

---

# pyXCP Documentation

*Release 0.9*

**Christoph Schueler**

**Jun 15, 2022**



---

## Contents:

---

<b>1</b>	<b>Installation and Getting Started</b>	<b>1</b>
1.1	Prerequisites . . . . .	1
<b>2</b>	<b>Tutorial</b>	<b>3</b>
<b>3</b>	<b>Configuration</b>	<b>5</b>
3.1	General pyXCP Parameters . . . . .	5
3.2	General CAN Parameters . . . . .	6
3.3	Specific CAN Drivers . . . . .	6
<b>4</b>	<b>HOW-TOs</b>	<b>9</b>
4.1	How-to write your own command-line tools . . . . .	9
4.2	How-to build your own CAN drivers . . . . .	9
<b>5</b>	<b>API Documentation</b>	<b>11</b>
5.1	pyxcp package . . . . .	11
<b>6</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



# CHAPTER 1

---

## Installation and Getting Started

---

**Pythons:** *Python*  $\geq 3.4$  (*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

---





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” | “SICan” | “SocketCAN” | “Systec” | “Usb2Can” | “Vector” (the driver names reflect the corresponding 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\_HANDLE*: 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

**4.1 How-to write your own command-line tools**

**4.2 How-to build your own CAN drivers**



## 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\_DAO\_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'>, False, None), 'RECEIVE_OWN_MESSAGE': (<class 'bool'>, False, None)}
    PARAMETER_TO_KW_ARG_MAP = {'BITRATE': 'bitrate', 'CHANNEL': 'channel', 'RECEIVE_OWN_MESSAGE': 'receive_own_message'}

    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** `IdentifierOutOfRangeException` –

**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** `IdentifierOutOfRangeException` –

**raw\_id**

**Returns** Raw XCP formatted identifier.

**Return type** `int`

**exception** `pyxcp.transport.can.IdentifierOutOfRangeException`

Bases: `Exception`

Signals an identifier greater than `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 extended 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, initialRemainder, finalXorValue, reflectData, reflectRemainder*)

Bases: object

Calculate CRC (16-bit)

#### Parameters

- **table** (*list-like*) – lookup table for CRC calculation
- **initialRemainder** (*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](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](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 occurred.

## 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 typical 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 typical ~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`



**p**

`pyxcp`, 27  
`pyxcp.asam`, 13  
`pyxcp.asam.types`, 11  
`pyxcp.checksum`, 18  
`pyxcp.config`, 20  
`pyxcp.errormatrix`, 21  
`pyxcp.logger`, 22  
`pyxcp.master`, 13  
`pyxcp.transport`, 18  
`pyxcp.transport.base`, 13  
`pyxcp.transport.can`, 15  
`pyxcp.transport.eth`, 17  
`pyxcp.transport.sxi`, 18  
`pyxcp.types`, 23  
`pyxcp.utils`, 27



## A

A\_Float32 (class in *pyxcp.asam.types*), 11  
 A\_Float64 (class in *pyxcp.asam.types*), 11  
 A\_Int16 (class in *pyxcp.asam.types*), 11  
 A\_Int32 (class in *pyxcp.asam.types*), 11  
 A\_Int64 (class in *pyxcp.asam.types*), 12  
 A\_Int8 (class in *pyxcp.asam.types*), 12  
 A\_Uint16 (class in *pyxcp.asam.types*), 12  
 A\_Uint32 (class in *pyxcp.asam.types*), 12  
 A\_Uint64 (class in *pyxcp.asam.types*), 12  
 A\_Uint8 (class in *pyxcp.asam.types*), 12  
 Action (class in *pyxcp.errormatrix*), 21  
 action (*pyxcp.errormatrix.Handler* attribute), 21  
 ADD11 () (in module *pyxcp.checksum*), 18  
 ADD12 () (in module *pyxcp.checksum*), 18  
 ADD14 () (in module *pyxcp.checksum*), 18  
 ADD22 () (in module *pyxcp.checksum*), 19  
 ADD24 () (in module *pyxcp.checksum*), 19  
 ADD44 () (in module *pyxcp.checksum*), 19  
 adder () (in module *pyxcp.checksum*), 19  
 address (*pyxcp.types.MtaType* attribute), 26  
 Algorithm (class in *pyxcp.checksum*), 19  
 ALLOC\_DAQ (*pyxcp.types.Command* attribute), 23  
 ALLOC\_ODT (*pyxcp.types.Command* attribute), 23  
 ALLOC\_ODT\_ENTRY (*pyxcp.types.Command* attribute), 23  
 AsamBaseType (class in *pyxcp.asam.types*), 12  
 ASCII\_TEXT (*pyxcp.types.XcpGetIdType* attribute), 26  
 availableTransports () (in module *pyxcp.transport.base*), 14

