *pyxcp.asam.types.AsamBaseType*

ASAM A_UINT8 codec.

FMT = 'B'

class pyxcp.asam.types.**AsamBaseType**(*byteorder*)
Bases: object

Base class for ASAM codecs.

Note: Always use derived classes.

---

decode(*value*)
Decode a value.

Decode means convert a byte-string to a meaningful data-type, eg. an integer.

Parameters **value** (*bytes*) –

Returns data-type is determined by derived class.

Return type data-type

encode(*value*)
Encode a value.

Encode means convert a value, eg. an integer, to a byte-string.

Parameters **value** (*data-type*) – data-type is determined by derived class.

Returns Encoded value.

Return type bytes

pyxcp.asam.types.**MOTOROLA = '>'**
pseudo type for non-existing elements A_BIT: one bit A_ASCIISTRING: string, ISO-8859-1 encoded A_UTF8STRING: string, UTF-8 encoded A_UNICODE2STRING: string, UCS-2 encoded A_BYTEFIELD: Field of bytes

Type A_VOID

Module contents

pyxcp.master package

Submodules

pyxcp.master.base module

pyxcp.master.pre35 module

pyxcp.master.py35 module

Module contents

Lowlevel API reflecting available XCP services

Note: For technical reasons the API is split into two parts; common methods and a Python version specific part.

pyxcp.transport package

Subpackages

Submodules

pyxcp.transport.base module

class pyxcp.transport.base.**BaseTransport**(*config=None*)
Bases: object

Base class for transport-layers (Can, Eth, Sxi).

Parameters

- **config** (*dict-like*) – Parameters like bitrate.

- **loglevel** (*["INFO", "WARN", "DEBUG", "ERROR", "CRITICAL"]*) – Controls the verbosity of log messages.

PARAMETER_MAP = {'ALIGNMENT': (<class 'int'>, False, 1), 'CREATE_DAQ_TIMESTAMPS': (<cl

block_receive(*length_required: int*) → bytes
Implements packet reception for block communication model (e.g. for XCP on CAN)

Parameters **length_required** (*int*) – number of bytes to be expected in block response packets

Returns all payload bytes received in block response packets

Return type bytes

Raises *pyxcp.types.XcpTimeoutError* –

block_request(*cmd*, **data*)
Implements packet transmission for block communication model (e.g. DOWNLOAD block mode) All parameters are the same as in request(), but it does not receive response.

close()
Close the transport-layer connection and event-loop.

closeConnection()
Does the actual connection shutdown. Needs to be implemented by any sub-class.

connect()

finishListener()

listen()

loadConfig(*config*)
Load configuration data.

processResponse(*response*, *length*, *counter*, *recv_timestamp=None*)

process_event_packet(*packet*)

request(*cmd*, **data*)

request_optional_response(*cmd*, **data*)

send(*frame*)

startListener()

exception pyxcp.transport.base.**Empty**
Bases: Exception

pyxcp.transport.base.**availableTransports**()
List all subclasses of *BaseTransport*.

Returns name: class

Return type dict

pyxcp.transport.base.**createTransport**(*name*, **args*, ***kws*)
Factory function for transports.

Returns

Return type *BaseTransport* derived instance.

pyxcp.transport.base.**get**(*q*, *timeout*, *restart_event*)
Get an item from a deque considering a timeout condition.

pyxcp.transport.can module

class pyxcp.transport.can.**Can**(*config=None*)
Bases: *pyxcp.transport.base.BaseTransport*

HEADER = <pyxcp.transport.can.EmptyHeader object>

HEADER_SIZE = 0

MAX_DATAGRAM_SIZE = 7

PARAMETER_MAP = {'BITRATE': (<class 'int'>, False, 250000), 'CAN_DRIVER': (<class 'str

PARAMETER_TO_KW_ARG_MAP = {'BITRATE': 'bitrate', 'CHANNEL': 'channel', 'RECEIVE_OWN_ME

close()
Close the transport-layer connection and event-loop.

closeConnection()
Does the actual connection shutdown. Needs to be implemented by any sub-class.

connect()

dataReceived(*payload: bytes*, *recv_timestamp: float = None*)

listen()

send(*frame*)

class pyxcp.transport.can.**CanInterfaceBase**
Bases: `object`

Abstract CAN interface handler that can be implemented for any actual CAN device driver

PARAMETER_MAP = {}

close()
Must implement any required action for disconnecting from the can interface

connect()
Open connection to can interface

getTimestampResolution()
Get timestamp resolution in nano seconds.

init(*parent*, *receive_callback*)
Must implement any required action for initing the can interface

Parameters

- **parent** (*Can*) – Refers to owner.
- **receive_callback** (*callable*) – Receive callback function to register with the following argument: payload: bytes

loadConfig(*config*)
Load configuration data.

read()
Read incoming data

transmit(*payload: bytes*)
Must transmit the given payload on the master can id.

