canlib

Release 1.22.565

Kvaser AB <support@kvaser.com>

Sep 13, 2022

CONTENTS

1 Contents **3**
- 1.1 Welcome . . . 3
- 1.2 Supported Libraries and Installation . . . 3
- 1.3 Tutorials . . . 6
- 1.4 Using canlib (CANlib) . . . 14
- 1.5 Examples . . . 34
- 1.6 Reference . . . 65

2 Release Notes **241**
- 2.1 Release Notes . . . 241

Python Module Index **253**

Index **255**

The canlib module is a Python wrapper for Kvaser CANlib SDK.

"The CANlib Software Development Kit is your Application Programming Interface for working with all Kvaser hardware platforms."

Using the Python canlib package, you will be able to control most aspects of any Kvaser CAN interface via Python.

CHAPTER

ONE

CONTENTS

1.1 Welcome

The canlib package is a Python wrapper for Kvaser CANlib SDK.

"The CANlib Software Development Kit is your Application Programming Interface for working with all Kvaser hardware platforms."

Using the Python canlib package, you will be able to control most aspects of any Kvaser CAN interface via Python.

canlib - a Python wrapper for Kvaser CANlib

1.2 Supported Libraries and Installation

The Python canlib module wraps the Kvaser CANlib API in order to make it easy to control most aspects of any Kvaser CAN interface. For more information about Kvaser please go to https://www.kvaser.com/.

The latest version of this package is available on the Kvaser Download page (pycanlib.zip).

1.2.1 Supported platforms

Windows and Linux using Python v3.6+ (both 32 and 64 bit).

1.2.2 Supported libraries

The following libraries are currently supported:

Library	Module	Windows	Linux
CANlib	canlib.canlib	canlib32.dll	libcanlib.so
kvaMemoLibXML	canlib.kvamemolibxml	kvaMemoLibXML.dll	libkvamemolibxml.so
kvrlib	canlib.kvrlib	kvrlib.dll, irisdll.dll, irisflash.dll, libxml2.dll	not supported
kvmlib	canlib.kvmlib	kvaMemoLib0600.dll, kvaMemoLib0700.dll, kvaMemoLib.dll, kvmlib.dll	not supported,[2] libkvamemolib0700.so, libkvamemolib.so, libkvmlib.so
kvlclib	canlib.kvlclib	kvlclib.dll[1]	libkvlclib.so
kvaDbLib	canlib.kvadblib	kvaDbLib.dll	libkvadblib.so
LINlib	canlib.linlib	linlib.dll	liblinlib.so

1.2.3 Installation

Install the Python package from PyPI using e.g. `pip`:

```
$ pip install canlib
```

If you have downloaded the package zip file from the Kvaser Download page, first unzip `pycanlib.zip`. Then navigate to the unzipped pycanlib using the command line. It should contain the file `canlib-x.y.z-py2.py3-none-any.whl`, where x,y,z are version numbers. Run the following command:

```
$ pip install canlib-x.y.z-py2.py3-none-any.whl
```

The Kvaser CANlib DLLs or shared libraries also need to be installed:

Windows

On **Windows**, first install the `canlib32.dll` by downloading and installing "Kvaser Drivers for Windows" which can be found on the Kvaser Download page (kvaser_drivers_setup.exe) This will also install `kvrlib.dll`, `irisdll.dll`, `irisflash.dll` and `libxml2.dll` used by kvrlib.

The "Kvaser CANlib SDK" also needs to be downloaded from the same place (canlib.exe) and installed if more than just CANlib will be used. This will install the rest of the supported library dll's.

The two packages, "Kvaser Drivers for Windows" and "Kvaser CANlib SDK", contains both 32 and 64 bit versions of the included dll's.

[2] The kvaMemoLib0600.dll, which supports older devices, is not supported under Linux.
[1] The kvlclib.dll depends on dll files from matlab wich are installed alongside kvlclib.dll.

Linux

On **Linux**, first install the `libcanlib.so` by downloading and installing "Kvaser LINUX Driver and SDK" which can be found on the Kvaser Download page (linuxcan.tar.gz).

If more than just CANlib will be used, the rest of the supported libraries will be available by downloading and installing "Linux SDK library" (kvlibsdk.tar.gz).

1.2.4 Usage

Example of using canlib to list some information about connected Kvaser devices:

```
from canlib import canlib

num_channels = canlib.getNumberOfChannels()
print(f"Found {num_channels} channels")
for ch in range(num_channels):
    chd = canlib.ChannelData(ch)
    print(f"{ch}. {chd.channel_name} ({chd.card_upc_no} / {chd.card_serial_no})")
```

Which may result in:

```
Found 4 channels
0. Kvaser Memorator Pro 2xHS v2 (channel 0) (73-30130-00819-9 / 12330)
1. Kvaser Memorator Pro 2xHS v2 (channel 1) (73-30130-00819-9 / 12330)
2. Kvaser Virtual CAN Driver (channel 0) (00-00000-00000-0 / 0)
3. Kvaser Virtual CAN Driver (channel 1) (00-00000-00000-0 / 0)
```

1.2.5 Support

You are invited to visit the Kvaser web pages at https://www.kvaser.com/support/. If you don't find what you are looking for, you can obtain support on a time-available basis by sending an e-mail to support@kvaser.com.

Bug reports, contributions, suggestions for improvements, and similar things are much appreciated and can be sent by e-mail to support@kvaser.com.

1.2.6 What's new

A complete set of release notes are available in the package documentation included in the zip file available at the Kvaser Download page.

1.2.7 Links

- Kvaser CANlib SDK Page: https://www.kvaser.com/developer/canlib-sdk/
- Description of CANlib SDK library contents: https://www.kvaser.com/developer-blog/get-hardware-kvaser-sdk-libraries/

1.2.8 License

This project is licensed under the terms of the MIT license.

1.2.9 Wrapped libraries

- *canlib*
- *kvadblib*
- *kvamemolibxml*
- *kvlclib*
- *kvmlib*
- *kvrlib*
- *linlib*

1.3 Tutorials

1.3.1 canlib

Contents

- *canlib*
  - *List connected devices*
  - *Send and receive single frame*
  - *Send and receive CAN FD frame*

The following sections contain sample code for inspiration on how to use Kvaser Python canlib.

List connected devices

```
"""list_devices.py -- List all connected CAN devices

This code probes each connected device and prints information about them.

"""

import canlib


for dev in canlib.connected_devices():
    print(dev.probe_info())
```

Sample Output:

```
CANlib Channel: 0
Card Number   : 7
Device        : Kvaser Memorator Pro 2xHS v2 (channel 0)
Driver Name   : kcany7a
EAN           : 73-30130-00819-9
Firmware      : 3.24.0.722
Serial Number : 12330
CANlib Channel: 2
Card Number   : 8
Device        : Kvaser Memorator Pro 5xHS (channel 0)
Driver Name   : kcany8a
EAN           : 73-30130-00832-8
Firmware      : 3.23.0.646
Serial Number : 10028
CANlib Channel: 7
Card Number   : 0
Device        : Kvaser Virtual CAN Driver (channel 0)
Driver Name   : kcanv0a
EAN           : 00-00000-00000-0
Firmware      : 0.0.0.0
Serial Number : 0
```

Send and receive single frame

```
"""send_and_receive_can.py

Here is some basic code to send and receive a single frame.

"""


from canlib import canlib, Frame
from canlib.canlib import ChannelData


def setUpChannel(channel=0,
                 openFlags=canlib.Open.ACCEPT_VIRTUAL,
                 outputControl=canlib.Driver.NORMAL):
    ch = canlib.openChannel(channel, openFlags)
    print("Using channel: %s, EAN: %s" % (ChannelData(channel).channel_name,
                                          ChannelData(channel).card_upc_no))
    ch.setBusOutputControl(outputControl)
    # Specifying a bus speed of 250 kbit/s. See documentation
    # for more informationon how to set bus parameters.
    params = canlib.busparams.BusParamsTq(
        tq=8,
        phase1=2,
        phase2=2,
        sjw=1,
        prescaler=40,
        prop=3
```

(continues on next page)

(continued from previous page)

```
    )
    ch.set_bus_params_tq(params)
    ch.busOn()
    return ch


def tearDownChannel(ch):
    ch.busOff()
    ch.close()


print("canlib version:", canlib.dllversion())
ch0 = setUpChannel(channel=0)
ch1 = setUpChannel(channel=1)
frame = Frame(
    id_=100,
    data=[1, 2, 3, 4],
    flags=canlib.MessageFlag.EXT
)
ch1.write(frame)
while True:
    try:
        frame = ch0.read()
        print(frame)
        break
    except canlib.canNoMsg:
        pass
    except canlib.canError as ex:
        print(ex)
tearDownChannel(ch0)
tearDownChannel(ch1)
```

Send and receive CAN FD frame

```
"""send_and_receive_canfd.py

Here are some minimal code to send and receive a CAN FD frame.

"""

from canlib import canlib, Frame

# Specifying an arbitration phase bus speed of 1 Mbit/s,
# and a data phase bus speed of 2 Mbit/s. See documentation
# for more information on how to set bus parameters.
params_arbitration = canlib.busparams.BusParamsTq(
    tq=40,
    phase1=8,
    phase2=8,
    sjw=8,
```

(continues on next page)

(continued from previous page)

```
    prescaler=2,
    prop=23
)
params_data = canlib.busparams.BusParamsTq(
    tq=20,
    phase1=8,
    phase2=4,
    sjw=4,
    prescaler=2,
    prop=7
)

# open channel as CAN FD using the flag
ch0 = canlib.openChannel(channel=0, flags=canlib.Open.CAN_FD)
ch0.setBusOutputControl(drivertype=canlib.Driver.NORMAL)
ch0.set_bus_params_tq(params_arbitration, params_data)
ch0.busOn()

ch1 = canlib.openChannel(channel=1, flags=canlib.Open.CAN_FD)
ch1.setBusOutputControl(drivertype=canlib.Driver.NORMAL)
ch1.set_bus_params_tq(params_arbitration, params_data)
ch1.busOn()

# set FDF flag to send using CAN FD
# set BRS flag to send using higher bit rate in the data phase
frame = Frame(
    id_=100,
    data=range(32),
    flags=canlib.MessageFlag.FDF | canlib.MessageFlag.BRS
)
print('Sending', frame)
ch0.write(frame)

frame = ch1.read(timeout=1000)
print('Received', frame)

ch0.busOff()
ch1.busOff()
```

1.3.2 kvrlib

Contents

- *kvrlib*
  - *Connect to your remote device*

The following sections contain sample code for inspiration on how to use Kvaser Python kvrlib.

Connect to your remote device

```
"""connect_to_remote_device.py

Use the discovery functions to scan and connect to a remote device. Our remote
device has serial number 10545 and is already connected to the same network as our
computer.

"""
from canlib import kvrlib

SERIAL_NO = 10545

print("kvrlib version: %s" % kvrlib.getVersion())
print("Connecting to device with serial number %s" % SERIAL_NO)

addressList = kvrlib.kvrDiscovery.getDefaultAddresses(kvrlib.kvrAddressTypeFlag_
↪BROADCAST)
print("Looking for device using addresses: %s" % addressList)
discovery = kvrlib.kvrDiscovery()
discovery.setAddresses(addressList)
deviceInfos = discovery.getResults()
print("Scanning result:\n%s" % deviceInfos)
# Connect to device with correct SERIAL_NO
for deviceInfo in deviceInfos:
    if deviceInfo.ser_no == SERIAL_NO:
        deviceInfo.connect()
        print('\nConnecting to the following device:')
        print('-----------------------------------------------')
        print(deviceInfo)
        discovery.storeDevices(deviceInfos)
        break
discovery.close()
```

This results in the following:

```
kvrlib version: 2070
Connecting to device with serial number 10545
Looking for device using addresses: 10.0.255.255:0 (IPV4_PORT)
Scanning result:
[
name/hostname  : "MiMi-06348-000710" / "kv-06348-000710"
  ean/serial   : 73301-30006348 / 710
  fw           : 2.4.231
  addr/cli/AP  : 10.0.3.138 (IPV4) / 10.0.3.84 (IPV4) / - (UNKNOWN)
  availability : Availability.STORED|FOUND_BY_SCAN
  usage/access : DeviceUsage.FREE / Accessibility.PUBLIC
  pass/enc.key : yes / yes,
name/hostname  : "TestClient1-2-DUT-01" / "swtdut01"
  ean/serial   : 73301-30006713 / 10545
  fw           : 3.4.822
  addr/cli/AP  : 10.0.3.54 (IPV4) / 10.0.3.98 (IPV4) / - (UNKNOWN)
  availability : Availability.STORED|FOUND_BY_SCAN
```

(continues on next page)

(continued from previous page)

```
  usage/access : DeviceUsage.REMOTE / Accessibility.PUBLIC
  pass/enc.key : yes / yes]

Connecting to the following device:
-------------------------------------------

name/hostname  : "TestClient1-2-DUT-01" / "swtdut01"
  ean/serial   : 73301-30006713 / 10545
  fw           : 3.4.822
  addr/cli/AP  : 10.0.3.54 (IPV4) / 10.0.3.98 (IPV4) / - (UNKNOWN)
  availability : Availability.STORED|FOUND_BY_SCAN
  usage/access : DeviceUsage.REMOTE / Accessibility.PUBLIC
  pass/enc.key : yes / yes
```

1.3.3 linlib

The following sections contain sample code for inspiration on how to use Kvaser Python linlib.

Basic master slave usage

```
"""basic_master_slave_lin.py

This code opens up one master and one slave, sets bitrate and then the slave
sends a wakeup message to the master.

"""

# import the linlib wrapper from the canlib package
from canlib import linlib

# print information about device firmware version
print(linlib.getChannelData(channel_number=0,
                            item=linlib.ChannelData.CARD_FIRMWARE_REV))

# open the first channel as a Master
master = linlib.openChannel(channel_number=0,
                            channel_type=linlib.ChannelType.MASTER)

# open the next channel as a Slave
slave = linlib.openChannel(channel_number=1,
                           channel_type=linlib.ChannelType.SLAVE)

# setup bitrate
master.setBitrate(10000)
slave.setBitrate(10000)

# activate the LIN interface by going bus on
master.busOn()
slave.busOn()
```

(continues on next page)

(continued from previous page)

```
# send a wakeup frame from the slave
slave.writeWakeup()

# read the frame when it arrives at the master
frame = master.read(timeout=100)
print(frame)

# go bus off
master.busOff()
slave.busOff()
```

Sending message from master

```
"""send_lin_message.py

This example uses two shorthand helper functions to open the channels. We then
send some messages from the master and see that they arrive.

"""
from canlib import linlib, Frame

# open the first channel as Master, using helper function
master = linlib.openMaster(0)

# open the next channel as a Slave, using helper function
slave = linlib.openSlave(1)

# go bus on
master.busOn()
slave.busOn()

# send some messages from master
NUM_MESSAGES = 2
for i in range(NUM_MESSAGES):
    master.writeMessage(Frame(id_=i, data=[1, 2, 3, 4, 5, 6, 7, 8]))
master.writeSync(100)

# print the received messages at the slave
for i in range(NUM_MESSAGES):
    frame = slave.read(timeout=100)
    print(frame)

# the master should also have recorded the messages
for i in range(NUM_MESSAGES):
    frame = master.read(timeout=100)
    print(frame)

# go bus off
master.busOff()
slave.busOff()
```

Requesting LIN 2.0 message

```
"""request_lin_message.py

Here we look at using LIN 2.0 and setting up a message, using the `Frame`
object, on the slave which is then requested by the master.

"""
from canlib import linlib, Frame


ID = 0x17
DATA = bytearray([1, 2, 3, 4])

# open the first channel as Master, using helper function
master = linlib.openMaster(0, bps=20000)

# open the next channel as a Slave, using helper function
slave = linlib.openSlave(1)

master.busOn()
slave.busOn()

# configure channels to use LIN 2.0
slave.setupLIN(flags=linlib.Setup.ENHANCED_CHECKSUM | linlib.Setup.VARIABLE_DLC)
master.setupLIN(flags=linlib.Setup.ENHANCED_CHECKSUM | linlib.Setup.VARIABLE_DLC)

# setup a message in the slave
slave.updateMessage(Frame(id_=ID, data=DATA))

# request the message and print it
master.requestMessage(ID)
frame = master.read(timeout=100)
print(frame)

# clear the message
slave.clearMessage(0x17)

# we should now get an empty message
master.requestMessage(0x17)
frame = master.read(timeout=100)
print(frame)

# go bus off
master.busOff()
slave.busOff()
```

1.4 Using canlib (CANlib)

The canlib module wraps the CAN bus API (CANlib), which is used to interact with Kvaser CAN devices connected to your computer and the CAN bus. At its core you have functions to set bus parameters (e.g. bit rate), go bus on/off and read/write CAN messages. You can also use CANlib to download and start t programs on supported devices.

1.4.1 Introduction

Hello, CAN!

Let's start with a simple example:

```
# The CANlib library is initialized when the canlib module is imported.
from canlib import canlib, Frame

# Open a channel to a CAN circuit. In this case we open channel 0 which
# should be the first channel on the CAN interface. EXCLUSIVE means we don't
# want to share this channel with any other currently executing program.
# We also set the CAN bus bit rate to 250 kBit/s, using a set of predefined
# bus parameters.
ch = canlib.openChannel(
    channel=0,
    flags=canlib.Open.EXCLUSIVE,
    bitrate=canlib.Bitrate.BITRATE_250K,
)

# Set the CAN bus driver type to NORMAL.
ch.setBusOutputControl(canlib.Driver.NORMAL)

# Activate the CAN chip.
ch.busOn()

# Transmit a message with (11-bit) CAN id = 123, length 6 and contents
# (decimal) 72, 69, 76, 76, 79, 33.
frame = Frame(id_=123, data=b'HELLO!', dlc=6)
ch.write(frame)

# Wait until the message is sent or at most 500 ms.
ch.writeSync(timeout=500)

# Inactivate the CAN chip.
ch.busOff()

# Close the channel.
ch.close()
```

canlib Core API Calls

The following calls can be considered the "core" of canlib as they are essential for almost any program that uses the CAN bus:

- *openChannel* and *close*
- *busOn* and *busOff*
- *read*
- *write* and *writeSync*

1.4.2 Initialization

Library Initialization

The underlaying CANlib library is initialized when the module `canlib.canlib` is imported. This will initialize the CANlib library and enumerate all currently available CAN channels.

Library Deinitialization and Cleanup

Strictly speaking it is not necessary to clean up anything before terminating the application. If the application quits unexpectedly, the device driver will ensure the CAN controller is deactivated and the driver will also ensure the firmware (if any) is left in a consistent state.

To reinitialize the library in an orderly fashion you may want to call *canlib.Channel.writeSync* with a short timeout for each open handle before closing them with *canlib.Channel.close*, to ensure the transmit queues are empty. You can then start afresh by calling *canlib.reinitializeLibrary*.

Note: When calling *canlib.reinitializeLibrary*, all previously opened CAN handles (*canlib.Channel*) will be closed and invalidated.

Manually Enumerating CAN channels

The function *canlib.enumerate_hardware* scans all currently connected devices and creates a completely new set of CANlib channel numbers, while still keeping all currently opened channel handles valid and usable. This can be viewed upon as a replacement for calling *canlib.reinitializeLibrary* which do invalidate all open channel handles.

One thing to keep in mind when using this functionality is to never track devices based on their CANlib channel number, since this number may change anytime *enumerate_hardware* is called. To retrieve information about a specific channel use *Channel.channel_data* to get a safe *ChannelData*, instead of relying on an old *ChannelData* object created from a channel number.

Note: On Linux, no re-enumeration is needed since enumeration takes place when a device is plugged in or unplugged.

1.4.3 Devices and Channels

Identifying Devices and Channels

Once we have imported `canlib.canlib`, which enumerates the connected Kvaser CAN devices, we can call *getNumberOfChannels* to get the number of enumerated channels in our system.

This code snippet reads the number of enumerated channels found in the PC:

```
>>> from canlib import canlib
>>> canlib.getNumberOfChannels()
8
```

Channel Information

Use *ChannelData* to obtain data for a specific channel, for example, the hardware type of the CAN interface.

We can use *ChannelData* for the CANlib channel numbers 0, 1, 2,..., n-1 (where n is the number returned by *getNumberOfChannels*) to get information about that specific channel.

To uniquely identify a device, we need to look at both the `ChannelData.card_upc_no` and `ChannelData.card_serial_no`.

The following code snippet loops through all known channels and prints the type of the CAN card they're on.

```
>>> from canlib import canlib
...
... num_channels = canlib.getNumberOfChannels()
... print("Found %d channels" % num_channels)
... for channel in range(0, num_channels):
...     chdata = canlib.ChannelData(channel)
...     print("%d. %s (%s / %s)" % (
...         channel,
...         chdata.channel_name,
...         chdata.card_upc_no,
...         chdata.card_serial_no)
...     )
Found 8 channels
0. Kvaser Leaf Light HS (channel 0) (73-30130-00241-8 / 1346)
1. Kvaser Memorator Pro 2xHS v2 (channel 0) (73-30130-00819-9 / 11573)
2. Kvaser Memorator Pro 2xHS v2 (channel 1) (73-30130-00819-9 / 11573)
3. Kvaser Leaf Pro HS v2 (channel 0) (73-30130-00843-4 / 10012)
4. Kvaser Hybrid 2xCAN/LIN (channel 0) (73-30130-00965-3 / 1100)
5. Kvaser Hybrid 2xCAN/LIN (channel 1) (73-30130-00965-3 / 1100)
6. Kvaser Virtual CAN Driver (channel 0) (00-00000-00000-0 / 0)
7. Kvaser Virtual CAN Driver (channel 1) (00-00000-00000-0 / 0)
```

Customized Channel Name

It is possible to set the customized name returned by `ChannelData.card_serial_no` on the device using Kvaser Device Guide by right clicking on the device channel and selecting "Edit Channel Name"

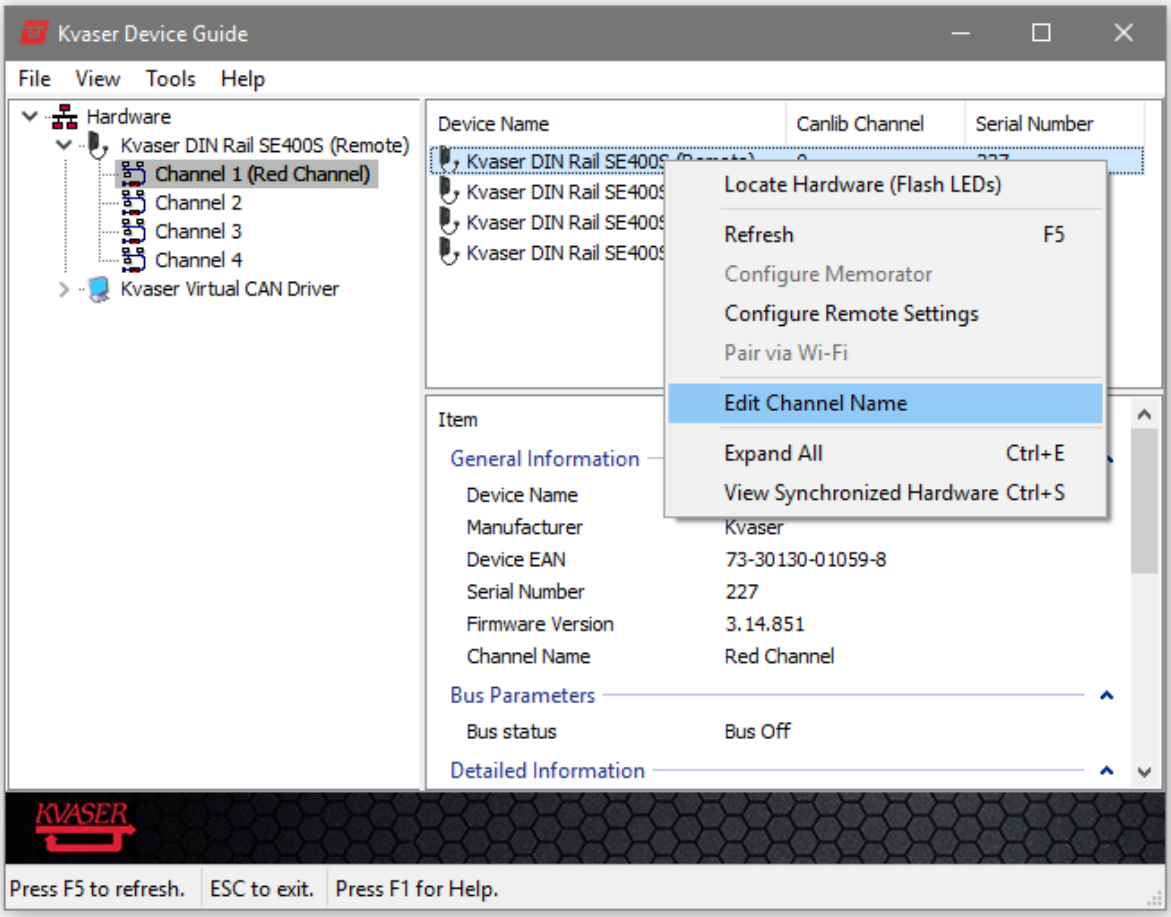

Fig. 1: Setting the device's Channel Name from inside Kvaser Device Guide

Now we can read the customized name:

```
>>> from canlib import canlib
>>> chdata = canlib.ChannelData(channel_number=0)
>>> chdata.custom_name
'Red Channel'
```

Virtual Channels

CANlib supports virtual channels that you can use for development, test or demonstration purposes when you don't have any hardware installed.