## B

BaseTransport (class in *pyxcp.transport.base*), 13  
 block\_receive () (*pyxcp.transport.base.BaseTransport* method), 14  
 block\_request () (*pyxcp.transport.base.BaseTransport* method), 14  
 BUILD\_CHECKSUM (*pyxcp.types.Command* attribute), 23

## C

CAL\_PAG (*pyxcp.types.CommandCategory* attribute), 25  
 calculateFilter () (in module *pyxcp.transport.can*), 16  
 Can (class in *pyxcp.transport.can*), 15  
 CanInterfaceBase (class in *pyxcp.transport.can*), 15  
 check () (in module *pyxcp.checksum*), 20  
 CLEAR\_DAQ\_LIST (*pyxcp.types.Command* attribute), 23  
 close () (*pyxcp.transport.base.BaseTransport* method), 14  
 close () (*pyxcp.transport.can.Can* method), 15  
 close () (*pyxcp.transport.can.CanInterfaceBase* method), 15  
 close () (*pyxcp.transport.eth.Eth* method), 18  
 closeConnection () (*pyxcp.transport.base.BaseTransport* method), 14  
 closeConnection () (*pyxcp.transport.can.Can* method), 15  
 closeConnection () (*pyxcp.transport.eth.Eth* method), 18  
 closeConnection () (*pyxcp.transport.sxi.SxI* method), 18  
 Command (class in *pyxcp.types*), 23  
 CommandCategory (class in *pyxcp.types*), 25  
 Configuration (class in *pyxcp.config*), 20  
 CONNECT (*pyxcp.types.Command* attribute), 23  
 connect () (*pyxcp.transport.base.BaseTransport* method), 14  
 connect () (*pyxcp.transport.can.Can* method), 15  
 connect () (*pyxcp.transport.can.CanInterfaceBase* method), 15  
 connect () (*pyxcp.transport.eth.Eth* method), 18  
 connect () (*pyxcp.transport.sxi.SxI* method), 18  
 ConnectResponse (in module *pyxcp.utils*), 27  
 COPY\_CAL\_PAGE (*pyxcp.types.Command* attribute), 23  
 Crc16 (class in *pyxcp.checksum*), 19

CRC32 () (in module *pyxcp.checksum*), 19  
 createTransport () (in module *pyxcp.transport.base*), 14  
 critical () (*pyxcp.logger.Logger* method), 22