Parameters **payload** (*bytes*) – payload to transmit

class pyxcp.transport.can.**EmptyHeader**
Bases: object

There is no header for XCP on CAN

pack(**args*, ***kwargs*)

class pyxcp.transport.can.**Frame**(*id_: pyxcp.transport.can.Identifier*, *dlc: int*, *data: bytes*, *timestamp: int*)
Bases: object

class pyxcp.transport.can.**Identifier**(*raw_id: int*)
Bases: object

Convenience class for XCP formatted CAN identifiers.

raw_id: int Bit 32 set (i.e. 0x80000000) signals an extended (29-bit) identifier.

Raises ***IdentifierOutOfRangeError*** –

id

Returns Identifier as seen on bus.

Return type int

is_extended

Returns

- True - 29-bit identifier.
- False - 11-bit identifier.

Return type bool

static make_identifier(*identifier: int*, *extended: bool*) → int
Factory method.

Parameters

- **identifier** (*int*) – Identifier as seen on bus.
- **extended** (*bool*) –

  bool

  - True - 29-bit identifier.
  - False - 11-bit identifier.

Returns

Return type *Identifier*

Raises ***IdentifierOutOfRangeError*** –

raw_id

Returns Raw XCP formatted identifier.

Return type int

exception pyxcp.transport.can.**IdentifierOutOfRangeError**
Bases: Exception

Signals an identifier greater then MAX_11_BIT_IDENTIFIER or MAX_29_BIT_IDENTIFIER.

pyxcp.transport.can.**calculateFilter**(*ids: list*)

Parameters **ids** – An iterable (usually list or tuple) containing CAN identifiers.

Returns Calculated filter and mask.

Return type tuple (int, int)

pyxcp.transport.can.**isExtendedIdentifier**(*identifier: int*) → bool
Check for extendend CAN identifier.

Parameters **identifier** (*int*) –

Returns

Return type bool

pyxcp.transport.can.**padFrame**(*frame: bytes*, *padding_value: int*, *padding_len: int = 0*) → bytes
Pad frame to next discrete DLC value.

ISO/DIS 15765 - 4; 8.2 Data length Code (DLC) AUTOSAR CP Release 4.3.0, Specification of CAN Transport Layer; 7.3.8 N-PDU padding AUTOSAR CP Release 4.3.0, Specification of CAN Driver; [SWS_CAN_00502], [ECUC_Can_00485] AUTOSAR CP Release 4.3.0, Requirements on CAN; [SRS_Can_01073], [SRS_Can_01086], [SRS_Can_01160]

pyxcp.transport.can.**registered_drivers**()

Returns Dictionary containing CAN driver names and classes of all available drivers (pyxcp supplied and user-defined).

Return type dict (name, class)

pyxcp.transport.can.**samplePointToTsegs**(*tqs: int*, *samplePoint: float*) → tuple
Calculate TSEG1 and TSEG2 from time-quantas and sample-point.

Parameters

- **tqs** (*int*) – Number of time-quantas
- **samplePoint** (*float or int*) – Sample-point as a percentage value.

Returns

Return type tuple (TSEG1, TSEG2)

pyxcp.transport.can.**setDLC**(*length: int*)
Return DLC value according to CAN-FD.

Parameters **length** – Length value to be mapped to a valid CAN-FD DLC. (0 <= length <= 64)

pyxcp.transport.can.**stripIdentifier**(*identifier: int*) → int
Get raw CAN identifier (remove CAN_EXTENDED_ID bit if present).