To open a virtual channel, call *openChannel* with the appropriate channel number, and specify *ACCEPT_VIRTUAL* in the flags argument to canOpenChannel().

1.4.4 Open Channel

Once we have imported `canlib.canlib` to enumerate the connected Kvaser CAN devices, the next call is likely to be a call to *openChannel*, which returns a *Channel* object for the specific CAN circuit. This object is then used for subsequent calls to the library. The *openChannel* function's first argument is the number of the desired channel, the second argument is modifier flags *Open*.

openChannel may raise several different exceptions, one of which is *CanNotFound*. This means that the channel specified in the first parameter was not found, or that the flags passed to *openChannel* is not applicable to the specified channel.

Open as CAN

No special *Open* modifier flag is needed in the flags argument to *openChannel* when opening a channel in CAN mode.

```
>>> from canlib import canlib
>>> canlib.openChannel(channel=0)
<canlib.canlib.channel.Channel object at 0x0000015B787EDA90>
```

Open as CAN FD

To open a channel in CAN FD mode, either *CAN_FD* or *CAN_FD_NONISO* needs to be given in the flags argument to *openChannel*.

This example opens channel 0 in CAN FD mode:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(
...     channel=0,
...     flags=canlib.Open.CAN_FD,
... )
>>> ch.close()
```

Close Channel

Closing a channel is done using *close*. If no other handles are referencing the same CANlib channel, the channel is taken off bus.

The CAN channel can also be opened and closed using a context manager:

```
>>> from canlib import canlib
>>> with canlib.openChannel(channel=1) as ch:
...     ...
```

Check Channel Capabilities

Channel specific information and capabilities are made available by reading attributes of an instance of type *ChannelData*.

The device clock frequency can be obtained via `frequency()`:

```
>>> from canlib import canlib
>>> chd = canlib.ChannelData(channel_number=0)
>>> clock_info = chd.clock_info
>>> clock_info.frequency()
80000000
```

The capabilities of a channel can be obtained by reading attribute `channel_cap` and `channel_cap_ex`:

```
>>> from canlib import canlib
>>> chd = canlib.ChannelData(channel_number=0)
>>> chd.channel_cap
ChannelCap.IO_API|SCRIPT|LOGGER|SINGLE_SHOT|SILENT_MODE|CAN_FD_NONISO|CAN_FD|
TXACKNOWLEDGE|TXREQUEST|GENERATE_ERROR|ERROR_COUNTERS|BUS_STATISTICS|EXTENDED_CAN
>>> chd.channel_cap_ex[0]
ChannelCapEx.BUSPARAMS_TQ
```

A bitwise AND operator can be used to see if a channel has a specific capability.

```
>>> if (chd.channel_cap & canlib.ChannelCap.CAN_FD):
>>>   print("Channel has support for CAN FD!")
Channel has support for CAN FD!
```

The above printouts are just an example, and will differ for different devices and installed firmware.

Set CAN Bitrate

After opening the channel in classic CAN mode (see *Open as CAN*), use *set_bus_params_tq* to specify the bit timing parameters on the CAN bus. Bit timing parameters are packaged in an instance of type *BusParamsTq*. Note that the synchronization segment is excluded as it is always one time quantum long.

Example: Set the bus speed to 500 kbit/s on a CAN device with an 80 MHz oscillator:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> params = canlib.busparams.BusParamsTq(
...     tq=8,
...     phase1=2,
...     phase2=2,
...     sjw=1,
...     prescaler=20,
...     prop=3
... )
>>> ch.set_bus_params_tq(params)
```

In the example a prescaler of 20 is used, resulting in each bit comprising of 160 time quanta (8 * 20). The nominal bus speed is given by 80 * 10^6 / (20 * 8) = 500 * 10^3.

If uncertain how to set a specific bus speed, one can use *calc_busparamstq*, which returns a *BusParamsTq* object:

```
>>> calc_busparamstq(
... target_bitrate=470_000,
... target_sample_point=82,
... target_sync_jump_width=15.3,
... clk_freq=clock_info.frequency(),
... target_prop_tq=50,
... prescaler=2)
BusParamsTq(tq=85, prop=25, phase1=44, phase2=15, sjw=13, prescaler=2)
```

For users that are not interested in specifying individual bit timing parameters, CANlib also provides a set of default parameter settings for the most common bus speeds through the *Bitrate* class. The predefined bitrate constants may be set directly in the call to *openChannel*:

```
>>> ch = canlib.openChannel(channel=0, bitrate=canlib.Bitrate.BITRATE_500K)
```

Table 1: Bit timing parameters for some of the most common bus speeds on a CAN device with an 80 MHz oscillator[1]

	tq	phase1	phase2	sjw	prop	prescaler	Sample point	Bitrate
BITRATE_10K	16	4	4	1	7	500	75%	10 kbit/s
BITRATE_50K	16	4	4	1	7	100	75%	50 kbit/s
BITRATE_62K	16	4	4	1	7	80	75%	62 kbit/s
BITRATE_83K	8	2	2	2	3	120	75%	83 kbit/s
BITRATE_100K	16	4	4	1	7	50	75%	100 kbit/s
BITRATE_125K	16	4	4	1	7	40	75%	125 kbit/s
BITRATE_250K	8	2	2	1	3	40	75%	250 kbit/s
BITRATE_500K	8	2	2	1	3	20	75%	500 kbit/s
BITRATE_1M	8	2	2	1	3	10	75%	1 Mbit/s

If uncertain how to calculate bit timing parameters, appropriate values can be acquired using the Bit Timing Calculator. Note that in classic CAN mode, only the nominal bus parameters are of concern when using the Bit Timing Calculator.

Set CAN FD Bitrate

After opening a channel in CAN FD mode (see *Open as CAN FD*), bit timing parameters for both the arbitration and data phases need to be set. This is done by a call to *set_bus_params_tq*, with two separate instances of type *BusParamsTq* as arguments.

Example: Set the arbitration phase bitrate to 500 kbit/s and the data phase bitrate to 1000 kbit/s, with sampling points at 80%.

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0, flags=canlib.Open.CAN_FD)
>>> params_arbitration = canlib.busparams.BusParamsTq(
...     tq=80,
...     phase1=16,
...     phase2=16,
...     sjw=16,
...     prescaler=2,
...     prop=47
```

(continues on next page)

[1] See *Check Channel Capabilities* for information on clock frequency.

(continued from previous page)

```
... )
>>> params_data = canlib.busparams.BusParamsTq(
...     tq=40,
...     phase1=31,
...     phase2=8,
...     sjw=8,
...     prescaler=2,
...     prop=0
... )
>>> ch.set_bus_params_tq(params_arbitration, params_data)
```

For users that are not interested in specifying individual bit timing parameters, CANlib also provides a set of default parameter settings for the most common bus speeds through the *BitrateFD* class. The predefined bitrates may be set directly in the call to *openChannel*:

```
>>> ch = canlib.openChannel(
...     channel=0,
...     flags=canlib.Open.CAN_FD,
...     bitrate=canlib.BitrateFD.BITRATE_500K_80P,
...     data_bitrate=canlib.BitrateFD.BITRATE_1M_80P,
... )
```

For CAN FD bus speeds other than the predefined *BitrateFD*, bit timing parameters have to be specified manually.

Table 2: Available predefined bitrate constants with corresponding bit timing parameters for a CAN FD device with an 80 MHz oscillator[Page 20, 1]

	tq	phase1	phase2	sjw	prop	prescaler	Sample point	Bitrate
BITRATE_500K_80P	40	8	8	8	23	4	80%	500 kbit/s
BITRATE_1M_80P	40	8	8	8	23	2	80%	1 Mbit/s
BITRATE_2M_80P	20	8	4	4	7	2	80%	2 Mbit/s
BITRATE_2M_60P	20	8	8	4	3	2	60%	2 Mbit/s
BITRATE_4M_80P	10	7	2	2	0	2	80%	4 Mbit/s
BITRATE_8M_80P	10	7	2	1	0	1	80%	8 Mbit/s
BITRATE_8M_70P	10	6	3	1	0	1	70%	8 Mbit/s
BITRATE_8M_60P	5	2	2	1	0	2	60%	8 Mbit/s

If uncertain how to calculate bit timing parameters, appropriate values can be acquired using the Bit Timing Calculator.

CAN Driver Modes

Use *setBusOutputControl* to set the bus driver mode. This is usually set to *NORMAL* to obtain the standard push-pull type of driver. Some controllers also support *SILENT* which makes the controller receive only, not transmit anything, not even ACK bits. This might be handy for e.g. when listening to a CAN bus without interfering.

```
>>> from canlib import canlib
>>> with canlib.openChannel(channel=1) as ch:
...     ch.setBusOutputControl(canlib.Driver.SILENT)
...     ...
```

NORMAL is set by default.

Note: Using *setBusOutputControl* to set the bus driver mode to *SILENT* on a device that do not support Silent mode will not result in any error messages or warnings, the CAN Driver Mode will just remain in *NORMAL* mode.

A device that supports Silent mode returns *SILENT_MODE* when asked using `canlib.ChannelData.channel_cap`.

Legacy Functions

The following functions are still supported by canlib.

Set CAN Bitrate

setBusParams can be used to set the CAN bus parameters, including bitrate, the position of the sampling point etc, they are also described in most CAN controller data sheets. Depending on device and installed firmware, the requested parameters may be subject to scaling in order to accommodate device specific restrictions. As such, reading back bus parameters using *getBusParamsFd* can return bus parameter settings different than the ones supplied. Note however, that a successful call to *setBusParamsFd* will always result in the requested bit rate being set on the bus, along with bus parameters that for all intents and purposes are equivalent to the ones requested.

Set the speed to 125 kbit/s, each bit comprising 8 (= 1 + 4 + 3) quanta, the sampling point occurs at 5/8 of a bit; SJW = 1; one sampling point:

```
>>> ch.setBusParams(freq=125000, tseg1=4, tseg2=3, sjw=1, noSamp=1)
```

Set the speed to 111111 kbit/s, the sampling point to 75%, the SJW to 2 and the number of samples to 1:

```
>>> ch.setBusParams(freq=111111, tseg1=5, tseg2=2, sjw=2, noSamp=1)
```

For full bit timing control, use *set_bus_params_tq* instead.

Set CAN FD Bitrate

After a channel has been opened in CAN FD mode, *setBusParams*, and *setBusParamsFd* can be used to set the arbitration and data phase bitrates respectively. Depending on device and installed firmware, the requested parameters may be subject to scaling in order to accommodate device specific restrictions. As such, reading back bus parameters using *getBusParamsFd* can return bus parameter settings different than the ones supplied. Note however, that a successful call to *setBusParamsFd* will always result in the requested bit rate being set on the bus, along with bus parameters that for all intents and purposes are equivalent to the ones requested.

Set the nominal bitrate to 500 kbit/s and the data phase bitrate to 1000 kbit/s, with sampling points at 80%.

```
>>> ch.setBusParams(freq=500000, tseg1=63, tseg2=16, sjw=16, noSamp=1);
>>> ch.setBusParamsFd(freq_brs=1000000, tseg1_brs=31, tseg2_brs=8, sjw_brs=8);
```

For full bit timing control, use *set_bus_params_tq* instead.

1.4.5 CAN Frames

CAN Data Frames

The CAN Data Frame, represented by the *Frame* object, is the most common message type, which consists of the following major parts (a few details are omitted for the sake of brevity):

CAN identifier: *canlib.Frame.id*
The CAN identifier, or Arbitration Field, determines the priority of the message when two or more nodes are contending for the bus. The CAN identifier contains for:

- CAN 2.0A, an 11-bit Identifier and one bit, the RTR bit, which is dominant for data frames.
- CAN 2.0B, a 29-bit Identifier, with the EXT bit set, (which also contains two recessive bits: SRR and IDE) and the RTR bit.

Data field: *canlib.Frame.data*
The Data field contains zero to eight bytes of data.

Data Length Code: *canlib.Frame.dlc*
The DLC field specifies the number of data bytes in the Data field.

CRC Field:
The CRC Field contains a 15-bit checksum calculated on most parts of the message. This checksum is used for error detection.

Acknowledgement Slot:
Any CAN controller that has been able to correctly receive the message sends an Acknowledgement bit at the end of each message. The transmitter checks for the presence of the Acknowledge bit and retransmits the message if no acknowledge was detected.

Note: It is worth noting that the presence of an Acknowledgement Bit on the bus does not mean that any of the intended addressees has received the message. The only thing we know is that one or more nodes on the bus has received it correctly. The Identifier in the Arbitration Field is not, despite of its name, necessarily identifying the contents of the message.

The *canlib.Frame.flags* attribute consists of message information flags, according to *MessageFlag*.

CAN FD Data Frames

A standard CAN network is limited to 1 MBit/s, with a maximum payload of 8 bytes per frame. CAN FD increases the effective data-rate by allowing longer data fields - up to 64 bytes per frame - without changing the CAN physical layer. CAN FD also retains normal CAN bus arbitration, increasing the bit-rate by switching to a shorter bit time only at the end of the arbitration process and returning to a longer bit time at the CRC Delimiter, before the receivers send their acknowledge bits. A realistic bandwidth gain of 3 to 8 times what's possible in CAN will particularly benefit flashing applications.

Error Frames

Nearly all hardware platforms support detection of Error Frames. If an Error Frame arrives, the flag *ERROR_FRAME* is set in the *Frame*. The identifier is garbage if an Error Frame is received, but for LAPcan it happens to be 2048 plus the error code from the SJA1000.

Many platforms support transmission of Error Frames as well. To send Error Frames, set the *ERROR_FRAME* flag in the *Frame* before sending using *write*.

Simply put, the Error Frame is a special message that violates the framing rules of a CAN message. It is transmitted when a node detects a fault and will cause all other nodes to detect a fault - so they will send Error Frames, too. The transmitter will then automatically try to retransmit the message. There is an elaborate scheme of error counters that ensures that a node can't destroy the bus traffic by repeatedly transmitting error frames.

The Error Frame consists of an Error Flag, which is 6 bits of the same value (thus violating the bit-stuffing rule) and an Error Delimiter, which is 8 recessive bits. The Error Delimiter provides some space in which the other nodes on the bus can send their Error Flags when they detect the first Error Flag.

Remote Requests

You can send remote requests by passing the *RTR* flag to *write*. Received remote frames are reported by *read* et.al. using the same flag.

The Remote Frame is just like the Data Frame, with two important differences:

- It is explicitly marked as a Remote Frame (the RTR bit in the Arbitration Field is recessive)
- There is no Data Field.

The intended purpose of the Remote Frame is to solicit the transmission of the corresponding Data Frame. If, say, node A transmits a Remote Frame with the Arbitration Field set to 234, then node B, if properly initialized, might respond with a Data Frame with the Arbitration Field also set to 234.

Remote Frames can be used to implement a type of request-response type of bus traffic management. In practice, however, the Remote Frame is little used. It is also worth noting that the CAN standard does not prescribe the behaviour outlined here. Most CAN controllers can be programmed either to automatically respond to a Remote Frame, or to notify the local CPU instead.

There's one catch with the Remote Frame: the Data Length Code must be set to the length of the expected response message even though no data is sent. Otherwise the arbitration will not work.

Sometimes it is claimed that the node responding to the Remote Frame is starting its transmission as soon as the identifier is recognized, thereby "filling up" the empty Remote Frame. This is not the case.

Overload Frames

Overload Frames aren't used nowadays. Certain old CAN controllers (Intel 82526) used them to delay frame processing in certain cases.

Other frame features of interest

There are some other frame features of interest:

- You can send wakeup frames (used for Single-Wire CAN) if your hardware supports it, for example, a LAPcan plus a DRVcan S. Just set the *WAKEUP* flag.
- For "low-speed CAN" (1053/1054 type transceivers), the *NERR* flag is set if a frame is received in "fault-tolerant" mode.

1.4.6 Send and Receive

Bus On / Bus Off

When the CAN controller is on bus, it is receiving messages and is sending acknowledge bits in response to all correctly received messages. A controller that is off bus is not taking part in the bus communication at all.

When you have a *Channel* object, use *busOn* to go on bus and *busOff* to go off bus.

If you have multiple *Channel* objects to the same controller, the controller will go off bus when the last of the *Channel* objects go off bus (i.e. all *Channel* objects must be off bus for the controller to be off bus). You can use *readStatus* and watch the flag *BUS_OFF* to see if the controller has gone off bus.

You can set a channel to silent mode by using the `SILENT` mode if you want it to be on-bus without interfering with the traffic in any way, see *CAN Driver Modes*.

This example opens a channel, takes it on-bus, then takes it off-bus and closes it:

```
>>> from canlib import canlib
... with canlib.openChannel(channel=1) as ch:
...     ch.busOn()
...     ...
...     ch.busOff()
```

Reading Messages

Incoming messages are placed in a queue in the driver. In most cases the hardware does message buffering as well. You can read the first message in the queue by calling *read*, which will raise the exception *CanNoMsg* if there was no message available.

The *flags* attribute of the *Frame* returned by *read* contains a combination of the *MessageFlag* flags, including *FDF*, *BRS*, and *ESI* if the CAN FD protocol is enabled, and error flags such as *OVERRUN* which provides you with more information about the message; for example, a frame with a 29-bit identifier will have the *EXT* bit set, and a remote frame will have the *RTR* bit set. Note that the flag argument is a combination of the *MessageFlag*, so more than one flag might be set.

See *CAN Frames* for more information.

Sometimes it is desirable to have a peek into the more remote parts of the queue. Is there, for example, any message waiting that has a certain identifier?

- If you want to read just a message with a specified identifier, and throw all others away, you can call *readSpecificSkip*. This routine will return the first message with the specified identifier, discarding any other message in front of the desired one.
- If you want to wait until a message arrives (or a timeout occurs) and then read it, call *read* with a timeout.

- If you want to wait until there is at least one message in the queue with a certain identifier, but you don't want to read it, call *readSyncSpecific*.

The following code fragment reads the next available CAN message, (using default bitrate 500 kbit/s):

```
>>> from canlib import canlib
... with canlib.openChannel(channel=0) as ch:
...     ch.busOn()
...     frame = ch.read(timeout=1000)
...     ch.busOff()
>>> frame
Frame(id=709, data=bytearray(b'\xb5R'), dlc=2, flags=<MessageFlag.STD: 2>, timestamp=3)
```

Acceptance Filters

You can set filters to reduce the number of received messages. CANlib supports setting of the hardware filters on the CAN interface board. This is done with the *canAccept* function.

You set an acceptance code and an acceptance mask which together determine which CAN identifiers are accepted or rejected.

If you want to remove an acceptance filter, call *canAccept* with the mask set to NULL_MASK.

To set the mask to 0xF0 and the code to 0x60:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> ch.canAccept(0x0f0, canlib.AcceptFilterFlag.SET_MASK_STD)
>>> ch.canAccept(0x060, canlib.AcceptFilterFlag.SET_CODE_STD)
>>> ...
>>> ch.close()
```

This code snippet will cause all messages having a standard (11-bit) identifier with bit 7 - bit 4 in the identifier equal to 0110 (binary) will pass through. Other messages with standard identifiers will be rejected.