## D

DAQ (*pyxcp.types.CommandCategory* attribute), 25  
 daqListNumber (*pyxcp.types.DaqPtr* attribute), 25  
 DaqPtr (class in *pyxcp.types*), 25  
 dataReceived () (*pyxcp.transport.can.Can* method), 15  
 DBG\_ATTACH (*pyxcp.types.Command* attribute), 23  
 DBG\_EXCLUSIVE\_TARGET\_ACCESS (*pyxcp.types.Command* attribute), 23  
 DBG\_GET\_HWIO\_INFO (*pyxcp.types.Command* attribute), 23  
 DBG\_GET\_JTAG\_ID (*pyxcp.types.Command* attribute), 23  
 DBG\_GET\_MODE\_INFO (*pyxcp.types.Command* attribute), 23  
 DBG\_GET\_TRI\_DESC\_TBL (*pyxcp.types.Command* attribute), 23  
 DBG\_GET\_VENDOR\_INFO (*pyxcp.types.Command* attribute), 23  
 DBG\_HALT\_AFTER\_RESET (*pyxcp.types.Command* attribute), 23  
 DBG\_HWIO\_CONTROL (*pyxcp.types.Command* attribute), 23  
 DBG\_LLBT (*pyxcp.types.Command* attribute), 23  
 DBG\_LLT (*pyxcp.types.Command* attribute), 23  
 DBG\_READ (*pyxcp.types.Command* attribute), 23  
 DBG\_READ\_CAN1 (*pyxcp.types.Command* attribute), 23  
 DBG\_READ\_CAN2 (*pyxcp.types.Command* attribute), 23  
 DBG\_READ\_MODIFY\_WRITE (*pyxcp.types.Command* attribute), 23  
 DBG\_SEQUENCE\_MULTIPLE (*pyxcp.types.Command* attribute), 23  
 DBG\_SET\_HWIO\_EVENT (*pyxcp.types.Command* attribute), 23  
 DBG\_WRITE (*pyxcp.types.Command* attribute), 23  
 DBG\_WRITE\_CAN1 (*pyxcp.types.Command* attribute), 23  
 DBG\_WRITE\_CAN2 (*pyxcp.types.Command* attribute), 23  
 DBG\_WRITE\_CAN\_NEXT (*pyxcp.types.Command* attribute), 23  
 DBG\_WRITE\_NEXT (*pyxcp.types.Command* attribute), 23  
 debug () (*pyxcp.logger.Logger* method), 22  
 decode () (*pyxcp.asam.types.AsamBaseType* method), 12  
 delay () (in module *pyxcp.utils*), 27  
 DISCONNECT (*pyxcp.types.Command* attribute), 23

DISPLAY\_ERROR (*pyxcp.errormatrix.Action* attribute), 21  
 DISPLAY\_ERROR (*pyxcp.errormatrix.PreAction* attribute), 21  
 DOWNLOAD (*pyxcp.errormatrix.PreAction* attribute), 21  
 DOWNLOAD (*pyxcp.types.Command* attribute), 23  
 DOWNLOAD\_MAX (*pyxcp.types.Command* attribute), 23  
 DOWNLOAD\_NEXT (*pyxcp.types.Command* attribute), 23  
 DTO\_CTR\_PROPERTIES (*pyxcp.types.Command* attribute), 24

## E

ECU (*pyxcp.types.XcpGetIdType* attribute), 26  
 Empty, 14  
 EmptyHeader (class in *pyxcp.transport.can*), 15  
 encode () (*pyxcp.asam.types.AsamBaseType* method), 12  
 EPK (*pyxcp.types.XcpGetIdType* attribute), 26  
 error () (*pyxcp.logger.Logger* method), 22  
 Eth (class in *pyxcp.transport.eth*), 17  
 EV\_CLEAR\_DAQ (*pyxcp.types.Event* attribute), 25  
 EV\_CMD\_PENDING (*pyxcp.types.Event* attribute), 25  
 EV\_DAQ\_OVERLOAD (*pyxcp.types.Event* attribute), 25  
 EV\_RESUME\_MODE (*pyxcp.types.Event* attribute), 25  
 EV\_SESSION\_TERMINATED (*pyxcp.types.Event* attribute), 26  
 EV\_SLEEP (*pyxcp.types.Event* attribute), 26  
 EV\_STIM\_TIMEOUT (*pyxcp.types.Event* attribute), 26  
 EV\_STORE\_CAL (*pyxcp.types.Event* attribute), 26  
 EV\_STORE\_DAQ (*pyxcp.types.Event* attribute), 26  
 EV\_TIME\_SYNC (*pyxcp.types.Event* attribute), 26  
 EV\_TRANSPORT (*pyxcp.types.Event* attribute), 26  
 EV\_USER (*pyxcp.types.Event* attribute), 26  
 EV\_WAKE\_UP (*pyxcp.types.Event* attribute), 26  
 Event (class in *pyxcp.types*), 25  
 ext (*pyxcp.types.MtaType* attribute), 26