Parameters **identifier** (*int*) –

Returns

Return type int

pyxcp.transport.can.**try_to_install_system_supplied_drivers**()
Register available pyxcp CAN drivers.

pyxcp.transport.eth module

class pyxcp.transport.eth.**Eth**(*config=None*)
Bases: *pyxcp.transport.base.BaseTransport*

HEADER = <Struct object>

HEADER_SIZE = 4

MAX_DATAGRAM_SIZE = 512

PARAMETER_MAP = {'HOST': (<class 'str'>, False, 'localhost'), 'IPV6': (<class 'bool'>

close()
Close the transport-layer connection and event-loop.

closeConnection()
Does the actual connection shutdown. Needs to be implemented by any sub-class.

connect()

invalidSocket

listen()

send(*frame*)

startListener()

pyxcp.transport.sxi module

class pyxcp.transport.sxi.**SxI**(*config=None*)
Bases: *pyxcp.transport.base.BaseTransport*

HEADER = <Struct object>

HEADER_SIZE = 4

MAX_DATAGRAM_SIZE = 512

PARAMETER_MAP = {'BITRATE': (<class 'int'>, False, 38400), 'BYTESIZE': (<class 'int'>,

TIMEOUT = 0.75

closeConnection()
Does the actual connection shutdown. Needs to be implemented by any sub-class.

connect()

flush()

listen()

output(*enable*)

send(*frame*)

Module contents

5.1.2 Submodules

5.1.3 pyxcp.checksum module

Checksum calculation for memory ranges

pyxcp.checksum.**ADD11**(*frame*)

pyxcp.checksum.**ADD12**(*frame*)

pyxcp.checksum.**ADD14**(*frame*)

pyxcp.checksum.**ADD22**(*frame*)

pyxcp.checksum.**ADD24**(*frame*)

pyxcp.checksum.**ADD44**(*frame*)

class pyxcp.checksum.**Algorithm**
Bases: enum.IntEnum

Enumerates available checksum algorithms

XCP_ADD_11 = 1

XCP_ADD_12 = 2

XCP_ADD_14 = 3

XCP_ADD_22 = 4

XCP_ADD_24 = 5

XCP_ADD_44 = 6

XCP_CRC_16 = 7

XCP_CRC_16_CITT = 8

XCP_CRC_32 = 9

XCP_USER_DEFINED = 10

pyxcp.checksum.**CRC32**(*x*)

class pyxcp.checksum.**Crc16**(*table*, *initalRemainder*, *finalXorValue*, *reflectData*, *reflectRemainder*)
Bases: object

Calculate CRC (16-bit)

Parameters

- **table** (*list-like*) – lookup table for CRC calculation
- **initalRemainder** (*int*) – value to start with
- **finalXorValue** (*int*) – final XOR value
- **reflectData** (*bool*) – reflect input data
- **reflectRemainder** (*bool*) – reflect output data
- **[1] A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS** (.) – http://www.ross.net/crc/download/crc_v3.txt
- **[2] Understanding and implementing CRC (Cyclic Redundancy Check)** (.) – calculation http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html
- **[3] Online CRC calculator** (.) – http://zorc.breitbandkatze.de/crc.html

WIDTH = 16

reflectIn(*ch*, *remainder*)

reflectOut(*remainder*)

pyxcp.checksum.**adder**(*modulus*)
Factory function for modulus adders

Parameters **modulus** (*int*) – modulus to use

Returns adder function

Return type function

Examples

```
>>> a256=adder(256)
>>> a256([11, 22, 33, 44, 55, 66, 77, 88, 99])
239
```

pyxcp.checksum.**check**(*frame*, *algo*)
Calculate checksum using given algorithm

Parameters

- **frame** (*list of integers*) –
- **algo** (*ALGO*) –

Returns

Return type int

pyxcp.checksum.**reflect**(*data*, *nBits*)
Reflect data, i.e. reverse bit order.

Parameters

- **data** (*int*) –
- **nBits** (*int*) – width in bits of *data*

pyxcp.checksum.**userDefined**(*x*)
User defined algorithms are not supported yet.

pyxcp.checksum.**wordSum**(*modulus*, *step*)
Factory function for (double-)word modulus sums

Parameters

- **modulus** (*int*) –
- **step** (*[2, 4]*) – 2 - word wise 4 - double-word wise

Returns summation function

Return type function

5.1.4 pyxcp.config module

class pyxcp.config.**Configuration**(*parameters*, *config*)
Bases: object

get(*key*)

pyxcp.config.**readConfiguration**(*conf*)
Read a configuration file either in JSON or TOML format.

5.1.5 pyxcp.errormatrix module

Types and structures to support error-handling as specified by XCP.

class pyxcp.errormatrix.**Action**
Bases: enum.IntEnum

Action to be taken, s. XCP spec.