How acceptance filters can be used in a smaller project:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> # The acceptance filter only have to be called once for each ch object
>>> ch.canAccept(0x0f0, canlib.AcceptFilterFlag.SET_MASK_STD)
>>> ch.canAccept(0x060, canlib.AcceptFilterFlag.SET_CODE_STD)
>>> ...
>>> # We can now run the rest of the program and the acceptance filter
>>> # will reject unwanted CAN messages.
>>> while(True):
>>>     frame = ch.read()
>>>     ...
>>> ...
```

Code and Mask Format

Explanation of the code and mask format used by *canAccept()* and *MessageFilter*:

> A binary 1 in a mask means "the corresponding bit in the code is relevant" A binary 0 in a mask means "the corresponding bit in the code is not relevant" A relevant binary 1 in a code means "the corresponding bit in the identifier must be 1" A relevant binary 0 in a code means "the corresponding bit in the identifier must be 0"

In other words, the message is accepted if `((code XOR id) AND mask) == 0`.

Sending Messages

You transmit messages by calling *write*. Outgoing CAN messages are buffered in a transmit queue and sent on a First-In First-Out basis. You can use *writeSync* to wait until the messages in the queue have been sent.

Sending a CAN message:

```
>>> from canlib import canlib, Frame
... with canlib.openChannel(channel=0) as ch:
...     ch.busOn()
...     frame = Frame(id_=234, data=[1,2])
...     ch.write(frame)
...     ch.busOff()
```

Using Extended CAN (CAN 2.0B)

"Standard" CAN has 11-bit identifiers in the range 0 - 2047. "Extended" CAN, also called CAN 2.0B, has 29-bit identifiers. You specify which kind of identifiers you want to use in your call to canWrite(): if you set the `EXT` flag in the flag argument, the message will be transmitted with a 29-bit identifier. Conversely, received 29-bit-identifier messages have the `EXT` flag set.

The following code fragment sends a CAN message on an already open channel. The CAN message will have identifier 1234 (extended) and DLC = 8. The contents of the data bytes will be whatever the data array happens to contain:

```
>>> frame = Frame(id_=1234, data=[1,2,3,4,5,6,7,8], flags=canlib.MessageFlag.EXT)
>>> frame
Frame(id=1234, data=bytearray(b'\x01\x02\x03\x04\x05\x06\x07\x08'), dlc=8, flags=
↪<MessageFlag.EXT: 4>, timestamp=None)
>>> ch.write(frame)
```

Object Buffers

Some of the Kvaser interfaces are equipped with hardware buffers for automatic sending and responding to messages. They can be used when the timing conditions are strict, and might not be possible to fulfill on the application level. The number of buffers are, depending on the device, typically limited to around 8 buffers.

There are two types of buffers, auto response and auto transmit.

- **Auto response** sends a defined message immediately upon receiving a message meeting some condition.
- **Auto transmit** sends a message periodically, with higher timing accuracy than can be achieved by an application working through driver and operating system.

The following example sets up an Auto response object buffer which responds with a CAN frame with CAN ID 200 when a CAN frame with CAN ID 100 is received.:

```
>>> from canlib import canlib, Frame
>>> ch = canlib.openChannel(0)
>>> msg_filter = canlib.objbuf.MessageFilter(code=100, mask=0xFFFF)
>>> frame = Frame(id_=200, data=[1, 2, 3, 4])
>>> response_buf = ch.allocate_response_objbuf(filter=msg_filter, frame=frame)
>>> response_buf.enable()
```

When creating the *MessageFilter*, you can use *MessageFilter()* to verify that the correct CAN ID will be filtered:

```
>>> msg_filter = canlib.objbuf.MessageFilter(code=100, mask=0xFFFF)
>>> msg_filter(100)
True
>>> msg_filter(110)
False
```

See also *Code and Mask Format* for an explanation of the *code and mask* format used by *MessageFilter*.

The following example sets up an Auto transmit buffer to periodically send a CAN frame with CAN ID 300 every second, for 5 seconds.:

```
>>> from canlib import canlib, Frame
>>> ch = canlib.openChannel(0)
>>> frame = Frame(id_=300, data=[1, 2, 3, 4])
>>> periodic_buffer = ch.allocate_periodic_objbuf(period_us=1_000_000, frame=frame)
>>> periodic_buffer.set_msg_count(5)
>>> periodic_buffer.enable()
```

For more advanced usecases, see *t Programming*.

1.4.7 Bus Errors

Obtaining Bus Status Information

Use *read_error_counters* to read the error counters of the CAN controller. There are two such counters in a CAN controller (they are required by the protocol definition). Not all CAN controllers allow access to the error counters, so CANlib may provide you with an "educated guess" instead.

Use *readStatus* to obtain the bus status (error active, error passive, bus off; as defined by the CAN standard).

Overruns

If the CAN interface or the driver runs out of buffer space, or if the bus load is so high that the CAN controller can't keep up with the traffic, an overload condition is flagged to the application.

The driver will set the *HW_OVERRUN* and/or *SW_OVERRUN* flags in the flag argument of *read* and its relatives. The flag(s) will be set in the first message read from the driver after the overrun or overload condition happened.

Not all hardware platforms can detect the difference between hardware overruns and software overruns, so your application should test for both conditions. You can use the symbol *OVERRUN* for this purpose.

Error Frames

When a CAN controller detects an error, it transmits an error frame. This is a special CAN message that causes all other CAN controllers on the bus to notice that an error has occurred.

CANlib will report error frames to the application just like it reports any other CAN message, but the *ERROR_FRAME* flag will be set in the flags parameter when e.g. *read* returns.

When an error frame is received, its identifier, DLC and data bytes will be undefined. You should test if a message is an error frame before checking its identifier, DLC or data bytes.

In an healthy CAN system, error frames should rarely, if ever, occur. Error frames usually mean there is some type of serious problem in the system, such as a bad connector, a bad cable, bus termination missing or faulty, or another node transmitting at wrong bit rate, and so on.

1.4.8 Time Measurement

CAN messages are time stamped as they arrive. This time stamping is, depending on your hardware platform, done either by the CAN interface hardware or by CANlib.

In the former case, the accuracy is pretty good, in the order of 1 - 10 microseconds; when CANlib does the job, the accuracy is more like 100 microseconds to 10 milliseconds and you may experience a rather large jitter. This is because Windows is not a real-time operating system.

Use *Channel.readTimer* to read the current time, the return value is the current time using the clock of that channel.

Accuracy

The accuracy of the time stamps depends on the hardware.

The members of the Kvaser Leaf family have an onboard CPU. The time stamp accuracy varies (check the hardware manual) but the high-end members have very precise time stamping. The accuracy can be as good as one microsecond depending on the hardware. If more than one Leaf is used, their clocks are automatically kept in sync by the Kvaser MagiSync™ technology.

Other CAN interfaces, like the Kvaser Leaf, LAPcan and USBcan II, have an on-board CPU and clock and provide very accurate time stamps for incoming CAN messages. The accuracy is typically 10-20 microseconds.

Certain interfaces, like the PCIcan (PCI) series of boards, don't have an on-board CPU so the driver relies on the clock in the PC to timestamp the incoming messages. As Windows is not a real-time operating system, this gives an accuracy which is in the order of one millisecond.

Resolution

The resolution of the time stamps is, by default, 1 ms. It can be changed to a better resolution if desired.

Use *IOControl* attribute `timer_scale` to change the resolution of the time stamps, if desired. This will not affect the accuracy of the time stamps.

1.4.9 Using Threads

Handles are thread-specific

CANlib supports programs with multiple threads as long as one important condition is met: A handle to a CAN circuit should be used in only one thread.

This means that you cannot share e.g. *canlib.Channel* objects between threads. Each thread has to open its own handle to the circuit.

Also note that you must call *busOn* and *busOff* once for each handle even if the handles are opened on the same physical channel.

Local echo feature

If you are using the same channel via multiple handles, note that the default behaviour is that the different handles will "hear" each other just as if each handle referred to a channel of its own. If you open, say, channel 0 from thread A and thread B and then send a message from thread A, it will be "received" by thread B. This behaviour can be changed using *IOControl* and `local_txecho`.

Init access

Init access means that the thread that owns the handle can set bit rate and CAN driver mode. Init access is the default. At most one thread can have init access to any given channel. If you try to set the bit rate or CAN driver mode for a handle to which you don't have init access, the call will silently fail, unless you enable access error reporting by using *IOControl* and `report_access_errors`. Access error reporting is by default off.

Using the same handle in different threads

In spite of what was said above, you can use a single handle in different threads, provided you create the appropriate mutual exclusion mechanisms yourself. Two threads should never call CANlib simultaneously unless they are using different handles.

1.4.10 t Programming

The Kvaser t programming language is event oriented and modeled after C. It can be used to customize the behavior of the Kvaser Memorator v2 and other Kvaser t capable devices.

A t program is invoked via hooks, which are entry points that are executed at the occurrence of certain events. These events can be, for example, the arrival of specific CAN messages, timer expiration, or external input.

Here we will describe how to interact with t programs on a Kvaser device (i.e. loading, starting, stopping) For a complete reference to the t language, see the Kvaser t Programming Language available from https://www.kvaser.com/downloads.

Load and Unload t Program

The first step is to compile your t program into a .txe file, see the Kvaser t Programming Language. A compiled .txe file may be examined using *Txe*:

```
>>> t = canlib.Txe("HelloWorld.txe")
>>> t.description
'This is my hello world program.'
```

Before starting a t program you need to load it into an available "slot". Some Kvaser devices have multiple slots, and are therefore capable of running multiple programs simultaneously.

To load a t program located on the host PC, use *Channel.scriptLoadFile()*. The *canlib.Channel* used determines the default channel for the loaded t program. If your channel was opened to a device's second channel, the default channel number will be set to 1 (the numbering of channel on the card starts from 0). You can read this value using `Channel.channel_data.chan_no_on_card`

To load a t program located on the device, use *Channel.scriptLoadFileOnDevice()*. To copy arbitrary files to and from the the device, use *Channel.fileCopyToDevice()* and *Channel.fileCopyFromDevice()* respectively.

To unload a stopped script, use *Channel.scriptUnload()*.

You may use *Channel.fileGetCount()*, and *Channel.fileGetName()* to examine files located on the Kvaser device, and *Channel.fileDelete()* to delete a specific file.

Note: Not all Kvaser devices support storing t programs and other files locally on the device (i.e. USBcan Pro 2xHS v2). Please refer to your device's User Guide for more information. All User Guides may be downloaded from www.kvaser.com/downloads.

Start and Stop a t Program

To start a previously loaded t program, use *Channel.scriptStart()*. You may stop a running script using *Channel.scriptStop()*. To examine the status of a slot (i.e. if the slot is free or has a program loaded or running), use *Channel.scriptStatus()*.

Example

The following code fragment shows how to load the compiled t program "HelloWorld.txe" from the PC, and then start and stop the t program:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(0)
>>> ch.scriptLoadFile(slot=0, filePathOnPC="C:/dev/HelloWorld.txe")
>>> ch.scriptStatus(slot=0)
<ScriptStatus.LOADED: 1>
>>> ch.scriptStart(slot=0)
>>> ch.scriptStatus(slot=0)
<ScriptStatus.RUNNING|LOADED: 3>
>>> ch.scriptStop(slot=0)
>>> ch.close()
```

Environment Variables

To communicate between the PC and your t program, you can use t Environment Variables (Envvar). When a t program has been loaded, the Envvar defined in the t program can be accessed via `Channel.envvar`, however the t program must be running in order to be able to set the value of an Envvar.

There are three types of Envvar in t; `int`, `float`, and `char*`. The first two are accessed as the corresponding Python type, and the last is accessed using *canlib.envvar.DataEnvVar* which behaves as an array of bytes.

If we have a t program, `envvar.txe`, that set up three Envvar as follows:

```
envvar
{
  int   IntVal;
  float FloatVal;
  char  DataVal[512];
}

on start {
  envvarSetValue(IntVal, 0);
  envvarSetValue(FloatVal, 15.0);
  envvarSetValue(DataVal, "Fear not this night\nYou will not go astray");
}
```

The following example starts the t program `envvar.txe` and acesses it's Envvar.

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(0)
>>> ch.scriptLoadFile(slot=0, filePathOnPC="envvar.txe")
>>> ch.scriptStart(slot=0)
>>> ch.envvar.IntVal
0
>>> ch.envvar.IntVal = 3
>>> ch.envvar.IntVal
3
>>> ch.envvar.FloatVal
15.0
>>> ch.envvar.DataVal[9:20]
b'this night\n'
>>> ch.scriptStop(slot=0)
>>> ch.close()
```

Note that setting of the Envvars has also been done in the t program. For examples on how to use an Envvar in your t program, see the Kvaser t Programming Language.

Send Event

You may trigger the "on key" hook in your t program by sending a `kvEVENT_TYPE_KEY` to a running t program using *Channel.scriptSendEvent()*. The following will trigger an `on key 'a' {...}` hook:

```
>>> ch.scriptSendEvent(eventNo=ord('a'))
```

1.4.11 I/O Pin Handling

Initialize

Some Kvaser products feature I/O pins that can be used in real-time applications using a part of the API dedicated to I/O Pin Handling. This API is initialized by confirming the I/O pin configuration, see kvIoConfirmConfig. Before the configuration is confirmed the user can only retrieve information about the pins.

```
>>> from canlib import canlib, Device
... device = Device.find(serial=66666)
... channel = device.channel_number()
... ch = canlib.openChannel(channel)
... config = canlib.iopin.Configuration(ch)
... ch.get_io_pin(86).pin_type
<PinType.ANALOG: 2>
>>> for pin in config:
...     print(pin)
Pin 0: <PinType.DIGITAL: 1> <Direction.OUT: 8> bits=1 range=0.0-24.0 (<ModuleType.
↪DIGITAL: 1>)
Pin 1: <PinType.DIGITAL: 1> <Direction.OUT: 8> bits=1 range=0.0-24.0 (<ModuleType.
↪DIGITAL: 1>)
 :
Pin 31: <PinType.DIGITAL: 1> <Direction.IN: 4> bits=1 range=0.0-24.0 HL_filter=5000 LH_
↪filter=5000 (<ModuleType.DIGITAL: 1>)
```

After the configuration has been confirmed the user may set or read any values of the I/O pins:

```
>>> config.confirm()

>>> ch.get_io_pin(0).value
0
>>> ch.get_io_pin(0).value = 1

>>> ch.get_io_pin(0).value
1
```

Pin Information

Pins are identified by their pin number, which is a number from zero up to, but not including, the value returned by *number_of_io_pins*. Using the pin number, the specific properties of any pin is retrieved and set using *canlib.iopin.IoPin*.

I/O pin types

There are currently three types of pins that is supported by the API dedicated to I/O Pin Handling. These include analog, digital and relay pins. To learn what pin type a given pin is, use *canlib.iopin.IoPin.pin_type*. See *PinType* to see all supported types.

Analog Pins

The analog pins are represented by multiple bits, the number of bits can be retrieved by calling ~`.canlib.iopin.IoPin.number_of_bits`. The value of an analog pin is within in the interval given by *range_min* and *range_min*. The analog input pin has two configurable properties, namely the low pass filter order and the hysteresis. See *lp_filter_order* and *hysteresis*. Pins are read and set using *value*. When reading an output, the latest value set is retrieved.

Digital Pins

The digital pins have two configurable properties, namely the low-to-high and the high-to-low filter time. See *high_low_filter* and *low_high_filter*. Pins are read and set using *value*. When reading an output, the latest value set is retrieved.

Relay Pins

The relay pins have no configurable properties. All of these pins are considered as outputs. Pins are set and read using *value*.

1.5 Examples

This section contains a number of examples or how-tos solving common problems. They are all scripts ready to be run, using python's `argparse` to accept arguments.

1.5.1 Convert a .kme50 file to plain ASCII

```
"""convert_kme_asc.py -- Convert a .kme50 logfile into plain ASCII

This example script uses canlib.kvlclib to convert a logfile from .kme50 format
into plain ASCII.

"""
import argparse
from pathlib import Path

```

(continues on next page)

(continued from previous page)

```
from canlib import kvlclib


def try_set_propery(cnv, property, value=None):
    # Check if the format supports the given property
    if cnv.format.isPropertySupported(property):
        # If a value is specified, set the property to this value
        if value is not None:
            cnv.setProperty(property, value)

        # Get the property's default value
        default = cnv.format.getPropertyDefault(property)
        print(f'  {property} is supported (Default: {default})')

        # Get the property's current value
        value = cnv.getProperty(property)
        print(f'    Current value: {value}')
    else:
        print(f'  {property} is not supported')


def convert_events(cnv):
    # Get estimated number of remaining events in the input file. This
    # can be useful for displaying progress during conversion.
    total = cnv.eventCount()
    print(f"Converting about {total} events...")
    while True:
        try:
            # Convert events from input file one by one until EOF
            # is reached
            cnv.convertEvent()
            if cnv.isOutputFilenameNew():
                print(f"New output filename: '{cnv.getOutputFilename()}'")
                print(f"About {cnv.eventCount()} events left...")
        except kvlclib.KvlcEndOfFile:
            if cnv.isOverrunActive():
                print("NOTE! The extracted data contained overrun.")
                cnv.resetOverrunActive()
            if cnv.isDataTruncated():
                print("NOTE! The extracted data was truncated.")
                cnv.resetStatusTruncated()
            break


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Convert a .kme50 logfile into plain␣
↪ASCII.")
    parser.add_argument(
        'filename', metavar='LOGFILE.KME50', help=("The filename of the .kme50 logfile.")
    )
    args = parser.parse_args()
    in_file = Path(args.filename)
```

(continues on next page)

(continued from previous page)

```
    # set up formats
    out_fmt = kvlclib.WriterFormat(kvlclib.FileFormat.PLAIN_ASC)
    in_fmt = kvlclib.ReaderFormat(kvlclib.FileFormat.KME50)

    # set resulting output file name taking advantage of the extension
    # defined in the format.
    out_file = in_file.with_suffix('.' + out_fmt.extension)
    print(f"Output filename is '{out_file}'")

    # create converter
    cnv = kvlclib.Converter(out_file, out_fmt)

    # Set input file and format
    cnv.setInputFile(in_file, kvlclib.FileFormat.KME50)

    # split output files into max 100 MB files
    # The name of the resulting files will now end in '-partX.txt',
    # thus the first file will be named logfile-part0.txt, assuming we use
    # logfile.kme50 as input file name.
    try_set_propery(cnv, kvlclib.Property.SIZE_LIMIT, 100)

    # allow output file to overwrite existing files
    try_set_propery(cnv, kvlclib.Property.OVERWRITE, 1)

    # we are only interested in the first channel
    cnv.setProperty(kvlclib.Property.CHANNEL_MASK, 1)

    # add nice header to the output file
    try_set_propery(cnv, kvlclib.Property.WRITE_HEADER, 1)

    # we are converting CAN traffic with max 8 bytes, so we can minimize
    # the width of the data output to 8 bytes
    try_set_propery(cnv, kvlclib.Property.LIMIT_DATA_BYTES, 8)

    convert_events(cnv)

    # force flush result to disk
    cnv.flush()
```

Description

We have created a wrapper function `try_set_propery` that will examine the property we are trying to set, and ignore the setting if the current format used does not support the property. While converting events in the `convert_events` function, we also inform the user if any overruns or data truncation was detected.

Sample Output

```
C:\example>python convert_kme_asc.py gensig.kme50
Output filename is 'C:\example\gensig.txt'
  Property.SIZE_LIMIT is supported (Default: 0)
    Current value: 100
  Property.OVERWRITE is supported (Default: 0)
    Current value: 1
  Property.WRITE_HEADER is supported (Default: 0)
    Current value: 1
  Property.LIMIT_DATA_BYTES is supported (Default: 64)
    Current value: 8
Converting about 310 events...
New output filename: 'C:\example\gensig-part0.txt
About 309 events left...
```

1.5.2 Create a Database

```
"""create_db.py -- Creating a .dbc database from scratch

This script will use canlib.kvadblib to create a new database file filled with
arbitrary data.

"""

import argparse
from collections import namedtuple

from canlib import kvadblib

Message = namedtuple('Message', 'name id dlc signals')
Signal = namedtuple('Signal', 'name size scaling limits unit')
EnumSignal = namedtuple('EnumSignal', 'name size scaling limits unit enums')

_messages = [
    Message(
        name='EngineData',
        id=100,
        dlc=8,
        signals=[
            Signal(
                name='PetrolLevel',
                size=(24, 8),
                scaling=(1, 0),
                limits=(0, 255),
                unit="l",
            ),
            Signal(
                name='EngPower',
                size=(48, 16),
                scaling=(0.01, 0),
```

(continues on next page)

(continued from previous page)

```
                limits=(0, 150),
                unit="kW",
            ),
            Signal(
                name='EngForce',
                size=(32, 16),
                scaling=(1, 0),
                limits=(0, 0),
                unit="N",
            ),
            EnumSignal(
                name='IdleRunning',
                size=(23, 1),
                scaling=(1, 0),
                limits=(0, 0),
                unit="",
                enums={'Running': 0, 'Idle': 1},
            ),
            Signal(
                name='EngTemp',
                size=(16, 7),
                scaling=(2, -50),
                limits=(-50, 150),
                unit="degC",
            ),
            Signal(
                name='EngSpeed',
                size=(0, 16),
                scaling=(1, 0),
                limits=(0, 8000),
                unit="rpm",
            ),
        ],
    ),
    Message(
        name='GearBoxInfo',
        id=1020,
        dlc=1,
        signals=[
            Signal(
                name='EcoMode',
                size=(6, 2),
                scaling=(1, 0),
                limits=(0, 1),
                unit="",
            ),
            EnumSignal(
                name='ShiftRequest',
                size=(3, 1),
                scaling=(1, 0),
                limits=(0, 0),
                unit="",
```

(continues on next page)

(continued from previous page)

```
                enums={'Shift_Request_On': 1, 'Shift_Request_Off': 0},
            ),
            EnumSignal(
                name='Gear',
                size=(0, 3),
                scaling=(1, 0),
                limits=(0, 5),
                unit="",
                enums={
                    'Idle': 0,
                    'Gear_1': 1,
                    'Gear_2': 2,
                    'Gear_3': 3,
                    'Gear_4': 4,
                    'Gear_5': 5,
                },
            ),
        ],
    ),
]


def create_database(name, filename):
    db = kvadblib.Dbc(name=name)

    for _msg in _messages:
        message = db.new_message(
            name=_msg.name,
            id=_msg.id,
            dlc=_msg.dlc,
        )

        for _sig in _msg.signals:
            if isinstance(_sig, EnumSignal):
                _type = kvadblib.SignalType.ENUM_UNSIGNED
                _enums = _sig.enums
            else:
                _type = kvadblib.SignalType.UNSIGNED
                _enums = {}
            message.new_signal(
                name=_sig.name,
                type=_type,
                byte_order=kvadblib.SignalByteOrder.INTEL,
                mode=kvadblib.SignalMultiplexMode.MUX_INDEPENDENT,
                size=kvadblib.ValueSize(*_sig.size),
                scaling=kvadblib.ValueScaling(*_sig.scaling),
                limits=kvadblib.ValueLimits(*_sig.limits),
                unit=_sig.unit,
                enums=_enums,
            )

    db.write_file(filename)
```

(continues on next page)

(continued from previous page)

```
    db.close()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Create a database from scratch.")
    parser.add_argument('filename', help=("The filename to save the database to."))
    parser.add_argument(
        '-n',
        '--name',
        default='Engine example',
        help=("The name of the database (not the filename, the internal name."),
    )
    args = parser.parse_args()

    create_database(args.name, args.filename)
```

Description

While the name of the created database and the filename it is saved as is passed as arguments to `create_database`, the contents of the database is defined in the variable `_messages`. This is a list of `Message` namedtuples that describes all the messages to be put in the database:

- Their `name`, `id`, and `dlc` fields are passed to *new_message*.
- Their `signals` attribute is a list of `Signal` or `EnumSignal` namedtuples. All their fields will be passed to `new_signal`.

Sample Output

With the `_messages` variable as shown above, the following .dbc file is created:

```
VERSION "HIPBNYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY/4/%%%/4/'%**4YYY///"


NS_ :
     NS_DESC_
     CM_
     BA_DEF_
     BA_
     VAL_
     CAT_DEF_
     CAT_
     FILTER
     BA_DEF_DEF_
     EV_DATA_
     ENVVAR_DATA_
     SGTYPE_
     SGTYPE_VAL_
     BA_DEF_SGTYPE_
     BA_SGTYPE_
     SIG_TYPE_REF_
```

(continues on next page)

(continued from previous page)

```
    VAL_TABLE_
    SIG_GROUP_
    SIG_VALTYPE_
    SIGTYPE_VALTYPE_
    BO_TX_BU_
    BA_DEF_REL_
    BA_REL_
    BA_DEF_DEF_REL_
    BU_SG_REL_
    BU_EV_REL_
    BU_BO_REL_
    SG_MUL_VAL_

BS_:

BU_:


BO_ 100 EngineData: 8 Vector__XXX
 SG_ PetrolLevel : 24|8@1+ (1,0) [0|255] "l" Vector__XXX
 SG_ EngPower : 48|16@1+ (0.01,0) [0|150] "kW" Vector__XXX
 SG_ EngForce : 32|16@1+ (1,0) [0|0] "N" Vector__XXX
 SG_ IdleRunning : 23|1@1+ (1,0) [0|0] "" Vector__XXX
 SG_ EngTemp : 16|7@1+ (2,-50) [-50|150] "°C" Vector__XXX
 SG_ EngSpeed : 0|16@1+ (1,0) [0|8000] "rpm" Vector__XXX

BO_ 1020 GearBoxInfo: 1 Vector__XXX
 SG_ EcoMode : 6|2@1+ (1,0) [0|1] "" Vector__XXX
 SG_ ShiftRequest : 3|1@1+ (1,0) [0|0] "" Vector__XXX
 SG_ Gear : 0|3@1+ (1,0) [0|5] "" Vector__XXX


BA_DEF_  "BusType" STRING ;
BA_DEF_DEF_  "BusType" "";
BA_ "BusType" "CAN";
VAL_ 100 IdleRunning 0 "Running" 1 "Idle" ;
VAL_ 1020 ShiftRequest 1 "Shift_Request_On" 0 "Shift_Request_Off" ;
VAL_ 1020 Gear 0 "Idle" 2 "Gear_2" 1 "Gear_1" 5 "Gear_5" 3 "Gear_3" 4 "Gear_4" ;
```

1.5.3 Monitor Channel Using CAN Database

```
"""dbmonitor.py -- Read CAN messages using a database

This script will use canlib.canlib and canlib.kvadblib to monitor a CAN
channel, and look up all messages received in a database before printing them.