## F

FILE\_AND\_PATH (*pyxcp.types.XcpGetIdType* attribute), 26  
 FILE\_TO\_UPLOAD (*pyxcp.types.XcpGetIdType* attribute), 26  
 FILENAME (*pyxcp.types.XcpGetIdType* attribute), 26  
 finishListener () (*pyxcp.transport.base.BaseTransport* method), 14  
 FIRST\_PART (*pyxcp.types.XcpGetSeedMode* attribute), 26  
 flatten () (in module *pyxcp.utils*), 27  
 flush () (*pyxcp.transport.sxi.Sxl* method), 18  
 FMT (*pyxcp.asam.types.A\_Float32* attribute), 11  
 FMT (*pyxcp.asam.types.A\_Float64* attribute), 11  
 FMT (*pyxcp.asam.types.A\_Int16* attribute), 11  
 FMT (*pyxcp.asam.types.A\_Int32* attribute), 12  
 FMT (*pyxcp.asam.types.A\_Int64* attribute), 12

FMT (*pyxcp.asam.types.A\_Int8 attribute*), 12  
 FMT (*pyxcp.asam.types.A\_Uint16 attribute*), 12  
 FMT (*pyxcp.asam.types.A\_Uint32 attribute*), 12  
 FMT (*pyxcp.asam.types.A\_Uint64 attribute*), 12  
 FMT (*pyxcp.asam.types.A\_Uint8 attribute*), 12  
 FORMAT (*pyxcp.logger.Logger attribute*), 22  
 Frame (*class in pyxcp.transport.can*), 16  
 FrameSizeError, 26  
 FREE\_DAQ (*pyxcp.types.Command attribute*), 24

## G

get () (*in module pyxcp.transport.base*), 14  
 get () (*pyxcp.config.Configuration method*), 20  
 GET\_CAL\_PAGE (*pyxcp.types.Command attribute*), 24  
 GET\_COMM\_MODE\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_CLOCK (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_EVENT\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_LIST\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_LIST\_MODE (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_PACKED\_MODE (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_PROCESSOR\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_DAQ\_RESOLUTION\_INFO (*pyxcp.types.Command attribute*), 24  
 get\_error\_code () (*pyxcp.types.XcpResponseError method*), 27  
 GET\_ID (*pyxcp.types.Command attribute*), 24  
 GET\_PAG\_PROCESSOR\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_PAGE\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_PGM\_PROCESSOR\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_SECTOR\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_SEED (*pyxcp.types.Command attribute*), 24  
 GET\_SEED\_UNLOCK (*pyxcp.errormatrix.PreAction attribute*), 21  
 GET\_SEGMENT\_INFO (*pyxcp.types.Command attribute*), 24  
 GET\_SEGMENT\_MODE (*pyxcp.types.Command attribute*), 24  
 GET\_STATUS (*pyxcp.types.Command attribute*), 24  
 GET\_VERSION (*pyxcp.types.Command attribute*), 24  
 getLastError () (*pyxcp.logger.Logger method*), 22  
 getPythonVersion () (*in module pyxcp.utils*), 27  
 getTimestampResolution () (*pyxcp.transport.can.CanInterfaceBase method*), 15

## H

Handler (*class in pyxcp.errormatrix*), 21  
 HEADER (*pyxcp.transport.can.Can attribute*), 15  
 HEADER (*pyxcp.transport.eth.Eth attribute*), 17  
 HEADER (*pyxcp.transport.sxi.SxI attribute*), 18  
 HEADER\_SIZE (*pyxcp.transport.can.Can attribute*), 15  
 HEADER\_SIZE (*pyxcp.transport.eth.Eth attribute*), 18  
 HEADER\_SIZE (*pyxcp.transport.sxi.SxI attribute*), 18  
 hexDump () (*in module pyxcp.utils*), 27