DISPLAY_ERROR = 1

NEW_FLASH_WARE = 12

NONE = 0

REPEAT = 6

REPEAT_2_TIMES = 7

REPEAT_INF_TIMES = 8

RESTART_SESSION = 9

RETRY_PARAM = 3

RETRY_SYNTAX = 2

SKIP = 11

TERMINATE_SESSION = 10

USE_A2L = 4

USE_ALTERATIVE = 5

class pyxcp.errormatrix.**Handler**(*preAction*, *action*)
Bases: tuple

action
Alias for field number 1

preAction
Alias for field number 0

class pyxcp.errormatrix.**PreAction**
Bases: enum.IntEnum

Pre-action to be taken, s. XCP spec.

DISPLAY_ERROR = 9

DOWNLOAD = 10

GET_SEED_UNLOCK = 3

NONE = 0

PROGRAM = 11

REINIT_DAQ = 8

SET_DAQ_PTR = 6

SET_MTA = 4

START_STOP_X = 7

SYNCH = 2

UNLOCK_SLAVE = 13

UPLOAD = 12

WAIT_T7 = 1

class pyxcp.errormatrix.**Severity**
Bases: enum.IntEnum

Severity of error. — S0 = Information S1 = Warning / Request S2 = Resolvable Error S3 = Fatal Error

S0 = 0

S1 = 1

S2 = 2

S3 = 3

class pyxcp.errormatrix.**Timeout**
Bases: enum.IntEnum

Various timeouts, s. XCP spec.

T1 = 0

T2 = 1

T3 = 2

T4 = 3

T5 = 4

T6 = 5

T7 = 6

5.1.6 pyxcp.logger module

class pyxcp.logger.**Logger**(*name*, *level=30*)
Bases: object

FORMAT = '[%(levelname)s (%(name)s)]: %(message)s'

LOGGER_BASE_NAME = 'pyxcp'

critical(*message*)

debug(*message*)

error(*message*)

getLastError()

info(*message*)

log(*message*, *level*)

setLevel(*level*)

silent()

verbose()

warn(*message*)

5.1.7 pyxcp.types module

class pyxcp.types.**Command**
Bases: enum.IntEnum

An enumeration.

ALLOC_DAQ = 213

ALLOC_ODT = 212

ALLOC_ODT_ENTRY = 211

BUILD_CHECKSUM = 243

CLEAR_DAQ_LIST = 227

CONNECT = 255

COPY_CAL_PAGE = 228

DBG_ATTACH = 12647424

DBG_EXCLUSIVE_TARGET_ACCESS = 12647432

DBG_GET_HWIO_INFO = 12647429

DBG_GET_JTAG_ID = 12647427

DBG_GET_MODE_INFO = 12647426

DBG_GET_TRI_DESC_TBL = 12647444

DBG_GET_VENDOR_INFO = 12647425

DBG_HALT_AFTER_RESET = 12647428

DBG_HWIO_CONTROL = 12647431

DBG_LLBT = 12647445

DBG_LLT = 12647434

DBG_READ = 12647441

DBG_READ_CAN1 = 12647442

DBG_READ_CAN2 = 12647443

DBG_READ_MODIFY_WRITE = 12647435

DBG_SEQUENCE_MULTIPLE = 12647433

DBG_SET_HWIO_EVENT = 12647430

DBG_WRITE = 12647436

DBG_WRITE_CAN1 = 12647438

DBG_WRITE_CAN2 = 12647439

DBG_WRITE_CAN_NEXT = 12647440

DBG_WRITE_NEXT = 12647437

DISCONNECT = 254

DOWNLOAD = 240

DOWNLOAD_MAX = 238

```
    DOWNLOAD_NEXT = 239
    DTO_CTR_PROPERTIES = 197
    FREE_DAQ = 214
    GET_CAL_PAGE = 234
    GET_COMM_MODE_INFO = 251
    GET_DAQ_CLOCK = 220
    GET_DAQ_EVENT_INFO = 215
    GET_DAQ_LIST_INFO = 216
    GET_DAQ_LIST_MODE = 223
    GET_DAQ_PACKED_MODE = 49154
    GET_DAQ_PROCESSOR_INFO = 218
    GET_DAQ_RESOLUTION_INFO = 217
    GET_ID = 250
    GET_PAGE_INFO = 231
    GET_PAG_PROCESSOR_INFO = 233
    GET_PGM_PROCESSOR_INFO = 206
    GET_SECTOR_INFO = 205
    GET_SEED = 248
    GET_SEGMENT_INFO = 232
    GET_SEGMENT_MODE = 229
    GET_STATUS = 253
    GET_VERSION = 49152
    MODIFY_BITS = 236
    PROGRAM = 208
    PROGRAM_CLEAR = 209
    PROGRAM_FORMAT = 203
    PROGRAM_MAX = 201
    PROGRAM_NEXT = 202
    PROGRAM_PREPARE = 204
    PROGRAM_RESET = 207
    PROGRAM_START = 210
    PROGRAM_VERIFY = 200
    READ_DAQ = 219
    SET_CAL_PAGE = 235
    SET_DAQ_LIST_MODE = 224
    SET_DAQ_PACKED_MODE = 49153
```