It requires a CANlib channel with a connected device capable of receiving CAN
messages, some source of CAN messages, and the same database that the source is
using to generate the messages.
```

(continues on next page)

(continued from previous page)

```
In this example the channel is opened with flag canOPEN_ACCEPT_LARGE_DLC (optional).
This enables a DLC larger than 8 bytes (up to 15 for classic CAN). If
canOPEN_ACCEPT_LARGE_DLC is excluded, generated frames with DLC > 8, will attain
a DLC of 8 on the receiving end, which may compromise the DLC equivalence
check.

The source of the messages may be e.g. the pinger.py example script.

"""
import argparse

from canlib import canlib, kvadblib

# Create a dictionary of predefined CAN bitrates, using the name after
# "BITRATE_" as key. E.g. "500K".
bitrates = {x.name.replace("BITRATE_", ""): x for x in canlib.Bitrate}


def printframe(db, frame):
    try:
        bmsg = db.interpret(frame)
    except kvadblib.KvdNoMessage:
        print(f"<<< No message found for frame with id {frame.id} >>>")
        return

    if not bmsg._message.dlc == bmsg._frame.dlc:
        print(
            "<<< Could not interpret message because DLC does not match for"
            f" frame with id {frame.id} >>>"
        )
        print(f"\t- DLC (database): {bmsg._message.dlc}")
        print(f"\t- DLC (received frame): {bmsg._frame.dlc}")
        return

    msg = bmsg._message

    print('', msg.name)

    if msg.comment:
        print('', f'"{msg.comment}"')

    for bsig in bmsg:
        print('', bsig.name + ':', bsig.value, bsig.unit)

    print('')


def monitor_channel(ch, db, ticktime):

    timeout = 0.5
    tick_countup = 0
    if ticktime <= 0:
```

(continues on next page)

(continued from previous page)

```
        ticktime = None
    elif ticktime < timeout:
        timeout = ticktime

    print("Listening...")
    while True:
        try:
            frame = ch.read(timeout=int(timeout * 1000))
            printframe(db, frame)
        except canlib.CanNoMsg:
            if ticktime is not None:
                tick_countup += timeout
                while tick_countup > ticktime:
                    print("tick")
                    tick_countup -= ticktime
        except KeyboardInterrupt:
            print("Stop.")
            break


def parse_args(argv):
    parser = argparse.ArgumentParser(
        description="Listen on a CAN channel and print all signals received, as␣
↪specified by a database."
    )
    parser.add_argument(
        'channel', type=int, default=1, nargs='?', help=("The channel to listen on.")
    )
    parser.add_argument(
        '--db',
        default="engine_example.dbc",
        help=("The database file to look up messages and signals in."),
    )
    parser.add_argument(
        '--bitrate', '-b', default='500k', help=("Bitrate, one of " + ', '.join(bitrates.
↪keys()))
    )
    parser.add_argument(
        '--ticktime',
        '-t',
        type=float,
        default=0,
        help=("If greater than zero, display 'tick' every this many seconds"),
    )
    args = parser.parse_args()
    return args


def main(argv=None):
    args = parse_args(argv)
    db = kvadblib.Dbc(filename=args.db)
    ch = canlib.openChannel(args.channel, canlib.canOPEN_ACCEPT_LARGE_DLC, bitrates[args.
```

(continues on next page)

(continued from previous page)

```
↪bitrate.upper()])
    ch.setBusOutputControl(canlib.canDRIVER_NORMAL)
    ch.busOn()

    monitor_channel(ch, db, args.ticktime)


if __name__ == '__main__':
    raise SystemExit(main())
```

Description

Any CAN messages received on the specified channel will be looked up in the database using `interpret`, which allows the script to print the "phys" value of each signal instead of just printing the raw data (as *Monitor a Channel* does). The script also prints the message's name and comment (if available), as well as the signals name and unit.

Sample Output

```
EngineData
PetrolLevel: 0.0 l
EngPower: 12.0 kW
EngForce: 0.0 N
IdleRunning: 0.0
EngTemp: -30.0 °C
EngSpeed: 7735.0 rpm

GearBoxInfo
EcoMode: 0.0
ShiftRequest: 0.0
Gear: 0.0

EngineData
PetrolLevel: 0.0 l
EngPower: 28.0 kW
EngForce: 0.0 N
IdleRunning: 0.0
EngTemp: -30.0 °C
EngSpeed: 3467.0 rpm

GearBoxInfo
EcoMode: 1.0
ShiftRequest: 0.0
Gear: 0.0

EngineData
PetrolLevel: 0.0 l
EngPower: 0.0 kW
EngForce: 0.0 N
IdleRunning: 0.0
EngTemp: -50.0 °C
```

(continues on next page)

(continued from previous page)

```
EngSpeed: 1639.0 rpm

GearBoxInfo
EcoMode: 1.0
ShiftRequest: 0.0
Gear: 1.0

EngineData
PetrolLevel: 60.0 l
EngPower: 0.0 kW
EngForce: 0.0 N
IdleRunning: 0.0
EngTemp: 142.0 °C
EngSpeed: 0.0 rpm

GearBoxInfo
EcoMode: 0.0
ShiftRequest: 0.0
Gear: 0.0

EngineData
PetrolLevel: 172.0 l
EngPower: 51.0 kW
EngForce: 0.0 N
IdleRunning: 0.0
EngTemp: -50.0 °C
EngSpeed: 0.0 rpm

GearBoxInfo
EcoMode: 0.0
ShiftRequest: 0.0
Gear: 0.0
```

CAN FD version

This example is basically the same as `dbmonitor.py` above, except we are now using CAN FD.

Note that you also need the `dbmonitor.py` file, next to `dbmonitorfd.py` below, since we are reusing the `monitor_channel` function.

```
"""dbmonitorfd.py -- Read CAN FD messages using a database

This script will use canlib.canlib and canlib.kvadblib to monitor a CAN FD
channel, and look up all messages received in a database before printing them.

It requires a CANlib channel with a connected device capable of receiving CAN FD
messages, some source of CAN messages, and the same database that the source is
using to generate the messages.

The script also reuses the monitor_channel function defined in dbmonitor.py
```

(continues on next page)

(continued from previous page)

```
The source of the messages may be e.g. the pingerfd.py example script.

"""
import argparse

from canlib import canlib, kvadblib

import dbmonitor

# Create a dictionary of predefined CAN FD bitrates, using the name after
# "BITRATE_" as key. E.g. "500K_80P".
fd_bitrates = {x.name.replace("BITRATE_", ""): x for x in canlib.BitrateFD}


def parse_args(argv):
    parser = argparse.ArgumentParser(
        description="Listen on a CAN channel and print all signals received, as␣
↪specified by a database."
    )
    parser.add_argument(
        'channel', type=int, default=1, nargs='?', help=("The channel to listen on.")
    )
    parser.add_argument(
        '--db',
        default="engine_example.dbc",
        help=("The database file to look up messages and signals in."),
    )
    parser.add_argument(
        '--fdbitrate', '-f', default=['500k_80p', '1M_80p'], nargs=2,
        help=("CAN FD arbitration and data bitrate pair (e.g. -f 500k_80p 1M_80p), two␣
↪of " + ', '.join(fd_bitrates.keys()))
    )
    parser.add_argument(
        '--ticktime',
        '-t',
        type=float,
        default=0,
        help=("If greater than zero, display 'tick' every this many seconds"),
    )
    args = parser.parse_args()
    return args


def main(argv=None):
    args = parse_args(argv)
    db = kvadblib.Dbc(filename=args.db)
    ch = canlib.openChannel(
        args.channel,
        flags=canlib.Open.CAN_FD,
        bitrate=fd_bitrates[args.fdbitrate[0].upper()],
        data_bitrate=fd_bitrates[args.fdbitrate[1].upper()],
```

(continues on next page)

(continued from previous page)

```
    )
    ch.setBusOutputControl(canlib.canDRIVER_NORMAL)
    ch.busOn()

    dbmonitor.monitor_channel(ch, db, args.ticktime)


if __name__ == '__main__':
    raise SystemExit(main())
```

1.5.4 Examine the Contents of a Database

```
# examine_db.py
"""

This script uses canlib.kvadblib to parse a database and print its contents.

"""
import argparse
import sys

from canlib import kvadblib

INDENT = ' ' * 4


def print_db(db):
    print('DATABASE')
    print(db.name)
    for line in db_lines(db):
        print(INDENT + line)


def adef_lines(adef):
    yield 'type: ' + type(adef).__name__
    yield 'definition: ' + str(adef.definition)
    yield 'owner: ' + str(adef.owner)


def attr_lines(attrib):
    yield str(attrib.name) + ' = ' + str(attrib.value)


def db_lines(db):
    yield 'flags: ' + str(db.flags)
    yield 'protocol: ' + str(db.protocol)
    yield ''

    yield 'ATTRIBUTE DEFINITIONS'
    for adef in db.attribute_definitions():
```

(continues on next page)

(continued from previous page)

```
        yield str(adef.name)
        for line in adef_lines(adef):
            yield INDENT + line
    yield ''

    yield 'MESSAGES'
    for message in db:
        yield str(message.name)
        for line in msg_lines(message):
            yield INDENT + line
    yield ''


def enum_lines(enums):
    for name, val in enums.items():
        yield str(name) + ' = ' + str(val)


def msg_lines(message):
    yield 'id: ' + str(message.id)
    yield 'flags: ' + str(message.flags)
    yield 'dlc: ' + str(message.dlc)
    yield 'comment: ' + str(message.comment)
    yield ''

    yield 'ATTRIBUTES'
    for attr in message.attributes():
        for line in attr_lines(attr):
            yield line
    yield ''

    yield 'SIGNALS'
    for signal in message:
        yield str(signal.name)
        for line in sig_lines(signal):
            yield INDENT + line
    yield ''


def sig_lines(signal):
    for name in ('type', 'byte_order', 'mode', 'size', 'scaling', 'limits', 'unit',
↪'comment'):
        yield name + ': ' + str(getattr(signal, name))
    yield ''

    try:
        enums = signal.enums
    except AttributeError:
        pass
    else:
        yield 'ENUMERATIONS'
        for line in enum_lines(enums):
```

(continues on next page)

(continued from previous page)

```
            yield line
        yield ''

    yield 'ATTRIBUTES'
    for attr in signal.attributes():
        for line in attr_lines(attr):
            yield line
    yield ''


def examine_database(db_name):
    with kvadblib.Dbc(filename=db_name) as db:
        print_db(db)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description=sys.modules[__name__].__doc__)
    parser.add_argument(
        'db', type=str, metavar='<database.dbc>', help='The dbc database file to examine.
↪'
    )
    args = parser.parse_args()

    examine_database(args.db)
```

Description

The script is structured into several generator functions that take a `kvadblib` object and yield lines of information about it. This allows one function to add indentation to any other functions it uses.

Generally each function first yields information in the following order:

1. Any information about the object itself (e.g. `db.flags` and `db.protocol`)
2. An empty string
3. For each type of sub-object (e.g. attribute definitions):
    1. A heading (e.g. `'ATTRIBUTE_DEFINTIONS'`)
    2. For each object of that type (e.g. iterating through *attribute_definitions*):
        1. The objects name
        2. All lines from the `*_lines` function for the object type (e.g. `adef_lines`), with added indentation
    3. An empty string

Sample Output

Running this script on the database created by *Create a Database* gives the following:

```
DATABASE
engine_example
    flags: 0
    protocol: ProtocolType.CAN

    ATTRIBUTE DEFINITIONS
    BusType
        type: StringDefinition
        definition: DefaultDefinition(default='')
        owner: AttributeOwner.DB

    MESSAGES
    EngineData
        id: 100
        flags: MessageFlag.0
        dlc: 8
        comment:

        ATTRIBUTES

        SIGNALS
        PetrolLevel
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
            size: ValueSize(startbit=24, length=8)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=255.0)
            unit: l
            comment:

            ATTRIBUTES

        EngPower
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
            size: ValueSize(startbit=48, length=16)
            scaling: ValueScaling(factor=0.01, offset=0.0)
            limits: ValueLimits(min=0.0, max=150.0)
            unit: kW
            comment:

            ATTRIBUTES

        EngForce
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
```

(continues on next page)

(continued from previous page)

```
            size: ValueSize(startbit=32, length=16)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=0.0)
            unit: N
            comment:

            ATTRIBUTES

        IdleRunning
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: 0
            size: ValueSize(startbit=23, length=1)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=0.0)
            unit:
            comment:

            ENUMERATIONS
            Running = 0
            Idle = 1

            ATTRIBUTES

        EngTemp
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
            size: ValueSize(startbit=16, length=7)
            scaling: ValueScaling(factor=2.0, offset=-50.0)
            limits: ValueLimits(min=-50.0, max=150.0)
            unit: °C
            comment:

            ATTRIBUTES

        EngSpeed
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
            size: ValueSize(startbit=0, length=16)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=8000.0)
            unit: rpm
            comment:

            ATTRIBUTES


    GearBoxInfo
        id: 1020
        flags: MessageFlag.0
```

(continues on next page)

(continued from previous page)

```
        dlc: 1
        comment:

        ATTRIBUTES

        SIGNALS
        EcoMode
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: -1
            size: ValueSize(startbit=6, length=2)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=1.0)
            unit:
            comment:

            ATTRIBUTES

        ShiftRequest
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: 0
            size: ValueSize(startbit=3, length=1)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=0.0)
            unit:
            comment:

            ENUMERATIONS
            Shift_Request_On = 1
            Shift_Request_Off = 0

            ATTRIBUTES

        Gear
            type: SignalType.UNSIGNED
            byte_order: SignalByteOrder.INTEL
            mode: 0
            size: ValueSize(startbit=0, length=3)
            scaling: ValueScaling(factor=1.0, offset=0.0)
            limits: ValueLimits(min=0.0, max=5.0)
            unit:
            comment:

            ENUMERATIONS
            Gear_5 = 5
            Gear_1 = 1
            Gear_3 = 3
            Idle = 0
            Gear_4 = 4
            Gear_2 = 2
```

(continues on next page)

(continued from previous page)

```
            ATTRIBUTES
```

1.5.5 List channels

```
"""list_channels.py -- List all CANlib channel

This script uses canlib.canlib to list all CANlib channels and information
about the device that is using them.

"""
import argparse

from canlib import canlib


def print_channels():
    for ch in range(canlib.getNumberOfChannels()):
        chdata = canlib.ChannelData(ch)
        print(
            "{ch}. {name} ({ean} / {serial})".format(
                ch=ch,
                name=chdata.channel_name,
                ean=chdata.card_upc_no,
                serial=chdata.card_serial_no,
            )
        )


if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description="List all CANlib channels and information about them."
    )
    args = parser.parse_args()

    print_channels()
```

Sample Output

```
0. Kvaser Memorator Pro 2xHS v2 (channel 0) (73-30130-00819-9 / 10626)
1. Kvaser Memorator Pro 2xHS v2 (channel 1) (73-30130-00819-9 / 10626)
```

1.5.6 Monitor a Channel

```
"""monitor.py -- Print all data received on a CAN channel

This script uses canlib.canlib to listen on a channel and print all data
received.

It requires a CANlib channel with a connected device capable of receiving CAN
messages and some source of CAN messages.

The source of the messages may be e.g. the pinger.py example script.

Also see the dbmonitor.py example script for how to look up the messages
received in a database.

"""
import argparse
import shutil

from canlib import canlib

bitrates = {
    '1M': canlib.Bitrate.BITRATE_1M,
    '500K': canlib.Bitrate.BITRATE_500K,
    '250K': canlib.Bitrate.BITRATE_250K,
    '125K': canlib.Bitrate.BITRATE_125K,
    '100K': canlib.Bitrate.BITRATE_100K,
    '62K': canlib.Bitrate.BITRATE_62K,
    '50K': canlib.Bitrate.BITRATE_50K,
    '83K': canlib.Bitrate.BITRATE_83K,
    '10K': canlib.Bitrate.BITRATE_10K,
}


def printframe(frame, width):
    form = '^' + str(width - 1)
    print(format(" Frame received ", form))
    print("id:", frame.id)
    print("data:", bytes(frame.data))
    print("dlc:", frame.dlc)
    print("flags:", frame.flags)
    print("timestamp:", frame.timestamp)


def monitor_channel(channel_number, bitrate, ticktime):
    ch = canlib.openChannel(channel_number, bitrate=bitrate)
```

(continues on next page)

(continued from previous page)

```
    ch.setBusOutputControl(canlib.canDRIVER_NORMAL)
    ch.busOn()

    width, height = shutil.get_terminal_size((80, 20))

    timeout = 0.5
    tick_countup = 0
    if ticktime <= 0:
        ticktime = None
    elif ticktime < timeout:
        timeout = ticktime

    print("Listening...")
    while True:
        try:
            frame = ch.read(timeout=int(timeout * 1000))
            printframe(frame, width)
        except canlib.CanNoMsg:
            if ticktime is not None:
                tick_countup += timeout
                while tick_countup > ticktime:
                    print("tick")
                    tick_countup -= ticktime
        except KeyboardInterrupt:
            print("Stop.")
            break

    ch.busOff()
    ch.close()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description="Listen on a CAN channel and print all frames received."
    )
    parser.add_argument('channel', type=int, default=1, nargs='?')
    parser.add_argument(
        '--bitrate', '-b', default='500k', help=("Bitrate, one of " + ', '.join(bitrates.
↪keys()))
    )
    parser.add_argument(
        '--ticktime',
        '-t',
        type=float,
        default=0,
        help=("If greater than zero, display 'tick' every this many seconds"),
    )
    parser.add_argument
    args = parser.parse_args()

    monitor_channel(args.channel, bitrates[args.bitrate.upper()], args.ticktime)
```

Description

Any CAN frames received on the specified channel will be printed. Note that the signals contained in the frame is not be extracted, only the raw data is printed. To extract the signals, see *Monitor Channel Using CAN Database*.

Sample Output

```
Listening...
 Frame received
id: 1020
data: b'\x00'
flags: MessageFlag.STD
timestamp: 939214
 Frame received
id: 100
data: b'\xd1\x06\x00\x19\x00\x00\x18\x15'
flags: MessageFlag.STD
timestamp: 939215
 Frame received
id: 1020
data: b'\x01'
flags: MessageFlag.STD
timestamp: 939417
 Frame received
id: 100
data: b'\x00\x00\x00\x19\x00\x00\xc82'
flags: MessageFlag.STD
timestamp: 939418
 Frame received
id: 1020
data: b'\x00'
flags: MessageFlag.STD
timestamp: 939620
 Frame received
id: 100
data: b'\x00\x00\x00\x00\x00\x00\x903'
flags: MessageFlag.STD
timestamp: 939621
 Frame received
id: 1020
data: b'@'
flags: MessageFlag.STD
timestamp: 939823
 Frame received
id: 100
data: b')\x03\x17\xf5\x00\x00\x00\x00'
flags: MessageFlag.STD
timestamp: 939824
 Frame received
id: 1020
data: b'\x02'
flags: MessageFlag.STD
```

(continues on next page)

(continued from previous page)

```
timestamp: 940026
 Frame received
id: 100
data: b'\x1b\x0eC\x00\x00\x00\x00\x00'
flags: MessageFlag.STD
timestamp: 940027
```

1.5.7 Send Random Messages on CAN Channel

```
"""pinger.py -- Send random CAN messages based on a database

This script uses canlib.canlib and canlib.kvadblib to send random messages from
a database with random data.

It requires a CANlib channel a connected device capable of sending CAN
messages, something that receives those messages, and a database to inspect for
the messages to send.

Messages can be received and printed by e.g. the dbmonitor.py example script.