## I

id (*pyxcp.transport.can.Identifier attribute*), 16  
 Identifier (*class in pyxcp.transport.can*), 16  
 IdentifierOutOfRangeException, 16  
 info () (*pyxcp.logger.Logger method*), 22  
 init () (*pyxcp.transport.can.CanInterfaceBase method*), 15  
 invalidSocket (*pyxcp.transport.eth.Eth attribute*), 18  
 is\_extended (*pyxcp.transport.can.Identifier attribute*), 16  
 isExtendedIdentifier () (*in module pyxcp.transport.can*), 17

## L

listen () (*pyxcp.transport.base.BaseTransport method*), 14  
 listen () (*pyxcp.transport.can.Can method*), 15  
 listen () (*pyxcp.transport.eth.Eth method*), 18  
 listen () (*pyxcp.transport.sxi.SxI method*), 18  
 loadConfig () (*pyxcp.transport.base.BaseTransport method*), 14  
 loadConfig () (*pyxcp.transport.can.CanInterfaceBase method*), 15  
 log () (*pyxcp.logger.Logger method*), 22  
 Logger (*class in pyxcp.logger*), 22  
 LOGGER\_BASE\_NAME (*pyxcp.logger.Logger attribute*), 22

## M

make\_identifier () (*pyxcp.transport.can.Identifier static method*), 16  
 MAX\_DATAGRAM\_SIZE (*pyxcp.transport.can.Can attribute*), 15  
 MAX\_DATAGRAM\_SIZE (*pyxcp.transport.eth.Eth attribute*), 18  
 MAX\_DATAGRAM\_SIZE (*pyxcp.transport.sxi.SxI attribute*), 18  
 MODIFY\_BITS (*pyxcp.types.Command attribute*), 24  
 MOTOROLA (*in module pyxcp.asam.types*), 13  
 MtaType (*class in pyxcp.types*), 26

## N

NEW\_FLASH\_WARE (*pyxcp.errormatrix.Action attribute*), 21  
 NONE (*pyxcp.errormatrix.Action attribute*), 21  
 NONE (*pyxcp.errormatrix.PreAction attribute*), 21

## O

odtEntryNumber (*pyxcp.types.DaqPtr attribute*), 25  
 odtNumber (*pyxcp.types.DaqPtr attribute*), 25  
 output () (*pyxcp.transport.sxi.SxI method*), 18

## P

pack () (*pyxcp.transport.can.EmptyHeader method*), 16  
 padFrame () (*in module pyxcp.transport.can*), 17  
 PARAMETER\_MAP (*pyxcp.transport.base.BaseTransport attribute*), 14  
 PARAMETER\_MAP (*pyxcp.transport.can.Can attribute*), 15  
 PARAMETER\_MAP (*pyxcp.transport.can.CanInterfaceBase attribute*), 15  
 PARAMETER\_MAP (*pyxcp.transport.eth.Eth attribute*), 18  
 PARAMETER\_MAP (*pyxcp.transport.sxi.SxI attribute*), 18  
 PARAMETER\_TO\_KW\_ARG\_MAP (*pyxcp.transport.can.Can attribute*), 15  
 PGM (*pyxcp.types.CommandCategory attribute*), 25  
 PreAction (*class in pyxcp.errormatrix*), 21  
 preAction (*pyxcp.errormatrix.Handler attribute*), 21  
 process\_event\_packet () (*pyxcp.transport.base.BaseTransport method*), 14  
 processResponse () (*pyxcp.transport.base.BaseTransport method*), 14  
 PROGRAM (*pyxcp.errormatrix.PreAction attribute*), 21  
 PROGRAM (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_CLEAR (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_FORMAT (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_MAX (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_NEXT (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_PREPARE (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_RESET (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_START (*pyxcp.types.Command attribute*), 24  
 PROGRAM\_VERIFY (*pyxcp.types.Command attribute*), 24  
 pyxcp (*module*), 27  
 pyxcp.asam (*module*), 13  
 pyxcp.asam.types (*module*), 11  
 pyxcp.checksum (*module*), 18  
 pyxcp.config (*module*), 20  
 pyxcp.errormatrix (*module*), 21

pyxcp.logger (*module*), 22  
 pyxcp.master (*module*), 13  
 pyxcp.transport (*module*), 18  
 pyxcp.transport.base (*module*), 13  
 pyxcp.transport.can (*module*), 15  
 pyxcp.transport.eth (*module*), 17  
 pyxcp.transport.sxi (*module*), 18  
 pyxcp.types (*module*), 23  
 pyxcp.utils (*module*), 27