SET_DAQ_PTR = 226

SET_MTA = 246

SET_REQUEST = 249

SET_SEGMENT_MODE = 230

SHORT_DOWNLOAD = 237

SHORT_UPLOAD = 244

START_STOP_DAQ_LIST = 222

START_STOP_SYNCH = 221

SYNCH = 252

TIME_CORRELATION_PROPERTIES = 198

TRANSPORT_LAYER_CMD = 242

UNLOCK = 247

UPLOAD = 245

USER_CMD = 241

WRITE_DAQ = 225

WRITE_DAQ_MULTIPLE = 199

class pyxcp.types.**CommandCategory**
Bases: enum.IntEnum

Values reflect resources (resource protection status / unlock).

CAL_PAG = 1

DAQ = 4

PGM = 16

STD = 0

STIM = 8

class pyxcp.types.**DaqPtr**(*daqListNumber*, *odtNumber*, *odtEntryNumber*)
Bases: tuple

daqListNumber
Alias for field number 0

odtEntryNumber
Alias for field number 2

odtNumber
Alias for field number 1

class pyxcp.types.**Event**
Bases: enum.IntEnum

XCP Event Codes

EV_CLEAR_DAQ = 1

EV_CMD_PENDING = 5

EV_DAQ_OVERLOAD = 6

EV_RESUME_MODE = 0

EV_SESSION_TERMINATED = 7

EV_SLEEP = 10

EV_STIM_TIMEOUT = 9

EV_STORE_CAL = 3

EV_STORE_DAQ = 2

EV_TIME_SYNC = 8

EV_TRANSPORT = 255

EV_USER = 254

EV_WAKE_UP = 11

exception pyxcp.types.**FrameSizeError**
Bases: Exception

A frame with an invalid size was received.

class pyxcp.types.**MtaType**(*address*, *ext*)
Bases: tuple

address
Alias for field number 0

ext
Alias for field number 1

class pyxcp.types.**XcpGetIdType**
Bases: enum.IntEnum

An enumeration.

ASCII_TEXT = 0

ECU = 6

EPK = 5

FILENAME = 1

FILE_AND_PATH = 2

FILE_TO_UPLOAD = 4

SYSID = 7

URL = 3

VECTOR_MAPNAMES = 219

VECTOR_MDI = 220

class pyxcp.types.**XcpGetSeedMode**
Bases: enum.IntEnum

An enumeration.

FIRST_PART = 0

REMAINING = 1

exception pyxcp.types.**XcpResponseError**
Bases: Exception

Raise an *exception* from an XCP error packet.

get_error_code()

exception pyxcp.types.**XcpTimeoutError**
Bases: Exception

Timeout while waiting for a response occured.

5.1.8 pyxcp.utils module

pyxcp.utils.**delay**(*amount: float*)
Performe a busy-wait delay, which is much more precise than *time.sleep*

pyxcp.utils.**flatten**(**args*)

pyxcp.utils.**getPythonVersion**()

pyxcp.utils.**hexDump**(*arr*)

pyxcp.utils.**slicer**(*iterable*, *sliceLength*, *converter=None*)

pyxcp.utils.**time_perfcounter_correlation**()
Get the *perf_counter* value nearest to when time.time() is updated if the *time.time* on this platform has a resolution higher than 10us. This is tipical for the Windows platform were the beste resolution is ~500us.

On non Windows platforms the current time and perf_counter is directly returned since the resolution is tipical ~1us.

Note this value is based on when *time.time()* is observed to update from Python, it is not directly returned by the operating system.

Returns (t, performance_counter) time.time value and perf_counter value when the time.time is updated

pyxcp.utils.**ConnectResponse**
Some Doc

5.1.9 Module contents

Universal Calibration Protocol for Python

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X