"""
import argparse
import random
import time

from canlib import canlib, kvadblib

# Create a dictionary of predefined CAN bitrates, using the name after
# "BITRATE_" as key. E.g. "500K".
bitrates = {x.name.replace("BITRATE_", ""): x for x in canlib.Bitrate}


def set_random_framebox_signal(db, framebox, signals):
    sig = random.choice(signals)
    value = get_random_value(db, sig)
    framebox.signal(sig.name).phys = value


def get_random_value(db, sig):
    limits = sig.limits
    value = random.uniform(limits.min, limits.max)

    # round value depending on type...
    if sig.type is kvadblib.SignalType.UNSIGNED or sig.type is kvadblib.SignalType.
↪SIGNED:
        # ...remove decimals if the signal was of type unsigned
        value = int(round(value))
    else:
        # ...otherwise, round to get only one decimal
        value = round(value, 1)
```

(continues on next page)

(continued from previous page)

```
    return value


def ping_loop(ch, db, num_messages, quantity, interval, seed=None):

    if seed is None:
        seed = 0
    else:
        try:
            seed = int(seed)
        except ValueError:
            seed = seed
    random.seed(seed)

    if num_messages == -1:
        used_messages = list(db)
    else:
        used_messages = random.sample(list(db), num_messages)

    print()
    print("Randomly selecting signals from the following messages:")
    print(used_messages)
    print("Seed used was " + repr(seed))
    print()

    while True:
        # Create an empty framebox each time, ignoring previously set signal
        # values.
        framebox = kvadblib.FrameBox(db)

        # Add all messages to the framebox, as we may use send any signal from
        # any of them.
        for msg in db:
            # If the channel is opened as CAN FD, we ignore messages in dbc
            # that is not marked as CAN FD. Similarily, if the channel is
            # opened as CAN, we inore messages in dbc that is marked as CAN FD.
            if ch.is_can_fd() != (canlib.MessageFlag.FDF in msg.canflags):
                continue
            framebox.add_message(msg.name)

        # Make a list of all signals (which framebox has found in all messages
        # we gave it), so that set_random_framebox_signal() can pick a random
        # one.
        signals = [bsig.signal for bsig in framebox.signals()]

        # Set some random signals to random values
        for i in range(quantity):
            set_random_framebox_signal(db, framebox, signals)

        # Send all messages/frames
        for frame in framebox.frames():
```

(continues on next page)

(continued from previous page)

```
            print('Sending frame', frame)
            ch.writeWait(frame, timeout=5000)

        time.sleep(interval)


def parse_args(argv):
    parser = argparse.ArgumentParser(
        description="Send random CAN message based on a database.",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    )
    parser.add_argument(
        'channel', type=int, default=0, nargs='?', help=("The channel to send messages␣
↪on.")
    )
    parser.add_argument(
        '--bitrate', '-b', default='500k', help=("Bitrate, one of " + ', '.join(bitrates.
↪keys()))
    )
    parser.add_argument(
        '--db', default="engine_example.dbc", help=("The database file to base messages␣
↪on.")
    )
    parser.add_argument(
        '-Q', '--quantity', type=int, default=5, help=("The number of signals to send␣
↪each tick.")
    )
    parser.add_argument(
        '-I', '--interval', type=float, default=0.2, help=("The time, in seconds,␣
↪between ticks.")
    )
    parser.add_argument(
        '-n',
        '--num-messages',
        type=int,
        default=-1,
        help=("The number of message from the database to use, or -1 to use all."),
    )
    parser.add_argument(
        '-s',
        '--seed',
        nargs='?',
        default='0',
        help=(
            "The seed used for choosing messages. If possible, will be converted to an␣
↪int."
            " If no argument is given, a random seed will be used."
        ),
    )
    args = parser.parse_args()
    return args
```

(continues on next page)

(continued from previous page)

```
def main(argv=None):
    args = parse_args(argv)
    db = kvadblib.Dbc(filename=args.db)

    ch = canlib.openChannel(args.channel, bitrate=bitrates[args.bitrate.upper()])
    ch.setBusOutputControl(canlib.canDRIVER_NORMAL)
    ch.busOn()

    ping_loop(
        ch=ch,
        db=db,
        num_messages=args.num_messages,
        quantity=args.quantity,
        interval=args.interval,
        seed=args.seed,
    )


if __name__ == '__main__':
    raise SystemExit(main())
```

Description

Note: There must be some process reading the messages for `pinger.py` to work (see e.g. *Monitor Channel Using CAN Database*).

`ping_loop` will first extract a random list of messages (see *Randomness*), and then enter a loop that creates a new *FrameBox* before adding some random signals with random values (the quantity specified by the `quantity`/`--quantity` argument).

Adding random signals is done with `set_random_framebox_signal`, which picks a random signal from the framebox, and `get_random_value` which inspects the given signal and provides a random value based on the signal's definition.

Finally, the loop pauses for `interval`/`--interval` seconds between sending messages.

Randomness

The random selection of messages is done with the `seed`/`--seed` and `num_messages`/`num-messages` arguments. If `num_messages` is `-1`, all messages from the database will be used. Otherwise, `num_message` specifies the number of messages to be randomly picked from the database.

The `seed` argument will be sent to `random.seed` before the messages are selected (which is done with `random.sample`), which means as long as the seed remains the same, the same messages are selected. The `seed` can also be set to `None` for a pseudo-random seed.

Sample Output

```
Randomly selecting signals from the following messages:
[Message(name='EngineData', id=100, flags=<MessageFlag.0: 0>, dlc=8, comment=''),␣
↪Message(name='GearBoxInfo', id=1020, flags=<MessageFlag.0: 0>, dlc=1, comment='')]
Seed used was '0'

Sending frame Frame(id=1020, data=bytearray(b'\x00'), dlc=1, flags=<MessageFlag.0: 0>,␣
↪timestamp=None)
Sending frame Frame(id=100, data=bytearray(b'\x00\x00\x16]\x00\x00\x00\x00'), dlc=8,␣
↪flags=<MessageFlag.0: 0>, timestamp=None)
Sending frame Frame(id=1020, data=bytearray(b'\x00'), dlc=1, flags=<MessageFlag.0: 0>,␣
↪timestamp=None)
Sending frame Frame(id=100, data=bytearray(b'\x00\x00\x00\xdd\x00\x00\x00\x00'), dlc=8,␣
↪flags=<MessageFlag.0: 0>, timestamp=None)
Sending frame Frame(id=1020, data=bytearray(b'\x00'), dlc=1, flags=<MessageFlag.0: 0>,␣
↪timestamp=None)
Sending frame Frame(id=100, data=bytearray(b'\x00\x00\x00\xe0\x00\x00`\t'), dlc=8, flags=
↪<MessageFlag.0: 0>, timestamp=None)
Sending frame Frame(id=1020, data=bytearray(b'\x04'), dlc=1, flags=<MessageFlag.0: 0>,␣
↪timestamp=None)
Sending frame Frame(id=100, data=bytearray(b'f\x07\n\x00\x00\x00\x00\x00'), dlc=8, flags=
↪<MessageFlag.0: 0>, timestamp=None)
Sending frame Frame(id=1020, data=bytearray(b'\x00'), dlc=1, flags=<MessageFlag.0: 0>,␣
↪timestamp=None)
Sending frame Frame(id=100, data=bytearray(b'\x0c\x15-\x00\x00\x00\x00\x00'), dlc=8,␣
↪flags=<MessageFlag.0: 0>, timestamp=None)
```

CAN FD version

This example is basically the same as `pinger.py` above, except we are now using CAN FD.

Note that you also need the `pinger.py` file, next to `pingerfd.py` below, since we are reusing the `ping_loop` function.

```
"""pingerfd.py -- Send random CAN FD messages based on a database

This script uses canlib.canlib and canlib.kvadblib to send random messages from
a database with random data.

The script also reuses the ping_loop function defined in pinger.py

It requires a CANlib channel a connected device capable of sending CAN
messages, something that receives those messages, and a database to inspect for
the messages to send.

Messages can be received and printed by e.g. the dbmonitorfd.py example script.

"""
import argparse
import random
import time
```

(continues on next page)

(continued from previous page)

```
from canlib import canlib, kvadblib

import pinger

# Create a dictionary of predefined CAN FD bitrates, using the name after
# "BITRATE_" as key. E.g. "500K_80P".
fd_bitrates = {x.name.replace("BITRATE_", ""): x for x in canlib.BitrateFD}


def parse_args(argv):
    parser = argparse.ArgumentParser(
        description="Send random CAN FD message based on a database.",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    )
    parser.add_argument(
        'channel', type=int, default=0, nargs='?', help=("The channel to send messages␣
↪on.")
    )
    parser.add_argument(
        '--fdbitrate', '-f', default=['500k_80p', '1M_80p'], nargs=2,
        help=("CAN FD arbitration and data bitrate pair (e.g. -f 500k_80p 1M_80p), two␣
↪of " + ', '.join(fd_bitrates.keys()))
    )
    parser.add_argument(
        '--db', default="engine_example.dbc", help=("The database file to base messages␣
↪on.")
    )
    parser.add_argument(
        '-Q', '--quantity', type=int, default=5, help=("The number of signals to send␣
↪each tick.")
    )
    parser.add_argument(
        '-I', '--interval', type=float, default=0.2, help=("The time, in seconds,␣
↪between ticks.")
    )
    parser.add_argument(
        '-n',
        '--num-messages',
        type=int,
        default=-1,
        help=("The number of message from the database to use, or -1 to use all."),
    )
    parser.add_argument(
        '-s',
        '--seed',
        nargs='?',
        default='0',
        help=(
            "The seed used for choosing messages. If possible, will be converted to an␣
↪int. If no argument is given, a random seed will be used."
        ),
    )
```

(continues on next page)

(continued from previous page)

```
    args = parser.parse_args()

    if args.seed is not None:
        try:
            args.seed = int(args.seed)
        except ValueError:
            args.seed = args.seed
    return args


def main(argv=None):
    args = parse_args(argv)
    db = kvadblib.Dbc(filename=args.db)
    print(f"{args.channel=}, {fd_bitrates[args.fdbitrate[0].upper()]=}, {fd_
↪bitrates[args.fdbitrate[1].upper()]=}")
    ch = canlib.openChannel(
        args.channel,
        flags=canlib.Open.CAN_FD,
        bitrate=fd_bitrates[args.fdbitrate[0].upper()],
        data_bitrate=fd_bitrates[args.fdbitrate[1].upper()],
    )
    ch.setBusOutputControl(canlib.canDRIVER_NORMAL)
    ch.busOn()

    pinger.ping_loop(
        ch=ch,
        db=db,
        num_messages=args.num_messages,
        quantity=args.quantity,
        interval=args.interval,
        seed=args.seed,
    )


if __name__ == '__main__':
    raise SystemExit(main())
```

1.5.8 Validate a Memorator Configuration

```
"""validate_memo_config.py -- validate a Memorator configuration

This script uses canlib.kvamemolibxml to load and validate a Memorator
configuration in an xml file, and then prints any errors and warnings.

It requires a Memorator configuration in xml format.

"""
import argparse

from canlib import kvamemolibxml
```

(continues on next page)

(continued from previous page)

```
def validate(filename):
    # Read in the XML configuration file
    config = kvamemolibxml.load_xml_file(filename)

    # Validate the XML configuration
    errors, warnings = config.validate()

    # Print errors and warnings
    for error in errors:
        print(error)
    for warning in warnings:
        print(warning)

    if errors or warnings:
        raise Exception("Please fix validation errors/warnings.")
    else:
        print("No errors found!")


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Validate a Memorator configuration.")
    parser.add_argument(
        'filename', default='logall.xml', nargs='?', help=("The filename of the␣
↪configuration.")
    )
    args = parser.parse_args()

    validate(args.filename)
```

1.5.9 Write a Configuration to a Memorator

```
"""write_memo_config.py -- Write a configuration to a memorator

This example script uses canlib.kvamemolibxml and canlib.kvmlib to load a
configuration file in .xml format, validate it, and then write it to a
connected Memorator.

It requires a configuration xml file and a connected Memorator device.

"""
import argparse

from canlib import kvamemolibxml, kvmlib


def write_config(filename, channel_number):
    # Read in the XML configuration file
    config = kvamemolibxml.load_xml_file(filename)
```

(continues on next page)

(continued from previous page)

```
    # Validate the XML configuration
    errors, warnings = config.validate()

    if errors or warnings:
        raise Exception("Errors or warnings found! Check validate_memo_config example.")

    # Open the device and write the configuration
    with kvmlib.openDevice(channel_number) as memo:
        memo.write_config(config.lif)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Write a configuration to a Memorator.")
    parser.add_argument(
        'filename', default='logall.xml', nargs='?', help=("The filename of the␣
↪configuration.")
    )
    parser.add_argument(
        'channel',
        type=int,
        default=0,
        nargs='?',
        help=("The channel number of the device the configuration should be written to.
↪"),
    )
    args = parser.parse_args()

    write_config(args.filename, args.channel)
```

1.6 Reference

1.6.1 Base Exceptions

CanlibException

exception canlib.**CanlibException**

Base class for all exceptions in canlib

DllException

exception canlib.**DllException**

Bases: *CanlibException*

Base class for exceptions from dll calls in canlib

All instances of this class must have a `status` attribute defined (this is enforced in the constructor). Its value is the status code that caused the exception.

property canERR

Deprecated name for `status`

1.6.2 EAN

class canlib.**EAN**(*source*)

Helper object for dealing with European Article Numbers

Depending on the format the ean is in, *EAN* objects are created in different ways;

For strings:

```
EAN('73-30130-01234-5')
```

For integers:

```
EAN(7330130012345)
```

For iterables of integers:

```
EAN([7, 3, 3, 0, 1, 3, 0, 0, 1, 2, 3, 4, 5])
```

For BCD-coded bytes or bytearrays (str in python 2):

```
EAN.from_bcd(b'\x45\x23\x01\x30\x01\x33\x07')
```

For "hi-lo" format, i.e. two 32-bit integers containing half the ean each, both BCD-coded:

```
EAN.from_hilo([eanHi, eanLo])
```

The various representations can then be produced from the resulting object:

```
>>> str(ean)
'73-30130-01234-5'
>>> int(ean)
7330130012345
>>> tuple(ean)  # or list(), or any other sequence type
(7, 3, 3, 0, 1, 3, 0, 0, 1, 2, 3, 4, 5)
>>> ean.bcd()
b'E#\x010\x013\x07'
>>> ean.hilo()
(805380933, 471809)
```

Sometimes it is easier to only use the last six digits of the ean, the product code and check digit. This is supported when working with string representations; the constructor supports six-digit (seven-character) input:

```
EAN('01234-5')
```

In that cases, the country and manufacturer code is assumed to be that of Kvaser AB (73-30130).

A string containing only the product code and check digit can also be retrieved:

```
ean.product()
```

Instances can also be indexed which yields specific digits as integers:

```
>>> ean[7]
0
>>> ean[7:]
(0, 1, 2, 3, 4, 5)
```

Note: The byteorder is currently always assumed to be 'little'.

bcd()

Return a binary-coded bytes object with this EAN

fmt = '##-#####-#####-#'

classmethod from_bcd(*bcd_bytes*)

Create an EAN object from a binary coded bytes-like object

The EAN is automatically shortened to the correct length.

classmethod from_hilo(*hilo*)

Create an EAN object from a pair of 32-bit integers, (eanHi, eanLo)

classmethod from_string(*ean_string*)

Create an EAN object from a specially formatted string

Deprecated since version 1.6: Use the constructor, `EAN(ean_string)`, instead.

hilo()

Return a pair of 32-bit integers, (eanHi, eanLo), with this EAN

num_digits = 13

product()

Return only the product code and check digit of the string representation

1.6.3 Device

class canlib.**Device**(*ean*, *serial*)

Class for keeping track of a physical device

This class represents a physical device regardless of whether it is currently connected or not, and on which channel it is connected.

If the device is currently connected, *Device.find* can be used to get a *Device* object:

```
dev = Device.find(ean=EAN('67890-1'))
```

Device.find supports searching for devices based on a variety of information, and will return a *Device* object corresponding to the first physical device that matched its arguments. In this case, it would be the first device found with an EAN of 73-30130-67890-1.

If the device wanted is not currently connected, *Device* objects can be created with their EAN and serial number (as this is the minimal information needed to uniquely identify a specific device):

```
dev = Device(ean=EAN('67890-1'), serial=42)
```

Two *Device* objects can be checked for equality (whether they refer to the same device) and be converted to a `str`. *Device.probe_info* can also be used for a more verbose string representation that queries the device (if connected) for various pieces of information.

This class also provides functions for creating the other objects of `canlib`:

- *canlib.Channel* – *Device.channel*
- *canlib.ChannelData* – *Device.channel_data*
- *canlib.IOControl* – *Device.iocontrol*
- *kvmlib.Memorator*– *Device.memorator*
- *linlib.Channel* – *Device.lin_master* and *Device.lin_slave*

Parameters

- **ean** (*canlib.EAN*) – The EAN of this device.
- **serial** (`int`) – The serial number of this device.
- **last_channel_number** (`int`) – The channel number this device was last found on (used as an optimization; while the device stays on the same CANlib channel there is no need for a linear search of all channels).

New in version 1.6.

channel(**args*, ***kwargs*)

A *Channel* for this device's first channel

The experimental argument `_chan_no_on_card` may be given, the `int` provided will be added (without any verifications) to the *channel_number* where this device was found on, and may thus be used to open a specific local channel on this device.

Note: When using the `_chan_no_on_card` attribute, you must make sure that the card actually have the assumed number of local channels. Using this argument with a too large `int` could return a channel belonging to a different device.

Arguments to *openChannel* other than the channel number can be passed to this function.

Changed in version 1.13: Added argument `_chan_no_on_card`

Deprecated since version 1.16: Use *open_channel* instead

channel_data()

A *canlib.ChannelData* for this device's first channel

channel_number()

An `int` with this device's CANlib channel number

ean

classmethod find(*channel_number=None*, *ean=None*, *serial=None*, *channel_name=None*)

Searches for a specific device

Goes through all CANlib channels (from zero and up), until one of them matches the given arguments. If an argument is omitted or `None`, any device will match it. If no devices matches a *canlib.CanNotFound* exception will be raised.

Parameters

- **channel_number** (int) – Find a device on this CANlib channel (number).
- **ean** (*canlib.EAN*) – Find a device with this EAN.
- **serial** (int) – Find a device with this serial number.
- **channel_name** (str) – Find a device with this CANlib channel name.

iocontrol()

A *canlib.IOControl* for this device's first channel

isconnected()

A bool whether this device can currently be found

issubset(*other*)

Check if device is a subset of other Device.

This can be used to see if a found device fulfills criteria specified in other. Setting an attribute to None is regarded as a Any. This means that e.g. any serial number will be a subset of a serial number specified as None.

New in version 1.9.

last_channel_number

lin_master(**args*, ***kwargs*)

A *linlib.Channel* master for this device's first channel

Arguments to *linlib.openMaster* other than the channel number can be passed to this function.

lin_slave(**args*, ***kwargs*)

A *linlib.Channel* slave for this device's first channel

Arguments to *linlib.openSlave* other than the channel number can be passed to this function.

memorator(**args*, ***kwargs*)

A *kvmlib.Memorator* for this device's first channel

Arguments to *kvmlib.openDevice* other than the channel number can be passed to this function.

open_channel(*chan_no_on_card=0*, ***kwargs*)

A *canlib.Channel* for this device's first channel

The parameter chan_no_on_card will be added (without any verifications) to the *channel_number* where this device was found on, and may thus be used to open a specific local channel on this device.

Note: When using the chan_no_on_card parameter, you must make sure that the card actually have the assumed number of local channels. Using this parameter with a too large int could return a channel belonging to a different device.

Arguments to canlib.open_channel, other than the channel number, can be passed to this function, but must be passed as keyword arguments.

New in version 1.16.

probe_info()

A str with information about this device

This function is useful when the *Device*'s str() does not give enough information while debugging. When the device is connected various pieces of information such as the device name, firmware, and driver name is given. When the device is not connected only basic information can be given.

Note: Never inspect the return value of this function, only use it for displaying information. Exactly what is shown depends on whether the device is connected or not, and is not guaranteed to stay consistent between versions.

remote(**args*, ***kwargs*)

A *kvrlib.RemoteDevice* for this device

Arguments to *kvrlib.openDevice* other than the channel number can be passed to this function.

serial

connected_devices()

canlib.**connected_devices**()

Get all currently connected devices as *Device*

Returns an iterator of *Device* object, one object for every physical device currently connected.

New in version 1.6.

Changed in version 1.7: *Device.last_channel_number* will now be set.

1.6.4 Frames

Frame

class canlib.**Frame**(*id_*, *data*, *dlc=None*, *flags=0*, *timestamp=None*)

Represents a CAN message

Parameters

- **id_** – Message id
- **data** – Message data, will pad zero to match dlc (if dlc is given)
- **dlc** – Message dlc, default is calculated from number of data
- **flags** (*canlib.MessageFlag*) – Message flags, default is 0
- **timestamp** – Optional timestamp

data

dlc

flags

id

timestamp

LINFrame

class canlib.**LINFrame**(**args*, ***kwargs*)

Bases: *Frame*

Represents a LIN message

A *Frame* that also has an *info* attribute, which is a *linlib.MessageInfo* or `None`. This attribute is initialized via the *info* keyword-only argument to the constructor.

data

dlc

flags

id

info

timestamp

1.6.5 Version Numbers

VersionNumber

class canlib.**VersionNumber**(*major*, *minor=None*, *build=None*, *release=None*)

A tuple-subclass representing a version number

Version numbers can be created using one to three positional arguments, representing the major, minor, and build number respectively:

```
v1 = VersionNumber(1)
v12 = VersionNumber(1, 2)
v123 = VersionNumber(1, 2, 3)
```

Keyword arguments can also be used:

```
v1 = VersionNumber(major=1)
v12 = VersionNumber(major=1, minor=2)
v123 = VersionNumber(major=1, minor=2, build=3)
```

A fourth number, the release number, can also be given as a keyword-only argument:

```
v1293 = VersionNumber(major=1, minor=2, release=9, build=3)
```

This release number is placed between the minor and build numbers, both for the string representation and in the tuple.

The major number is required and the other numbers are optional in the order minor, build, release.

All numbers can be accessed as attributes (*major*, *minor*, *release*, *build*). If the number is unavailable, accessing the attribute returns `None`.

property beta

property build

property major

property minor

property release

BetaVersionNumber

class canlib.**BetaVersionNumber**(*major*, *minor=None*, *build=None*, *release=None*)

A tuple-subclass representing a beta (preview) version number

A *VersionNumber* that also has the attribute *beta* set to true.

New in version 1.6.

property beta

1.6.6 canlib

Exceptions

CanError

exception canlib.canlib.**CanError**

Bases: *DllException*

Base class for exceptions raised by the canlib class

Looks up the error text in the canlib dll and presents it together with the error code and the wrapper function that triggered the exception.

CanGeneralError

exception canlib.canlib.exceptions.**CanGeneralError**(*status*)

Bases: *CanError*

A canlib error that does not (yet) have its own Exception

> **Warning:** Do not explicitly catch this error, instead catch *CanError*. Your error may at any point in the future get its own exception class, and so will no longer be of this type. The actual status code that raised any *CanError* can always be accessed through a `status` attribute.

CanInvalidHandle

exception canlib.canlib.**CanInvalidHandle**

Bases: *CanError*

CANlib handle is invalid.

Raised e.g. when trying to access a CAN channel via a closed channel handle.

New in version 1.22.

status = -10

CanNoMsg

exception canlib.canlib.**CanNoMsg**

Bases: *CanError*

Raised when no matching message was available

status = -2

CanNotFound

exception canlib.canlib.**CanNotFound**

Bases: *CanError*

Specified device or channel not found

There is no hardware available that matches the given search criteria. For example, you may have specified *Open.REQUIRE_EXTENDED* but there's no controller capable of extended CAN. You may have specified a channel number that is out of the range for the hardware in question. You may have requested exclusive access to a channel, but the channel is already occupied.

New in version 1.6.

status = -3

CanOutOfMemory

exception canlib.canlib.**CanOutOfMemory**

Bases: *CanError*

A memory allocation failed.

Raised e.g. when trying to allocate an object buffer, when all available object buffers have already been allocated.

New in version 1.22.

status = -4

CanScriptFail

exception canlib.canlib.**CanScriptFail**

Bases: *CanError*

Raised when a script call failed.

This exception represents several different failures, for example:

- Trying to load a corrupt file or not a .txe file
- Trying to start a t script that has not been loaded
- Trying to load a t script compiled with the wrong version of the t compiler
- Trying to unload a t script that has not been stopped
- Trying to use an envvar that does not exist

status = -39

CanTimeout

exception canlib.canlib.**CanTimeout**

Bases: *CanError*

Raised when a timeout occured

Raised when an expected event did not happen within the expected time frame, see e.g. *canlib.Channel.writeWait()*.

New in version 1.21.

status = -7

EnvvarException

exception canlib.canlib.**EnvvarException**

Bases: *CanlibException*

Base class for exceptions related to environment variables.

EnvvarNameError

exception canlib.canlib.**EnvvarNameError**(*envvar*)

Bases: *EnvvarException*

Raised when the name of the environment variable is illegal.

EnvvarValueError

exception canlib.canlib.**EnvvarValueError**(*envvar*, *type_*, *value*)

Bases: *EnvvarException*

Raised when the type of the value does not match the type of the environment variable.

IoNoValidConfiguration

exception canlib.canlib.**IoNoValidConfiguration**

Bases: *CanError*

I/O pin configuration is invalid

No I/O pins was found, or unknown I/O pins was discovered.

New in version 1.8.

status = -48

IoPinConfigurationNotConfirmed

exception canlib.canlib.**IoPinConfigurationNotConfirmed**

Bases: *CanError*

I/O pin configuration is not confirmed

Before accessing any I/O pin value, the device I/O pin configuration must be confirmed, using e.g. *Channel.io_confirm_config*.

See also *iopin.Configuration*.

New in version 1.8.

status = -45

TxeFileIsEncrypted

exception canlib.canlib.**TxeFileIsEncrypted**

Bases: *CanlibException*

Raised when trying to access *Txe.source* and the source and byte-code sections of the .txe binary have been encrypted.

New in version 1.6.

Bus Parameters

calc_bitrate()

canlib.canlib.busparams.**calc_bitrate**(*target_bitrate*, *clk_freq*)

Calculate nearest available bitrate

Parameters

- **target_bitrate** (int) – Wanted bitrate (bit/s)
- **clk_freq** (int) – Device clock frequency (Hz)

Returns

The returned tuple is a `(bitrate, tq)` named tuple of –

1. `bitrate` (int): Available bitrate, could be a rounded value (bit/s)
2. `tq` (int): Number of time quanta in one bit

New in version 1.16.

calc_busparamstq()

canlib.canlib.busparams.**calc_busparamstq**(*target_bitrate*, *target_sample_point*, *target_sync_jump_width*, *clk_freq*, *target_prop_tq=None*, *prescaler=1*)

Calculate closest matching busparameters.

The device clock frequency, `clk_freq`, can be obtained via *ClockInfo.frequency()*:

```
>>> chd = canlib.ChannelData(channel_number=0)
>>> clock_info = chd.clock_info
>>> clock_info.frequency()
80000000.0
```

Now call *calc_busparamstq* with target values, and a *BusParamsTq* object will be returned:

```
>>> params = calc_busparamstq(
... target_bitrate=470_000,
... target_sample_point=82,
... target_sync_jump_width=15.3,
... clk_freq=clock_info.frequency())
>>> params
BusParamsTq(tq=170, prop=107, phase1=31, phase2=31, sjw=26, prescaler=1)
```

A target number of time quanta in the propagation segment can also be specified by the user.

The returned BusParamsTq may not be valid on all devices. If *Error.NOT_IMPLEMENTED* is encountered when trying to set the bitrate with the returned BusParamsTq, provide a prescaler argument higher than one and retry. This will lower the total number of time quanta in the bit and thus make the BusParamsTq valid.

Example

```
>>> params = calc_busparamstq(
... target_bitrate=470_000,
... target_sample_point=82,
... target_sync_jump_width=15.3,
... clk_freq=clock_info.frequency(),
... target_prop_tq=50,
... prescaler=2)
>>> params
BusParamsTq(tq=85, prop=25, phase1=44, phase2=15, sjw=13, prescaler=2)
```

Note:

- Minimum sjw returned is 1, maximum sjw is min(phase1, phase2).

Parameters

- **target_bitrate** (`float`) – Wanted bitrate (bit/s)
- **target_sample_point** (`float`) – Wanted sample point in percentage (0-100)
- **target_sync_jump_width** (`float`) – Wanted sync jump width in percentage (0-100)
- **clk_freq** (`float`) – Device clock frequency (Hz)
- **target_prop_tq** (`int`, Optional) – Wanted propagation segment (time quanta)
- **prescaler** (`int`, Optional) – Wanted prescaler (at most 2 for CAN FD)

Returns

BusParamsTq – Calculated bus parameters

New in version 1.16.

Changed in version 1.17.

calc_sjw()

canlib.canlib.busparams.**calc_sjw**(*tq*, *target_sync_jump_width*)

Calculate sync jump width

tq: Number of time quanta in one bit target_sync_jump_width: Wanted sync jump width, 0-100 (%)

Note: Minimum sjw_tq returned is 1.

Returns

The returned named tuple is a `(sjw_tq, sync_jump_width)` consisting of –

1. `sjw_tq` (`int`): Size of sync jump width in number of time quanta,
2. `sync_jump_width` (`number`): Size of sync jump width in percentage (%))

New in version 1.16.

to_BusParamsTq()

canlib.canlib.busparams.**to_BusParamsTq**(*clk_freq*, *bus_param*, *prescaler=1*, *data=False*)

Convert *BitrateSetting* or `tuple` to *BusParamsTq*.

The device clock frequency, `clk_freq`, can be obtained via *ClockInfo.frequency()*:

```
>>> chd = canlib.ChannelData(channel_number=0)
>>> clock_info = chd.clock_info
>>> clock_info.frequency()
80000000.0
```

Parameters

- **clk_freq** (`float`) – Clock frequency of device.
- **bus_param** (*BitrateSetting* or `tuple`) – *BitrateSetting* object or
- **(freq, tseg1, tseg2, sjw) `tuple` to convert.**
- **prescaler** (`int`) – The prescaler to use in the created *BusParamsTq*
- **object.**
- **data** (`bool`) – Set to True if the resulting *BusParamsTq* should be
- **used for CAN FD data bitrate parameters.**

Returns

BusParamsTq object with equivalent settings as the input argument.

New in version 1.17.

to_BitrateSetting()

canlib.canlib.busparams.**to_BitrateSetting**(*clk_freq*, *bus_param*)

Convert *BusParamsTq* to *BitrateSetting*.

The device clock frequency, `clk_freq`, can be obtained via *ClockInfo.frequency()*:

```
>>> chd = canlib.ChannelData(channel_number=0)
>>> clock_info = chd.clock_info
>>> clock_info.frequency()
80000000.0
```

Parameters

- **clk_freq** (`float`) – Clock frequency of device.
- **bus_param** (*BusParamsTq*) – *BusParamsTq* object to convert.

Returns

BitrateSetting object with equivalent settings as the input argument.

New in version 1.17.

ClockInfo

class canlib.canlib.busparams.**ClockInfo**(*numerator*, *denominator*, *power_of_ten*, *accuracy*)

Information about clock a oscillator

The clock frequency is set in the form:

frequency = numerator / denominator * 10 ** power_of_ten +/- accuracy

New in version 1.16.

frequency()

Returns an approximation of the clock frequency as a float.

BusParamsTq

class canlib.canlib.busparams.**BusParamsTq**(*tq*, *phase1*, *phase2*, *sjw*, *prescaler=1*, *prop=None*)

Holds parameters for busparameters in number of time quanta.

If you don't want to specify the busparameters in time quanta directly, you may use *calc_busparamstq* which returns an object of this class.

```
>>> params = calc_busparamstq(
... target_bitrate=470_000,
... target_sample_point=82,
... target_sync_jump_width=33.5,
... clk_freq=clk_freq)
>>> params
BusParamsTq(tq=170, prop=107, phase1=31, phase2=31, sjw=57, prescaler=1)
```

You may now query for the actual Sample Point and Sync Jump Width expressed as percentages of total bit time quanta:

```
>>> params.sample_point()
81.76470588235294
```

```
>>> params.sync_jump_width()
33.52941176470588
```

If you supply the clock frequency, you may also calculate the corresponding bitrate:

```
>>> params.bitrate(clk_freq=80_000_000)
470588.23529411765
```

Parameters

- **tq** (int) – Number of time quanta in one bit.
- **phase1** (int) – Number of time quanta in Phase Segment 1.
- **phase2** (int) – Number of time quanta in Phase Segment 2.
- **sjw** (int) – Number of time quanta in Sync Jump Width.
- **prescaler** (int) – Prescaler value (1-2 to enable auto in CAN FD)
- **prop** (int, optional) – Number of time quanta in Propagation Segment.

New in version 1.16.

bitrate(*clk_freq*)

Return bitrate assuming the given clock frequency

Parameters
clk_freq (int) – Clock frequency (in Hz)

sample_point()

Return sample point in percentage.

sample_point_ns(*clk_freq*)

Return sample point in ns.

New in version 1.17.

sync_jump_width()

Return sync jump width (SJW) in percentage.

BusParamTqLimits

class canlib.canlib.busparams.**BusParamTqLimits**(*arbitration_min*, *arbitration_max*, *data_min*, *data_max*)

Hold min and max values for both arbitration and data phase.

The `tq` field is ignored during validation since *ChannelData.bus_param_limits* always returns zero for this field.

If `prop` is zero for both `min` and `max` values, the `phase1` limit applies to (`phase1` + `prop`). This is used when a device does not distinguish between phase segment one and the propagation segment.

Example usage:

```
>>> ch = canlib.openChannel(channel=0)
>>> limits = canlib.ChannelData(channel_number=0).bus_param_limits
>>> limits.arbitration_max._asdict()
{'tq': 0, 'phase1': 32, 'phase2': 32, 'sjw': 32, 'prescaler': 1024, 'prop': 64}
>>> bp_tq = canlib.busparams.BusParamsTq(tq=121, phase1=100, phase2=10, sjw=10,
... prescaler=10, prop=10)
>>> limits.validate(bp_tq)
ValueError: The following does not match:
  Arbitration phase1: 1 <= 100 <= 32
```

NOTE: This class is preliminary and may change!

New in version 1.20.

validate(*bus_param*, *data_param=None*)

Validates busparameters for arbitration and data

Raises a `ValueError` if busparameters for arbritation and data is not within current limits. The failed validation is provided as an explanation:

```
ValueError: The following does not match:
  Arbitration phase1: 11 <= 1 <= 21
```

BitrateSetting

class canlib.canlib.busparams.**BitrateSetting**(*freq*, *tseg1*, *tseg2*, *sjw*, *nosamp=1*, *syncMode=0*)

Class that holds bitrate setting.

Parameters

- **freq** – Bitrate in bit/s.
- **tseg1** – Number of quanta from (but not including) the Sync Segment to the sampling point.
- **tseg2** – Number of quanta from the sampling point to the end of the bit.
- **sjw** – The Synchronization Jump Width.
- **nosamp** – The number of sampling points, only 1 is supported.
- **syncMode** – Unsupported and ignored.

New in version 1.17.

classmethod from_predefined(*bitrate*)

Create a BitrateSetting object using one of the *Bitrate* or *BitrateFD* enumerations.

Channel

openChannel()

canlib.canlib.**openChannel**(*channel*, *flags=0*, *bitrate=None*, *data_bitrate=None*)

Open CAN channel

Retrieves a *Channel* object for the given CANlib channel number using the supplied flags.

Example usage:

```
>>> bitrate = canlib.busparams.BusParamsTq(
... tq=40,
... phase1=5,
... phase2=6,
... sjw=4,
... prescaler=2,
... prop=28,
... )
>>> data_bitrate = canlib.busparams.BusParamsTq(
... tq=40,
... phase1=31,
... phase2=8,
... sjw=2,
... prescaler=2,
... )
>>> ch = canlib.openChannel(
... channel=0,
... flags=canlib.Open.CAN_FD,
... bitrate=bitrate,
... data_bitrate=data_bitrate,
... )
```

Note: If both `bitrate` and `data_bitrate` is given, both must be of the same type, i.e. either BusParamsTq or BITRATE flags. It is not supported to mix the two.

Parameters

- **channel** (`int`) – CANlib channel number
- **flags** (`int`) – Flags, a combination of the *Open* flag values. Default is zero, i.e. no flags.
- **bitrate** (*BusParamsTq* or *Bitrate* or *BitrateFD*) – The desired bitrate. If the bitrate is not a *BusParamsTq*, the predefined *Bitrate* values are used for classic CAN and *BitrateFD* values are used for CAN FD, e.g. `canlib.Bitrate.BITRATE_1M` and `canlib.BitrateFD.BITRATE_1M_80P`. For CAN FD, this parameter gives the arbitration bitrate.
- **data_bitrate** (*BusParamsTq* or *BitrateFD*) – The desired data bitrate for CAN FD. If not *BusParamsTq* is given, the predefined *BitrateFD* values are used, e.g. `canlib.BitrateFD.BITRATE_500K_80p`. This parameter is only used when opening a CAN FD channel.

Returns

A *Channel* object created with channel and flags

New in version 1.6: The *bitrate* and *data_bitrate* arguments was added.

Changed in version 1.16: The *bitrate* and *data_bitrate* arguments now accept *BusParamsTq* objects.

Changed in version 1.17: Added support for *Bitrate* and *BitrateFD* enumerations.

ErrorCounters

class canlib.canlib.channel.**ErrorCounters**(*tx*, *rx*, *overrun*)

Error counters returned by *Channel.read_error_counters*.

property overrun

Alias for field number 2

property rx

Alias for field number 1

property tx

Alias for field number 0

Channel

class canlib.canlib.**Channel**(*channel_number*, *flags=0*)

Helper class that represents a CANlib channel.

This class wraps the canlib class and tries to implement a more Pythonic interface to CANlib.

Channels are automatically closed on garbage collection, and can also be used as context managers in which case they close as soon as the context exits.

Variables

envvar (*EnvVar*) – Used to access *t* program environment variables

allocate_periodic_objbuf(*period_us*, *frame*)

Allocate a periodic object buffer (a.k.a. auto transmit buffer).

Call *objbuf.Periodic.enable()* in order to start the periodic transmissions of the CAN frame.

Parameters

- **period_us** (int) – Interval in microseconds between each sent CAN frame.
- **frame** (*Frame*) – The CAN frame to send.

Returns

objbuf.Periodic – A periodic object buffer.

New in version 1.22.

allocate_response_objbuf(*filter*, *frame*, *rtr_only=False*)

Allocate autoresponse object buffer.

Parameters

- **frame** (*Frame*) – The CAN frame to send.
- **filter** (*objbuf.MessageFilter*) – Messages not matching the filter is ignored.
- **rtr_only** (bool) – If True, only trigger response on remote requests.

Returns

objbuf.Response – A response object buffer.

New in version 1.22.

announceIdentityEx(*item*, *value*)

AnnounceIdentityEx function is used by certain OEM applications.

New in version 1.17.

bitrate_to_BusParamsTq(*freq_a*, *freq_d=None*)

Calculate bus parameters based on predefined frequencies.

This function uses the *Bitrate* and *BitrateFD* values to create bus parameters for use with the new bus parameter API.

Parameters

- **freq_a** (*Bitrate* or *BitrateFD*) – The bitrate for classic CAN channels specified as a *Bitrate*, or the arbitration bitrate for CAN FD channels specified as a *BitrateFD*.
- **freq_d** (*BitrateFD*) – Data bitrate for CAN FD channels.

Returns

(nominal, None) for classic CAN, where nominal is a *BusParamsTq* object.

(nominal, data) for CAN FD, where both nominal and data are *BusParamsTq* objects, representing the arbitration bitrate and the data bitrate, respectively.

New in version 1.17.

busOff()

Takes the specified channel off-bus.

Closes the channel associated with the handle. If no other threads are using the CAN circuit, it is taken off bus. The handle can not be used for further references to the channel.

busOn()

Takes the specified channel on-bus.

If you are using multiple handles to the same physical channel, for example if you are implementing a multithreaded application, you must call busOn() once for each handle.

canAccept(*envelope*, *flag*)

Set acceptance filters mask or code.

This routine sets the message acceptance filters on a CAN channel.

Setting flag to *AcceptFilterFlag.NULL_MASK* (0) removes the filter.

Note that not all CAN boards support different masks for standard and extended CAN identifiers.

Parameters

- **envelope** – The mask or code to set.
- **flag** – Any of *AcceptFilterFlag*

canSetAcceptanceFilter(*code*, *mask*, *is_extended=False*)

Set message acceptance filter.

This routine sets the message acceptance filters on a CAN channel. The message is accepted `if id AND mask == code` (this is actually imlepemented as `if ((code XOR id) AND mask) == 0`).

Using standard 11-bit CAN identifiers and setting

- mask = 0x7f0,
- code = 0x080

accepts CAN messages with standard id 0x080 to 0x08f.

Setting the *mask* to canFILTER_NULL_MASK (0) removes the filter.

Note that not all CAN boards support different masks for standard and extended CAN identifiers.

Parameters

- **mask** (`int`) – A bit mask that indicates relevant bits with '1'.
- **code** (`int`) – The expected state of the masked bits.
- **is_extended** (`bool`) – If `True`, both mask and code applies to 29-bit CAN identifiers.

property channel_data

`canGetHandleData` helper object for this channel

See the documentation for *ChannelData*/*HandleData* for how it can be used to perform all functionality of the C function `canGetHandleData`.

New in version 1.13.

Type
HandleData

close()

Close CANlib channel

Closes the channel associated with the handle. If no other threads are using the CAN circuit, it is taken off bus.

Note: It is normally not necessary to call this function directly, as the internal handle is automatically closed when the *Channel* object is garbage collected.

device()

Get a Device object from the current channel

Returns
Device – Device used by this channel,

New in version 1.16.

fileCopyFromDevice(*deviceFileName*, *hostFileName=None*)

Copy an arbitrary file from the device to the host.

Parameters

- **deviceFileName** (str) – The device file name.
- **hostFileName** (str, optional) – The target host file name. Defaults to deviceFileName.

fileCopyToDevice(*hostFileName*, *deviceFileName=None*)

Copy an arbitrary file from the host to the () isdevice.

The filename must adhere to the FAT '8.3' naming standard, max 8 characters - a dot - max 3 characters.

Parameters

- **hostFileName** (str) – The target host file name.
- **deviceFileName** (str, optional) – The device file name. Defaults to the same as *hostFileName*.

fileDelete(*deviceFileName*)

Delete file from device.

fileDiskFormat()

Format the disk on the device, not supported by all devices.

New in version 1.11.

fileGetCount()

Get the number of files on the device.

Returns
int – The number of files.

fileGetName(*fileNo*)

Get the name of the file with the supplied number.

Parameters
fileNo (int) – The number of the file.

Returns
str – The name of the file.

flashLeds(*action*, *timeout_ms*)

Turn Leds on or off.

Parameters

- **action** (int) – One of *LEDAction*, defining which LED to turn on or off.

- **timeout_ms** (int) – Specifies the time, in milliseconds, during which the action is to be carried out. When the timeout expires, the LED(s) will return to its ordinary function.

free_objbuf()

Free all allocated object buffers.

Free all object buffers allocated via *allocate_periodic_objbuf()* or *allocate_response_objbuf()*.

New in version 1.22.

getBusOutputControl()

Get driver type

This function retrieves the current CAN controller driver type. This corresponds loosely to the bus output control register in the CAN controller, hence the name of this function.

Note: Not all CAN driver types are supported on all cards.

Returns
drivertype (*canlib.Driver*) – Driver type to set.

New in version 1.11.

getBusParams()

Get bus timing parameters for classic CAN.

This function retrieves the current bus parameters for the specified channel.

Returns: A tuple containing:
freq: Bitrate in bit/s.

tseg1: Number of quanta from but not including the Sync Segment to the sampling point.

tseg2: Number of quanta from the sampling point to the end of the bit.

sjw: The Synchronization Jump Width, can be 1, 2, 3, or 4.

noSamp: The number of sampling points, only 1 is supported.

syncmode: Unsupported, always read as zero.

getBusParamsFd()

Get bus timing parameters for BRS in CAN FD.

This function retrieves the bus current timing parameters used in BRS (Bit rate switch) mode for the current CANlib channel.

The library provides default values for tseg1_brs, tseg2_brs and sjw_brs when freq is a *BitrateFD* value.

If freq is any other value, no default values are supplied by the library.

For finding out if a channel was opened as CAN FD, use *is_can_fd()*

Returns: A tuple containing:
freq_brs: Bitrate in bit/s.

tseg1_brs: Number of quanta from (but not including) the Sync Segment to the sampling point.

tseg2_brs: Number of quanta from the sampling point to the end of the bit.

sjw_brs: The Synchronization Jump Width.

getChannelData_CardNumber()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.card_number`

getChannelData_Chan_No_On_Card()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.chan_no_on_card`

getChannelData_Cust_Name()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.custom_name`

getChannelData_DriverName()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.driver_name`

getChannelData_EAN()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.card_upc_no`

getChannelData_EAN_short()

getChannelData_Firmware()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.card_firmware_rev`

getChannelData_Name()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.device_name`

getChannelData_Serial()

Deprecated function

Deprecated since version 1.5: Use *ChannelData*; `ChannelData.card_serial_no`

get_bus_params_tq()

Get bus timing parameters, in time quanta

This function retrieves the current bus parameters for the specified channel. Only the returned `nominal` parameter is valid when classic CAN is in use.

Returns: A `tuple` containing:

nominal: (*BusParamsTq*) Nominal bus timing parameters, also used in classic CAN.

data: (*BusParamsTq*) Bus timing parameters for data rate in CAN FD.

New in version 1.16.

get_bus_statistics()

Return bus statistics.

Returns
structures.CanBusStatistics

get_io_pin(*index*)

Return I/O pin using *index*

Returns
iopin.IoPin – io pin object for *index* (Any of *iopin.AnalogIn*, *iopin.DigitalOut* etc)

New in version 1.8.

ioCtl_flush_rx_buffer()

Deprecated function

Deprecated since version 1.5: Use *IOControl*; `Channel.iocontrol.flush_rx_buffer()`.

ioCtl_get_report_access_errors()

Deprecated function

Deprecated since version 1.5: Use *IOControl*; `Channel.iocontrol.report_access_errors`

ioCtl_set_report_access_errors(*on=0*)

Deprecated function

Deprecated since version 1.5: Use *IOControl*; `Channel.iocontrol.report_access_errors = True`

ioCtl_set_timer_scale(*scale*)

Deprecated function

Deprecated since version 1.5: Use *IOControl*; `Channel.iocontrol.timer_scale = scale`

io_confirm_config()

Confirm current I/O configuration

It is required to confirm a configuration by calling this function before accessing any I/O pins value.

New in version 1.8.

io_pins()

Generator that returns all I/O pins one by one

Returns object depending on pin type and direction: *iopin.AnalogIn*, *iopin.AnalogOut*, *iopin.DigitalIn*, *iopin.DigitalOut* or *iopin.Relay*.

New in version 1.8.

property iocontrol

`canIoCtl` helper object for this channel

See the documentation for *IOControl* for how it can be used to perform all functionality of the C function `canIoCtl`.

Example usage:

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> ch.iocontrol.timer_scale
1000
>>> ch.iocontrol.local_txecho = False
```

Type
IOControl

is_can_fd()

Return `True` if the channel has been opened with the *CAN_FD* or *CAN_FD_NONISO* flags.

Returns
`True` if CAN FD, `False` otherwise.

kvDeviceGetMode()

Read the current device's mode.

Note: The mode is device specific, which means that not all modes are implemented in all products.

Returns
`int` – One of *DeviceMode*, indicating which mode is in use.

kvDeviceSetMode(*mode*)

Set the current device's mode.

Note: The mode is device specific, which means that not all modes are implemented in all products.

Parameters
mode (`int`) – One of *DeviceMode*, defining which mode to use.

number_of_io_pins()

Return the total number of available I/O pins

New in version 1.8.

read(*timeout=0*)

Read a CAN message and metadata.

Reads a message from the receive buffer. If no message is available, the function waits until a message arrives or a timeout occurs.

The unit of the returned *Frame.timestamp* is configurable using `Channel.iocontrol.timer_scale`, default is 1 ms.

Note: If you are using the same channel via multiple handles, the default behaviour is that the different handles will "hear" each other just as if each handle referred to a channel of its own. If you open, say, channel 0 from thread A and thread B and then send a message from thread A, it will be "received" by thread B.