## R

raw\_id (*pyxcp.transport.can.Identifier attribute*), 16  
 read () (*pyxcp.transport.can.CanInterfaceBase method*), 15  
 READ\_DAQ (*pyxcp.types.Command attribute*), 24  
 readConfiguration () (*in module pyxcp.config*), 20  
 reflect () (*in module pyxcp.checksum*), 20  
 reflectIn () (*pyxcp.checksum.Crc16 method*), 19  
 reflectOut () (*pyxcp.checksum.Crc16 method*), 19  
 registered\_drivers () (*in module pyxcp.transport.can*), 17  
 REINIT\_DAQ (*pyxcp.errormatrix.PreAction attribute*), 21  
 REMAINING (*pyxcp.types.XcpGetSeedMode attribute*), 26  
 REPEAT (*pyxcp.errormatrix.Action attribute*), 21  
 REPEAT\_2\_TIMES (*pyxcp.errormatrix.Action attribute*), 21  
 REPEAT\_INF\_TIMES (*pyxcp.errormatrix.Action attribute*), 21  
 request () (*pyxcp.transport.base.BaseTransport method*), 14  
 request\_optional\_response () (*pyxcp.transport.base.BaseTransport method*), 14  
 RESTART\_SESSION (*pyxcp.errormatrix.Action attribute*), 21  
 RETRY\_PARAM (*pyxcp.errormatrix.Action attribute*), 21  
 RETRY\_SYNTAX (*pyxcp.errormatrix.Action attribute*), 21

## S

S0 (*pyxcp.errormatrix.Severity attribute*), 22  
 S1 (*pyxcp.errormatrix.Severity attribute*), 22  
 S2 (*pyxcp.errormatrix.Severity attribute*), 22  
 S3 (*pyxcp.errormatrix.Severity attribute*), 22  
 samplePointToTsegs () (*in module pyxcp.transport.can*), 17  
 send () (*pyxcp.transport.base.BaseTransport method*), 14  
 send () (*pyxcp.transport.can.Can method*), 15  
 send () (*pyxcp.transport.eth.Eth method*), 18  
 send () (*pyxcp.transport.sxi.SxI method*), 18  
 SET\_CAL\_PAGE (*pyxcp.types.Command attribute*), 24



- SET\_DAQ\_LIST\_MODE (*pyxcp.types.Command attribute*), 24
- SET\_DAQ\_PACKED\_MODE (*pyxcp.types.Command attribute*), 24
- SET\_DAQ\_PTR (*pyxcp.errormatrix.PreAction attribute*), 21
- SET\_DAQ\_PTR (*pyxcp.types.Command attribute*), 24
- SET\_MTA (*pyxcp.errormatrix.PreAction attribute*), 21
- SET\_MTA (*pyxcp.types.Command attribute*), 25
- SET\_REQUEST (*pyxcp.types.Command attribute*), 25
- SET\_SEGMENT\_MODE (*pyxcp.types.Command attribute*), 25
- setDLC () (*in module pyxcp.transport.can*), 17
- setLevel () (*pyxcp.logger.Logger method*), 22
- Severity (*class in pyxcp.errormatrix*), 22
- SHORT\_DOWNLOAD (*pyxcp.types.Command attribute*), 25
- SHORT\_UPLOAD (*pyxcp.types.Command attribute*), 25
- silent () (*pyxcp.logger.Logger method*), 22
- SKIP (*pyxcp.errormatrix.Action attribute*), 21
- slicer () (*in module pyxcp.utils*), 27
- START\_STOP\_DAQ\_LIST (*pyxcp.types.Command attribute*), 25
- START\_STOP\_SYNCH (*pyxcp.types.Command attribute*), 25
- START\_STOP\_X (*pyxcp.errormatrix.PreAction attribute*), 21
- startListener () (*pyxcp.transport.base.BaseTransport method*), 14
- startListener () (*pyxcp.transport.eth.Eth method*), 18
- STD (*pyxcp.types.CommandCategory attribute*), 25
- STIM (*pyxcp.types.CommandCategory attribute*), 25
- stripIdentifier () (*in module pyxcp.transport.can*), 17
- SxI (*class in pyxcp.transport.sxi*), 18
- SYNCH (*pyxcp.errormatrix.PreAction attribute*), 21
- SYNCH (*pyxcp.types.Command attribute*), 25
- SYSID (*pyxcp.types.XcpGetIdType attribute*), 26
- T**
- T1 (*pyxcp.errormatrix.Timeout attribute*), 22
- T2 (*pyxcp.errormatrix.Timeout attribute*), 22
- T3 (*pyxcp.errormatrix.Timeout attribute*), 22
- T4 (*pyxcp.errormatrix.Timeout attribute*), 22
- T5 (*pyxcp.errormatrix.Timeout attribute*), 22
- T6 (*pyxcp.errormatrix.Timeout attribute*), 22
- T7 (*pyxcp.errormatrix.Timeout attribute*), 22
- TERMINATE\_SESSION (*pyxcp.errormatrix.Action attribute*), 21
- TIME\_CORRELATION\_PROPERTIES (*pyxcp.types.Command attribute*), 25
- time\_perfcounter\_correlation () (*in module pyxcp.utils*), 27
- Timeout (*class in pyxcp.errormatrix*), 22
- TIMEOUT (*pyxcp.transport.sxi.SxI attribute*), 18
- transmit () (*pyxcp.transport.can.CanInterfaceBase method*), 15
- TRANSPORT\_LAYER\_CMD (*pyxcp.types.Command attribute*), 25
- try\_to\_install\_system\_supplied\_drivers () (*in module pyxcp.transport.can*), 17
- U**
- UNLOCK (*pyxcp.types.Command attribute*), 25
- UNLOCK\_SLAVE (*pyxcp.errormatrix.PreAction attribute*), 21
- UPLOAD (*pyxcp.errormatrix.PreAction attribute*), 22
- UPLOAD (*pyxcp.types.Command attribute*), 25
- URL (*pyxcp.types.XcpGetIdType attribute*), 26
- USE\_A2L (*pyxcp.errormatrix.Action attribute*), 21
- USE\_ALTERNATIVE (*pyxcp.errormatrix.Action attribute*), 21
- USER\_CMD (*pyxcp.types.Command attribute*), 25
- userDefined () (*in module pyxcp.checksum*), 20
- V**
- VECTOR\_MAPNAMES (*pyxcp.types.XcpGetIdType attribute*), 26
- VECTOR\_MDI (*pyxcp.types.XcpGetIdType attribute*), 26
- verbose () (*pyxcp.logger.Logger method*), 22
- W**
- WAIT\_T7 (*pyxcp.errormatrix.PreAction attribute*), 22
- warn () (*pyxcp.logger.Logger method*), 22
- WIDTH (*pyxcp.checksum.Crc16 attribute*), 19
- wordSum () (*in module pyxcp.checksum*), 20
- WRITE\_DAQ (*pyxcp.types.Command attribute*), 25
- WRITE\_DAQ\_MULTIPLE (*pyxcp.types.Command attribute*), 25
- X**
- XCP\_ADD\_11 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_ADD\_12 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_ADD\_14 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_ADD\_22 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_ADD\_24 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_ADD\_44 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_CRC\_16 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_CRC\_16\_CITT (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_CRC\_32 (*pyxcp.checksum.Algorithm attribute*), 19
- XCP\_USER\_DEFINED (*pyxcp.checksum.Algorithm attribute*), 19
- XcpGetIdType (*class in pyxcp.types*), 26
- XcpGetSeedMode (*class in pyxcp.types*), 26
- XcpResponseError, 26
- XcpTimeoutError, 27