This behaviour can be changed by setting `local_txecho` to `False` (using *IOControl*):

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> ch.iocontrol.local_txecho = False
```

Parameters
timeout (`int`) – Timeout in milliseconds, -1 gives an infinite timeout.

Returns
canlib.Frame

Raises
CanNoMsg – No CAN message is currently available.

readDeviceCustomerData(*userNumber=100*, *itemNumber=0*)

Read customer data stored in device

readSpecificSkip(*id_*)

Read a message with specified identifier

Reads a message with a specified identifier from the receive buffer. Any preceding message not matching the specified identifier will be removed in the receive buffer. If no message with the specified identifier is available, the function returns immediately with an error code.

The unit of the returned *Frame.timestamp* is configurable using `Channel.iocontrol.timer_scale`, default is 1 ms.

Returns
canlib.Frame

readStatus()

Return status for the current channel

Returns the latest status reported by the hardware in a combination of the flags *Stat* (bus on/error passive + status etc).

Returns
canlib.canlib.Stat

readSyncSpecific(*id_*, *timeout=0*)

Wait until the receive queue contains a message with the specified id

readTimer()

Read the hardware clock on the specified device

Returns the time value.

read_error_counters()

Read the error counters of the CAN controller

Returns the latest known values of the error counters in the specified circuit. If the error counters change values precisely when *read_error_counters* is called, it may not be reflected in the returned result.

Use *clear_error_counters* via *Channel.iocontrol* to clear the counters.

Returns

The returned tuple is a `(rx, tx, overrun)` named tuple of –

1. `rx` (`int`): Receive error counter
2. `tx` (`int`): Transmit error counter
3. `overrun` (`int`): Number of overrun errors.

New in version 1.11.

requestChipStatus()

Request chip status messages

Requests that the hardware report the chip status (bus on/error passive status etc.) to the driver. The chip status can later be retrieved using `canlib.Channel.readStatus`.

Note: The *requestChipStatus* function is asynchronous, that is, it completes before the answer is returned from the hardware. The time between a call to *requestChipStatus* and the point in time where the chip status is actually available via a call to *Channel.readStatus* is not defined. The *Channel.readStatus* always returns the latest data reported by the hardware.

scriptEnvvarClose(*envHandle*)

Low level function to close an Envvar

This should normally not be used directly, instead opening and closing of an envvar is automatically done when accessing via the *EnvVar* class through `Channel.envvar`

scriptEnvvarGetData(*envHandle*, *envSize*)

scriptEnvvarGetFloat(*envHandle*)

Low level function to read an Envvar of type float

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

scriptEnvvarGetInt(*envHandle*)

Low level function to read an Envvar of type int

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

scriptEnvvarOpen(*name*)

Low level function to open an Envvar

This should normally not be used directly, instead opening and closing of an envvar is automatically done when accessing via the *EnvVar* class through `Channel.envvar`

scriptEnvvarSetData(*envHandle*, *value*, *envSize*)

scriptEnvvarSetFloat(*envHandle*, *value*)

Low level function to set an Envvar of type float

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

scriptEnvvarSetInt(*envHandle*, *value*)

Low level function to set an Envvar of type int

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

scriptGetText()

Read text from subscribed script slots

Text-subscriptions must first be set up with *Channel.scriptRequestText*.

Returns
ScriptText

Raises
CanNoMsg – No more text is currently available.

New in version 1.7.

`scriptLoadFile`(*slot*, *filePathOnPC*)

Load compiled script file from host(PC)

Loads a compiled script file (.txe) stored on the host (PC) into a script slot on the device. The scripts default channel will be the same channel used when this Channel object was created.

Parameters

- **slot** (`int`) – slot containing the running script we want to stop.
- **filePathOnPC** (`str`) – Path to compiled script (.txe) to load.

`scriptLoadFileOnDevice`(*slot*, *localFile*)

Load compiled, locally stored, script file

Loads a compiled script file (.txe) stored locally on the device (SD card) into a script slot on the device. The scripts default channel will be the same channel used when this Channel object was created.

Parameters

- **slot** (`int`) – slot containing the running script we want to stop.
- **localFile** (`str`) – Name of compiled script (.txe) to load.

`scriptRequestText`(*slot*, *request=ScriptRequest.SUBSCRIBE*)

Set up a printf subscription to a selected script slot

Parameters

- **slot** (`int`) – The script slot to subscribe/unsubscribe from.
- **request** (*ScriptRequest*) – Whether to subscribe or unsubscribe.

Text printed with `printf()` by a t-program that you are subscribed to is saved and can be retrieved with *Channel.scriptGetText*.

New in version 1.7.

`scriptSendEvent`(*slotNo=0*, *eventType=1*, *eventNo=None*, *data=0*)

Send specified event to specified t script

Send an event with given type, event number, and associated data to a script running in a specific slot.

`scriptStart`(*slot*)

Start previously loaded script in specified slot

`scriptStatus`(*slot*)

Retreives t program status for selected slot

Parameters
slot (`int`) – Slot number to be queried

Returns
`canlib.ScriptStatus`

New in version 1.6.

`scriptStop`(*slot*, *mode=ScriptStop.NORMAL*)

Stop script running in specified slot

Parameters

- **slot** (`int`) – slot containing the running script we want to stop.
- **mode** (`canlib.ScriptStop`) – Default mode is `canlib.ScriptStop.NORMAL`.

scriptUnload(*slot*)

Unload previously stopped script in specified slot

script_envvar_get_data(*envHandle*, *len*, *start=0*)

Low level function to read a slice of an Envvar of type char

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

New in version 1.10.

script_envvar_set_data(*envHandle*, *value*, *len*, *start=0*)

Low level function to write a slice of an Envvar of type char

This should normally not be used directly, instead set and get the value of an envvar using the *EnvVar* class through `Channel.envvar`

`value` needs to be a bytes-like object or list

New in version 1.10.

setBusOutputControl(*drivertype=Driver.NORMAL*)

Set driver type

This function sets the driver type for a CAN controller to e.g. silent mode. This corresponds loosely to the bus output control register in the CAN controller, hence the name of this function.

Note: Not all CAN driver types are supported on all cards.

Parameters

drivertype (*canlib.Driver*) – Driver type to set.

setBusParams(*freq*, *tseg1=0*, *tseg2=0*, *sjw=0*, *noSamp=0*, *syncmode=0*)

Set bus timing parameters for classic CAN

This function sets the bus timing parameters for the specified CAN controller.

The library provides default values for tseg1, tseg2, sjw and noSamp when freq is a *Bitrate*, e.g. *Bitrate.BITRATE_1M*.

If freq is any other value, no default values are supplied by the library.

If you are using multiple handles to the same physical channel, for example if you are implementing a multithreaded application, you must call *busOn()* once for each handle. The same applies to *busOff()* - the physical channel will not go off bus until the last handle to the channel goes off bus.

Parameters

- **freq** – Bitrate in bit/s.
- **tseg1** – Number of quanta from (but not including) the Sync Segment to the sampling point.
- **tseg2** – Number of quanta from the sampling point to the end of the bit.
- **sjw** – The Synchronization Jump Width, can be 1, 2, 3, or 4.
- **nosamp** – The number of sampling points, only 1 is supported.
- **syncMode** – Unsupported and ignored.

Changed in version 1.17: Now accepts *Bitrate* enumerations.

setBusParamsFd(*freq_brs*, *tseg1_brs=0*, *tseg2_brs=0*, *sjw_brs=0*)

Set bus timing parameters for BRS in CAN FD

This function sets the bus timing parameters used in BRS (Bit rate switch) mode for the current CANlib channel.

The library provides default values for tseg1_brs, tseg2_brs and sjw_brs when freq is a *BitrateFD* value, e.g. *BitrateFD.BITRATE_1M_80P*.

If freq is any other value, no default values are supplied by the library.

For finding out if a channel was opened as CAN FD, use *is_can_fd()*

Parameters

- **freq_brs** – Bitrate in bit/s.
- **tseg1_brs** – Number of quanta from (but not including) the Sync Segment to the sampling point.
- **tseg2_brs** – Number of quanta from the sampling point to the end of the bit.
- **sjw_brs** – The Synchronization Jump Width.

Changed in version 1.17: Now accepts *BitrateFD* enumerations.

set_bus_params_tq(*nominal*, *data=None*)

Set bus timing parameters, using time quanta

This function sets the bus timing parameters for the specified CAN controller. When setting bus parameters for CAN FD, both `nominal` and `data` must be given.

If you are using multiple handles to the same physical channel, for example if you are implementing a multithreaded application, you must call *busOff()* once for each handle. The physical channel will not go off bus until the last handle to the channel goes off bus. The same applies to *busOn()*.

Parameters

- **nominal** (*BusParamsTq*) – Nominal Bus timing parameters, also used for classic CAN
- **data** (*BusParamsTq*) – Bus timing parameters for data rate in CAN FD.

New in version 1.16.

set_callback(*function*, *event*, *context=None*)

Register callback function

This will register a callback function which is called when certain events occur. You can register at most one callback function per handle at any time.

Note:

The callback function is called in the context of a high-priority thread created by CANlib. You should take precaution not to do any time consuming tasks in the callback.

Small example of usage:

```
# Declare callback function
def callback_func(hnd, context, event):
    event = canlib.Notify(event)

    # The nonlocal statement causes the listed identifiers to refer
    # to previously bound variables in the nearest enclosing scope
    # excluding globals.
```

(continues on next page)

(continued from previous page)

```
    nonlocal callback_has_been_called
    print("Callback called, context:{}, event:{!r}".format(context, event))
    # Notify the main program by setting the flag
    callback_has_been_called = True

# setup communication variable and callback
callback_has_been_called = False
callback = canlib.dll.KVCALLBACK_T(callback_func)

with canlib.openChannel(0) as ch:
ch.set_callback(callback, context=121, event=canlib.Notify.BUSONOFF)
# trigger the callback
ch.busOn()
# do something else
time.sleep(0.5)
# Verify that the callback was triggered
assert callback_has_been_called
```

Note:

It is very important to make sure that you keep a reference to the callback type (`callback` in the sample above) for as long as any C library might call it. If it gets deleted by the garbage collector, calling it from C is likely to either cause a segfault or maybe even interpret random memory as machine language.

Parameters

- **function** (`KVCALLBACK_T`) – A ctypes wrapped Python function
- **event** (*Notify*) – A combination of flags to indicate what events to trigger on

New in version 1.7.

write(*frame=None*, **args*, ***kwargs*)

Send a CAN message.

This function sends a Frame object as a CAN message. Note that the message has been queued for transmission when this calls return. It has not necessarily been sent.

Note: If you are using the same channel via multiple handles, the default behaviour is that the different handles will "hear" each other just as if each handle referred to a channel of its own. If you open, say, channel 0 from thread A and thread B and then send a message from thread A, it will be "received" by thread B.

This behaviour can be changed by setting `local_txecho` to `False` (using *IOControl*):

```
>>> from canlib import canlib
>>> ch = canlib.openChannel(channel=0)
>>> ch.iocontrol.local_txecho = False
```

Also see *Channel.write_raw* for sending messages without constructing *canlib.Frame* objects.

Deprecated since version 1.5: Sending the *canlib.Frame* contents as separate arguments; this functionality has been taken over by *write_raw*.

Parameters

frame (*canlib.Frame*)

writeSync(*timeout*)

Wait for queued messages to be sent

Waits until all CAN messages for the specified handle are sent, or the timeout period expires.

Parameters
timeout (int) – The timeout in milliseconds, None or 0xFFFFFFFF for an infinite timeout.

writeWait(*frame*, *timeout*, **args*, ***kwargs*)

Sends a CAN message and waits for it to be sent.

This function sends a CAN message. It returns when the message is sent, or the timeout expires. This is a convenience function that combines write() and writeSync().

Deprecated since version 1.5: Sending the Frame contents as separate arguments; this functionality has been taken over by *writeWait_raw*.

Parameters

- **frame** (*canlib.Frame*) – Frame containing the CAN data to be sent
- **timeout** (int) – The timeout, in milliseconds. 0xFFFFFFFF gives an infinite timeout.

writeWait_raw(*id_*, *msg*, *flag=0*, *dlc=0*, *timeout=0*)

Sends a CAN message and waits for it to be sent.

This function sends a CAN message. It returns when the message is sent, or the timeout expires. This is a convenience function that combines write() and writeSync().

Parameters

- **id_** – The identifier of the CAN message to send.
- **msg** – An array or bytearray of the message data
- **flag** – A combination of *canlib.canlib.MessageFlag*. Use this parameter e.g. to send extended (29-bit) frames.
- **dlc** – The length of the message in bytes. For Classic CAN dlc can be at most 8, unless *canlib.canlib.Open.ACCEPT_LARGE_DLC* is used. For CAN FD dlc can be one of the following 0-8, 12, 16, 20, 24, 32, 48, 64. Optional, if omitted, dlc is calculated from the msg array.
- **timeout** – The timeout, in milliseconds. 0xFFFFFFFF gives an infinite timeout.

write_raw(*id_*, *msg*, *flag=0*, *dlc=None*)

Send a CAN message

See docstring of *Channel.write* for general information about sending CAN messages.

The variable name id (as used by canlib) is a built-in function in Python, so the name id_ is used instead.

Parameters

- **id_** – The identifier of the CAN message to send.
- **msg** – An array or bytearray of the message data
- **flag** – A combination of *MessageFlag*. Use this parameter e.g. to send extended (29-bit) frames.
- **dlc** – The length of the message in bytes. For Classic CAN dlc can be at most 8, unless *Open.ACCEPT_LARGE_DLC* is used. For CAN FD dlc can be one of the following 0-8, 12, 16, 20, 24, 32, 48, 64. Optional, if omitted, *dlc* is calculated from the *msg* array.

CanBusStatistics

class canlib.canlib.structures.**CanBusStatistics**

Result from reading bus statistics using *canlib.canlib.Channel.get_bus_statistics*.

Variables

- **busLoad** (int) – The bus load, expressed as an integer in the interval 0 - 10000 representing 0.00% - 100.00% bus load.
- **errFrame** (int) – Number of error frames.
- **extData** (int) – Number of received extended (29-bit identifiers) data frames.
- **extRemote** (int) – Number of received extended (29-bit identifiers) remote frames.
- **overruns** (int) – The number of overruns detected by the hardware, firmware or driver.
- **stdData** (int) – Number of received standard (11-bit identifiers) data frames.
- **stdRemote** (int) – Number of received standard (11-bit identifiers) remote frames.

ChannelData

class canlib.canlib.**ChannelData**(*channel_number*)

Object for querying various information about a channel

After instantiating a *ChannelData* object with a channel number, a variety of information is available as attributes. Most attributes are named after the C constant used to retrieve the information and are found in the list below.

Other information does not follow the C implementation completely, and are documented as separate properties further down.

There is also the *raw* function, that is used internally to get all information and can also be used to interact more directly with the dll.

Variables

- **bus_type** – A member of the *BusTypeGroup* enum. Not implemented in Linux.
- **card_firmware_rev** – A *canlib.VersionNumber* object with the version of the card's firmware.
- **card_hardware_rev** – A *canlib.VersionNumber* object with the version of the card's hardware.
- **card_number** – An int with the card's number in the computer. Each card type is numbered separately.
- **card_serial_no** – An int with the serial number of the card, or 0 if it doesn't have a serial number.
- **card_type** – A member of the *HardwareType* enum representing the hardware type of the card.
- **card_upc_no** – An *canlib.EAN* object with the EAN of the card, or None if it doesn't one.
- **chan_no_on_card** – An int of the channel number on the card.
- **channel_cap** – A *ChannelCap* object with the capabilities of the channel as flags. Also see ChannelData.channel_cap_mask.

- **channel_cap_ex** – A tuple of `ChannelCapEx` with the extended capabilities of the channel (added in v1.17).
- **channel_cap_mask** – A *ChannelCap* with which flags this device knows about.
- **channel_flags** – A *ChannelFlags* object with the status of the channel as flags.
- **channel_quality** – An `int` between 0 and 100 (inclusively) with the quality of the channel in percent of optimal quality. Not implemented in Linux.
- **clock_info** – A *ClockInfo* object with clock characteristics for the device (added in v1.16).
- **devdescr_ascii** – A `str` with the product name of the device.
- **devdescr_unicode** – A `str` with the product name of the device. Not implemented in Linux.
- **device_physical_position** – An `int` with the address of the device on its underlying bus. Not implemented in Linux.
- **devname_ascii** – A `str` with the current device name. Not implemented in Linux.
- **dll_file_version** – A *canlib.VersionNumber* with the version of the dll file.
- **dll_filetype** – 1 if "kvalapw.dll" is used, 2 if "kvalapw2.dll"
- **dll_product_version** – A *canlib.VersionNumber* with the product version of the dll.
- **driver_file_version** – A *canlib.VersionNumber* with the version of the kernel-mode driver. Not implemented in Linux.
- **driver_name** – A `str` with the name of the device driver.
- **driver_product_version** – A *canlib.VersionNumber* with the product version of the kernel-mode driver. Not implemented in Linux.
- **feature_ean** – An *canlib.EAN* object with an internal EAN. This is only intended for internal use.
- **hw_status** – Six `int` with hardware status codes. This is only intended for internal use.
- **is_remote** – A `bool` for whether the device is currently connected as a remote device. Not implemented in Linux.
- **logger_type** – A member of the *LoggerType* enum. Not implemented in Linux.
- **max_bitrate** – An `int` with the maximum bitrate of the device. Zero means no limit on the bitrate.
- **mfgname_ascii** – A `str` with the manufacturer's name.
- **mfgname_unicode** – A `str` with the manufacturer's name. Not implemented in Linux.
- **remote_host_name** – A `str` with the remote host name of the device. Not implemented in Linux.
- **remote_mac** – A `str` with the remote mac address of the device. Not implemented in Linux.
- **remote_operational_mode** – A member of the *OperationalMode* enum. Not implemented in Linux.
- **remote_profile_name** – A `str` with the remote profile name of the device. Not implemented in Linux.
- **remote_type** – A member of the *RemoteType* enum. Not implemented in Linux.

- **roundtrip_time** – An `int` with the roundtrip time measured in milliseconds. Not implemented in Linux.
- **time_since_last_seen** – An `int` with the time in milliseconds since last communication occured. Not implemented in Linux.
- **timesync_enabled** – A `bool` for whether legacy time synchronization is enabled. Not implemented in Linux.
- **trans_cap** – A *DriverCap* object with the capabilities of the transceiver as flags. Not implemented in Linux.
- **trans_serial_no** – An `int` with the serial number of the transceiver, or 0 if it doesn't have a serial number. Not implemented in Linux.
- **trans_type** – A member of the *TransceiverType* enum.
- **trans_upc_no** – An *canlib.EAN* object with the EAN of the transceiver, or `None` if it doesn't have one. Not implemented in Linux.
- **ui_number** – An `int` with the number associated with the device that can be displayed in the user interface. Not implemented in Linux.

property bus_param_limits

Get device's bus parameter limits

Example usage:

```
>>> chd = canlib.ChannelData(channel_number=2)
>>> limits = chd.bus_param_limits
>>> limits.arbitration_min._asdict()
{'tq': 0, 'phase1': 1, 'phase2': 1, 'sjw': 1, 'prescaler': 1, 'prop': 0}
>>> limits.arbitration_max._asdict()
{'tq': 0, 'phase1': 512, 'phase2': 32, 'sjw': 16, 'prescaler': 8192, 'prop': 0}
>>> limits.data_min._asdict()
{'tq': 0, 'phase1': 1, 'phase2': 1, 'sjw': 1, 'prescaler': 1, 'prop': 0}
>>> limits.data_max._asdict()
{'tq': 0, 'phase1': 512, 'phase2': 32, 'sjw': 16, 'prescaler': 8192, 'prop': 0}
```

The `tq` field is always zero, and is reserved for possible other uses in future releases.

If `prop` is zero for both `min` and `max` values, that means that the device does not distinguish between phase segment one and the propagation segment, i.e. the `phase1` limit applies to (`phase1` + `prop`).

Returns
busparams.BusParamTqLimits

New in version 1.20.

property channel_name

The product channel name.

Retrieves a clear text name of the channel. The name is returned as a string.

Type
`str`

property custom_name

The custom channel name if set, or an empty string otherwise

Type
`str`

property device_name

Deprecated since version 1.7.

raw(*item*, *ctype*=<*class 'ctypes.c_uint'*>)

A raw call to `canGetChannelData`

Parameters

- **item** (*ChannelDataItem*) – The information to be retrieved.
- **ctype** – The `ctypes` type that the information should be interpreted as.

class canlib.canlib.**HandleData**(*channel*)

Object for querying various information about a handle

This is identical to the *ChannelData* object but it's constructor takes a `canlib.Channel` instead of a channel number.

New in version 1.13.

Variables

- **bus_type** – A member of the *BusTypeGroup* enum. Not implemented in Linux.
- **card_firmware_rev** – A *canlib.VersionNumber* object with the version of the card's firmware.
- **card_hardware_rev** – A *canlib.VersionNumber* object with the version of the card's hardware.
- **card_number** – An `int` with the card's number in the computer. Each card type is numbered separately.
- **card_serial_no** – An `int` with the serial number of the card, or 0 if it doesn't have a serial number.
- **card_type** – A member of the *HardwareType* enum representing the hardware type of the card.
- **card_upc_no** – An *canlib.EAN* object with the EAN of the card, or `None` if it doesn't one.
- **chan_no_on_card** – An `int` of the channel number on the card.
- **channel_cap** – A *ChannelCap* object with the capabilities of the channel as flags. Also see `ChannelData.channel_cap_mask`.
- **channel_cap_ex** – A tuple of `ChannelCapEx` with the extended capabilities of the channel (added in v1.17).
- **channel_cap_mask** – A *ChannelCap* with which flags this device knows about.
- **channel_flags** – A *ChannelFlags* object with the status of the channel as flags.
- **channel_quality** – An `int` between 0 and 100 (inclusively) with the quality of the channel in percent of optimal quality. Not implemented in Linux.
- **clock_info** – A *ClockInfo* object with clock characteristics for the device (added in v1.16).
- **devdescr_ascii** – A `str` with the product name of the device.
- **devdescr_unicode** – A `str` with the product name of the device. Not implemented in Linux.
- **device_physical_position** – An `int` with the address of the device on its underlying bus. Not implemented in Linux.

- **devname_ascii** – A `str` with the current device name. Not implemented in Linux.
- **dll_file_version** – A *canlib.VersionNumber* with the version of the dll file.
- **dll_filetype** – 1 if "kvalapw.dll" is used, 2 if "kvalapw2.dll"
- **dll_product_version** – A *canlib.VersionNumber* with the product version of the dll.
- **driver_file_version** – A *canlib.VersionNumber* with the version of the kernel-mode driver. Not implemented in Linux.
- **driver_name** – A `str` with the name of the device driver.
- **driver_product_version** – A *canlib.VersionNumber* with the product version of the kernel-mode driver. Not implemented in Linux.
- **feature_ean** – An *canlib.EAN* object with an internal EAN. This is only intended for internal use.
- **hw_status** – Six `int` with hardware status codes. This is only intended for internal use.
- **is_remote** – A `bool` for whether the device is currently connected as a remote device. Not implemented in Linux.
- **logger_type** – A member of the *LoggerType* enum. Not implemented in Linux.
- **max_bitrate** – An `int` with the maximum bitrate of the device. Zero means no limit on the bitrate.
- **mfgname_ascii** – A `str` with the manufacturer's name.
- **mfgname_unicode** – A `str` with the manufacturer's name. Not implemented in Linux.
- **remote_host_name** – A `str` with the remote host name of the device. Not implemented in Linux.
- **remote_mac** – A `str` with the remote mac address of the device. Not implemented in Linux.
- **remote_operational_mode** – A member of the *OperationalMode* enum. Not implemented in Linux.
- **remote_profile_name** – A `str` with the remote profile name of the device. Not implemented in Linux.
- **remote_type** – A member of the *RemoteType* enum. Not implemented in Linux.
- **roundtrip_time** – An `int` with the roundtrip time measured in milliseconds. Not implemented in Linux.
- **time_since_last_seen** – An `int` with the time in milliseconds since last communication occured. Not implemented in Linux.
- **timesync_enabled** – A `bool` for whether legacy time synchronization is enabled. Not implemented in Linux.
- **trans_cap** – A *DriverCap* object with the capabilities of the transceiver as flags. Not implemented in Linux.
- **trans_serial_no** – An `int` with the serial number of the transceiver, or 0 if it doesn't have a serial number. Not implemented in Linux.
- **trans_type** – A member of the *TransceiverType* enum.
- **trans_upc_no** – An *canlib.EAN* object with the EAN of the transceiver, or `None` if it doesn't have one. Not implemented in Linux.

- **ui_number** – An `int` with the number associated with the device that can be displayed in the user interface. Not implemented in Linux.

raw(*item*, *ctype*=<*class 'ctypes.c_uint'*>)

A raw call to `canGetHandleData`

Parameters

- **item** (*ChannelDataItem*) – The information to be retrieved.
- **ctype** – The `ctypes` type that the information should be interpreted as.

Environment Variables

EnvVar

class canlib.canlib.**EnvVar**(*channel*)

Used to access environment variables in t programs.

The environment variables are accessed as an attribute with the same name as declared in the t program. If we have a running t program, which has defined the following environment variables:

```
envvar
{
  int   IntVal;
  float FloatVal;
  char  DataVal[512];
}
```

We access the first two using *EnvVar*:

```
>>> ch.envvar.IntVal
0
>>> ch.envvar.IntVal = 3
>>> ch.envvar.IntVal
3
>>> ch.envvar.FloatVal
15.0
```

The third environment variable, declared as `char*`, is accessed using *DataEnvVar*.

DataEnvVar

class canlib.canlib.envvar.**DataEnvVar**(*channel*, *handle*, *name*, *size*)

Represent an environment variable declared as `char*` in t programs.

This attribute object behaves like an array of bytes:

```
>>> ch.envvar.DataVal[100:141]
b'ot working? Messages can be sent to and r'
```

The size of the array must match what was defined in the t program. One way to do this is to left align the data and fill with zeros:

```
>>> data = 'My new data'.encode('utf-8')
>>> size = len(ch.envvar.DataVal)
>>> ch.envvar.DataVal = data.ljust(size, b'\0')
>>> ch.envvar.DataVal[:15]
b'My new data\x00\x00\x00\x00\x00'
```

Another way is to use slicing:

```
>>> ch.envvar.DataVal[3:6] = b'old'
>>> ch.envvar.DataVal[:15]
b'My old data\x00\x00\x00\x00\x00'
```

Enumerations

AcceptFilterFlag

class canlib.canlib.**AcceptFilterFlag**(*value*)

canFILTER_SET_xxx

NULL_MASK = 0

SET_CODE_EXT = 5

SET_CODE_STD = 3

SET_MASK_EXT = 6

SET_MASK_STD = 4

Bitrate

class canlib.canlib.**Bitrate**(*value*)

canBITRATE_xxx

Predefined CAN bitrates. See *BitrateFD* for predefined CAN FD bitrates.

New in version 1.17.

BITRATE_100K = -5

Indicate a bitrate of 100 kbit/s.

BITRATE_10K = -9

Indicate a bitrate of 10 kbit/s.

BITRATE_125K = -4

Indicate a bitrate of 125 kbit/s.

BITRATE_1M = -1

Indicate a bitrate of 1 Mbit/s.

BITRATE_250K = -3

Indicate a bitrate of 250 kbit/s.

BITRATE_500K = -2

Indicate a bitrate of 500 kbit/s.

BITRATE_50K = -7

Indicate a bitrate of 50 kbit/s.

BITRATE_62K = -6

Indicate a bitrate of 62 kbit/s.

BITRATE_83K = -8

Indicate a bitrate of 83 kbit/s.

BitrateFD

class canlib.canlib.**BitrateFD**(*value*)

canFD_BITRATE_xxx

Predefined CAN FD bitrates. Used when setting bitrates using the CAN FD protocol, see *Bitrate* for predefined CAN bitrates.

New in version 1.17.

BITRATE_1M_80P = -1001

Indicates a bitrate of 1 Mbit/s and sampling point at 80%.

BITRATE_2M_60P = -1007

Indicates a bitrate of 2 Mbit/s and sampling point at 60%.

BITRATE_2M_80P = -1002

Indicates a bitrate of 2 Mbit/s and sampling point at 80%.

BITRATE_4M_80P = -1003

Indicates a bitrate of 4 Mbit/s and sampling point at 80%.

BITRATE_500K_80P = -1000

Indicates a bitrate of 500 kbit/s and sampling point at 80%.

BITRATE_8M_60P = -1004

Indicates a bitrate of 8 Mbit/s and sampling point at 60%.

BITRATE_8M_70P = -1006

Indicates a bitrate of 8 Mbit/s and sampling point at 70%.

BITRATE_8M_80P = -1005

Indicates a bitrate of 8 Mbit/s and sampling point at 80%.

BusTypeGroup

class canlib.canlib.**BusTypeGroup**(*value*)

kvBUSTYPE_GROUP_xxx

Bus type group. This is a grouping of the individual kvBUSTYPE_xxx.

INTERNAL = 4

kvBUSTYPE_PCI, kvBUSTYPE_PCMCIA, ...

LOCAL = 2
kvBUSTYPE_USB

REMOTE = 3
kvBUSTYPE_WLAN, kvBUSTYPE_LAN

VIRTUAL = 1

ChannelCap

class canlib.canlib.**ChannelCap**(*value*)
canCHANNEL_CAP_xxx

Channel capabilities.

Changed in version 1.8.

BUS_STATISTICS = 2
Can report busload etc.

CAN_FD = 524288
CAN-FD ISO compliant channel.

CAN_FD_NONISO = 1048576
CAN-FD NON-ISO compliant channel.

DIAGNOSTICS = 268435456
Channel has diagnostic capabilities.

ERROR_COUNTERS = 4
Can return error counters.

EXTENDED_CAN = 1
Can use extended identifiers.

GENERATE_ERROR = 16
Can send error frames.

GENERATE_OVERLOAD = 32
Can send CAN overload frame.

IO_API = 134217728
Channel has diagnostic capabilities.

LIN_HYBRID = 67108864
Channel has LIN capabilities.

LOGGER = 8388608
Channel has logger capabilities.

REMOTE_ACCESS = 16777216
Channel has remote capabilities.

RESERVED_1 = 262144

RESERVED_2 = 8
Obsolete, only used by LAPcan driver

SCRIPT = 33554432

Channel has script capabilities.

SILENT_MODE = 2097152

Channel supports Silent mode.

SIMULATED = 131072

Simulated CAN channel.

SINGLE_SHOT = 4194304

Channel supports Single Shot messages.

TXACKNOWLEDGE = 128

Can report when a CAN messages has been transmitted.

TXREQUEST = 64

Can report when a CAN messsage transmission is initiated.

VIRTUAL = 65536

Virtual CAN channel.

ChannelDataItem

class canlib.canlib.**ChannelDataItem**(*value*)

Low level helper object representing `canCHANNELDATA_xxx`.

See the properties of *ChannelData* for how to get retrieve this data.

BUS_PARAM_LIMITS = 45

BUS_TYPE = 30

see `ChannelData.bus_type`

CARD_FIRMWARE_REV = 9

firmware revision number on the card, see `ChannelData.card_firmware_rev`

CARD_HARDWARE_REV = 10

version of the card's hardware, see `ChannelData.card_hardware_rev`

CARD_NUMBER = 5

the card's number in the computer, see `ChannelData.card_number`

CARD_SERIAL_NO = 7

serial number of the card, or 0, see `ChannelData.card_serial_no`

CARD_TYPE = 4

hardware type of the card, see `ChannelData.card_type`

CARD_UPC_NO = 11

EAN of the card, see `ChannelData.card_upc_no`

CHANNEL_CAP = 1

capabilities of the CAN controller, see `ChannelData.channel_cap`

CHANNEL_CAP_EX = 47

see `ChannelData.channel_cap_ex`

CHANNEL_CAP_MASK = 38

see ChannelData.channel_cap_mask

CHANNEL_FLAGS = 3

status of the channel as flags, see ChannelData.channel_flags

CHANNEL_NAME = 13

Deprecated

CHANNEL_QUALITY = 28

see ChannelData.channel_quality

CHAN_NO_ON_CARD = 6

local channel number on the card, see ChannelData.chan_no_on_card

CLOCK_INFO = 46

see ChannelData.clock_info

CUST_CHANNEL_NAME = 39

see ChannelData.cust_channel_name

DEVDESCR_ASCII = 26

product name of the device, see ChannelData.devdescr_ascii

DEVDESCR_UNICODE = 25

product name of the device, see ChannelData.devdescr_unicode

DEVICE_PHYSICAL_POSITION = 18

see ChannelData.device_physical_position

DEVNAME_ASCII = 31

see ChannelData.devname_ascii

DLL_FILETYPE = 16

see ChannelData.dll_filetype

DLL_FILE_VERSION = 14

version of the dll file, see ChannelData.dll_file_version

DLL_PRODUCT_VERSION = 15

version of the CANlib, see ChannelData.dll_product_version

DRIVER_FILE_VERSION = 21

version of the driver, see ChannelData.driver_file_version

DRIVER_NAME = 27

device driver name, see ChannelData.driver_name

DRIVER_PRODUCT_VERSION = 22

version of the CANlib, see ChannelData.driver_product_version

FEATURE_EAN = 44

see ChannelData.feature_ean

HW_STATUS = 43

see ChannelData.hw_status

IS_REMOTE = 40

see ChannelData.is_remote

LOGGER_TYPE = 42

see ChannelData.logger_type

MAX_BITRATE = 37

see ChannelData.max_bitrate

MFGNAME_ASCII = 24

manufacturer's name, see ChannelData.mfgname_ascii

MFGNAME_UNICODE = 23

manufacturer's name, see ChannelData.mfgname_unicode

REMOTE_HOST_NAME = 35

see ChannelData.remote_host_name

REMOTE_MAC = 36

see ChannelData.remote_mac

REMOTE_OPERATIONAL_MODE = 33

see ChannelData.remote_operational_mode

REMOTE_PROFILE_NAME = 34

see ChannelData.remote_profile_name

REMOTE_TYPE = 41

see ChannelData.remote_type

ROUNDTRIP_TIME = 29

see ChannelData.roundtrip_time

TIMESYNC_ENABLED = 20

see ChannelData.timesync_enabled

TIME_SINCE_LAST_SEEN = 32

see ChannelData.time_since_last_seen

TRANS_CAP = 2

capabilities of the CAN transceiver, see ChannelData.trans_cap

TRANS_SERIAL_NO = 8

serial number of the transceiver, or 0, see ChannelData.trans_serial_no

TRANS_TYPE = 17

transceiver type, see ChannelData.trans_type

TRANS_UPC_NO = 12

EAN of the transceiver, see ChannelData.trans_upc_no

UI_NUMBER = 19

see ChannelData.ui_number

ChannelFlags

class canlib.canlib.**ChannelFlags**(*value*)

canCHANNEL_IS_xxx

These channel flags are used in conjunction with `ChannelDataItem.channel_flags`.

IS_CANFD = 4

Channel has been opened as CAN FD.

IS_EXCLUSIVE = 1

Channel is opened exclusively.

IS_LIN = 16

Channel has been opened as LIN.

IS_LIN_MASTER = 32

Channel has been opened as a LIN master.

IS_LIN_SLAVE = 64

Channel has been opened as a LIN slave.

IS_OPEN = 2

Channel is active, either opened in LIN mode or on-bus in CAN mode.

DeviceMode

class canlib.canlib.**DeviceMode**(*value*)

kvDEVICE_MODE_xxx

INTERFACE = 0

Device is running or should be running in interface mode.

LOGGER = 1

Device is running or should be running in logger mode.

Driver

class canlib.canlib.**Driver**(*value*)

An enumeration.

NORMAL = 4

The “normal” driver type (push-pull). This is the default.

OFF = 0

The driver is turned off. Not implemented in all types of hardware.

SELFRECEPTION = 8

Self-reception. Not implemented.

SILENT = 1

Sets the CAN controller in Silent Mode.

DriverCap

class canlib.canlib.**DriverCap**(*value*)

canDRIVER_CAP_xxx

Driver (transceiver) capabilities.

HIGHSPEED = 1

EnvVarType

class canlib.canlib.**EnvVarType**(*value*)

kvENVVAR_TYPE_xxx

FLOAT = 2

The type of the envvar is C `float`.

INT = 1

The type of the envvar is C `int`.

STRING = 3

The type of the envvar is C `string`.

Error

class canlib.canlib.**Error**(*value*)

canERR_xxx

BUFFER_TOO_SMALL = -43

Buffer provided was not large enough.

CONFIG = -37

Configuration Error.

CRC = -36

CRC error.

DEVICE_FILE = -33

Device File error.

DISK = -35

Disk error.

DRIVER = -12

Driver type not supported

DRIVERFAILED = -24

DeviceIOControl failed, use Win32 GetLastError() to learn more

DRIVERLOAD = -23

Can't find or load kernel driver

DYNAINIT = -18

Error when initializing a DLL

DYNALIB = -17

A DLL seems to have wrong version

DYNALOAD = -16

A driver DLL can't be found or loaded

HARDWARE = -15

A hardware error has occurred

HOST_FILE = -34

Host File error.

INIFILE = -11

Error in the ini-file (16-bit only)

INTERNAL = -30

Internal error in the driver.

INTERRUPTED = -6

Interrupted by signals.

INVALID_PASSWORD = -128

INVALID_SESSION = -131

INVHANDLE = -10

Handle is invalid

IO_CONFIG_CHANGED = -46

The I/O pin configuration has changed after last confirmation.

IO_NOT_CONFIRMED = -45

The I/O pin configuration is not confirmed.

IO_NO_VALID_CONFIG = -48

There is no valid I/O pin configuration.

IO_PENDING = -47

The previous I/O pin value has not yet changed the output.

IO_WRONG_PIN_TYPE = -44

I/O pin doesn't exist or the I/O pin type doesn't match.

LICENSE = -29

The license is not valid.

MEMO_FAIL = -38

Memo Error.

NOCARD = -26

The card was removed or not inserted

NOCHANNELS = -5

No channels available.

NOCONFIGMGR = -25

Can't find req'd config s/w (e.g. CS/SS)

NOHANDLES = -9

Out of handles

NOMEM = -4

Out of memory.

NOMSG = -2

There were no messages to read.

NOTFOUND = -3

Specified device or channel not found.

NOTINITIALIZED = -8

The library is not initialized.

NOT_AUTHORIZED = -130

NOT_IMPLEMENTED = -32

Not implemented.

NOT_SUPPORTED = -19

Operation not supported by hardware or firmware

NO_ACCESS = -31

Access denied.

NO_SUCH_FUNCTION = -129

PARAM = -1

Error in one or more parameters.

REGISTRY = -28

Error (missing data) in the Registry

RESERVED_2 = -22

Reserved

RESERVED_5 = -20

Reserved

RESERVED_6 = -21

Reserved

RESERVED_7 = -27

Reserved

SCRIPT_FAIL = -39

Script Fail.

SCRIPT_WRONG_VERSION = -40

Unsupported t program version.

TIMEOUT = -7

Timeout occurred.

TXBUFOFL = -13

Transmit buffer overflow

TXE_CONTAINER_FORMAT = -42

Parsing t program failed.

TXE_CONTAINER_VERSION = -41

Unsuppoted txe version.

HardwareType

class canlib.canlib.**HardwareType**(*value*)

canHWTYPE_xxx

The following constants can be returned from *ChannelData*, using the `card_type` property. They identify the hardware type for the current channel.

Note: The members indicate a hardware type, but not necessarily a specific product. For example, `canHWTYPE_LAPCAN` is returned both for LAPcan and LAPcan II. (Use the `card_upc_no` property of *ChannelData* to obtain the UPC/EAN code for the device. This number uniquely identifies the product.)

ACQUISITOR = 46

Kvaser Acquisitor (obsolete).

BAGEL = 64

Obsolete name, use *BLACKBIRD_V2* instead.

BLACKBIRD = 58

Kvaser BlackBird.

BLACKBIRD_V2 = 64

Kvaser BlackBird v2.

CANLINHYBRID = 84

Kvaser Hybrid CAN/LIN.

CANPARI = 3

CANpari (obsolete).

DINRAIL = 86

Kvaser DIN Rail SE400S and variants

EAGLE = 62

Kvaser Eagle family.

ETHERCAN = 70

Kvaser Ethercan.

IRIS = 58

Obsolete name, use *BLACKBIRD* instead.

LAPCAN = 2

LAPcan Family.

LEAF = 48

Kvaser Leaf Family.

LEAF2 = 80

Kvaser Leaf Pro HS v2 and variants.

MEMORATOR_II = 54

Kvaser Memorator Professional family.

MEMORATOR_LIGHT = 60

Kvaser Memorator Light.

MEMORATOR_PRO = 54

Kvaser Memorator Professional family.

MEMORATOR_PRO2 = 78

Kvaser Memorator Pro 5xHS and variants.

MEMORATOR_V2 = 82

Kvaser Memorator (2nd generation)

MINIHYDRA = 62

Obsolete name, use *EAGLE* instead.

MINIPCIE = 66

Kvaser Mini PCI Express.

NONE = 0

Unknown or undefined.

PC104_PLUS = 50

Kvaser PC104+.

PCCAN = 8

PCcan Family.

PCICAN = 9

PCIcan Family.

PCICANX_II = 52

Kvaser PCIcanx II.

PCICAN_II = 40

PCIcan II family.

PCIE_V2 = 76

Kvaser PCIEcan 4xHS and variants.

SIMULATED = 44

Simulated CAN bus for Kvaser Creator (obsolete).

U100 = 88

Kvaser U100 and variants

USBCAN = 11

USBcan (obsolete).

USBCAN_II = 42

USBcan II, USBcan Rugged, Kvaser Memorator.

USBCAN_KLINE = 68

USBcan Pro HS/K-Line.

USBCAN_LIGHT = 72

Kvaser USBcan Light.

USBCAN_PRO = 56

Kvaser USBcan Professional.

USBCAN_PRO2 = 74

Kvaser USBcan Pro 5xHS and variants.

VIRTUAL = 1

The virtual CAN bus.

IOControlItem

class canlib.canlib.**IOControlItem**(*value*)

An enumeration used in *Channel.iocontrol*.

CLEAR_ERROR_COUNTERS = 5

Clear the CAN error counters.

CONNECT_TO_VIRTUAL_BUS = 22

Windows only

DISCONNECT_FROM_VIRTUAL_BUS = 23

Windows only

FLUSH_RX_BUFFER = 10

Discard contents of the RX queue.

FLUSH_TX_BUFFER = 11

Discard contents of the TX queue.

GET_BUS_TYPE = 36

Windows only.

GET_CHANNEL_QUALITY = 34

Read remote channel quality.

GET_DEVNAME_ASCII = 37

Retreive device name.

GET_DRIVERHANDLE = 17

Windows only.

GET_EVENTHANDLE = 14

Windows only.

GET_REPORT_ACCESS_ERRORS = 21

Current setting of access error reporting

GET_ROUNDTRIP_TIME = 35

Round trip time in ms, for remote channel.

GET_RX_BUFFER_LEVEL = 8

Current receive queue, RX, level.

GET_THROTTLE_SCALED = 42

Windows only

GET_TIMER_SCALE = 12

Current time-stamp clock resolution in microseconds.

GET_TIME_SINCE_LAST_SEEN = 38

For WLAN devices, this is the time since the last keep-alive message.

GET_TREF_LIST = 39

Unsupported

GET_TXACK = 31

Status of Transmit Acknowledge.

GET_TX_BUFFER_LEVEL = 9

Current transmitt queue, TX, level.

GET_USB_THROTTLE = 29

For internal use only.

GET_USER_IOPORT = 25

Read IO port value.

GET_WAKEUP = 19

For internal use only.

LIN_MODE = 45

For internal use only.

MAP_RXQUEUE = 18

For internal use only.

PREFER_EXT = 1

Tells CANlib to "prefer" extended identifiers.

PREFER_STD = 2

Tells CANlib to "prefer" standard identifiers.

RESET_OVERRUN_COUNT = 44

Reset overrun count and flags.

SET_BRLIMIT = 43

Max bitrate limit can be overridden with this IOCTL.

SET_BUFFER_WRAPAROUND_MODE = 26

For internal use only.

SET_BUSON_TIME_AUTO_RESET = 30

Enable/disable time reset at bus on.

SET_BYPASS_MODE = 15

Not implemented.

SET_ERROR_FRAMES_REPORTING = 33

Windows only

SET_LOCAL_TXACK = 46

Enable reception of canMSG_LOCAL_TXACK.

SET_LOCAL_TXECHO = 32

Turn on/off local transmit echo.

SET_REPORT_ACCESS_ERRORS = 20

Turn access error reporting on/off.

SET_RX_QUEUE_SIZE = 27

Windows only.

SET_THROTTLE_SCALED = 41

Windows only

SET_TIMER_SCALE = 6

Set time-stamp clock resolution in microseconds, default 1000.

SET_TXACK = 7

Enable/disable Transmit Acknowledge.

SET_TXRQ = 13

Turn transmit requests on or off.

SET_USB_THROTTLE = 28

For internal use only.

SET_USER_IOPORT = 24

Set IO port to value.

SET_WAKEUP = 16

For internal use only.

TX_INTERVAL = 40

Minimum CAN message transmit interval

LEDAction

class canlib.canlib.**LEDAction**(*value*)

kvLED_ACTION_xxx

The following can be used together with *canlib.canlib.Channel.flashLeds*.

Changed in version 1.18: Added LEDs 4 through 11 (needs CANlib v5.19+)

ALL_LEDS_OFF = 1

Turn all LEDs off.

ALL_LEDS_ON = 0

Turn all LEDs on.

LED_0_OFF = 3

Turn LED 0 off.

LED_0_ON = 2

Turn LED 0 on.

LED_10_OFF = 23
Turn LED 10 off.

LED_10_ON = 22
Turn LED 10 on.

LED_11_OFF = 25
Turn LED 11 off.

LED_11_ON = 24
Turn LED 11 on.

LED_1_OFF = 5
Turn LED 1 off.

LED_1_ON = 4
Turn LED 1 on.

LED_2_OFF = 7
Turn LED 2 off.

LED_2_ON = 6
Turn LED 2 on.

LED_3_OFF = 9
Turn LED 3 off.

LED_3_ON = 8
Turn LED 3 on.

LED_4_OFF = 11
Turn LED 4 off.

LED_4_ON = 10
Turn LED 4 on.

LED_5_OFF = 13
Turn LED 5 off.

LED_5_ON = 12
Turn LED 5 on.

LED_6_OFF = 15
Turn LED 6 off.

LED_6_ON = 14
Turn LED 6 on.

LED_7_OFF = 17
Turn LED 7 off.

LED_7_ON = 16
Turn LED 7 on.

LED_8_OFF = 19
Turn LED 8 off.

LED_8_ON = 18
Turn LED 8 on.

LED_9_OFF = 21

Turn LED 9 off.

LED_9_ON = 20

Turn LED 9 on.

LoggerType

class canlib.canlib.**LoggerType**(*value*)

kvLOGGER_TYPE_xxx

Logger type, returned when using `ChannelData.logger_type`.

NOT_A_LOGGER = 0

V1 = 1

V2 = 2

MessageFlag

class canlib.canlib.**MessageFlag**(*value*)

Message information flags

The following flags can be returned from *Channel.read* et al, or passed to *Channel.write*.

This enum is a combination of flags for messages, CAN FD messages, and message errors. Normal messages flags are the flags covered by *MSG_MASK*, CAN FD message flags are those covered by *FDMSG_MASK*, and message errors are those covered by *MSGERR_MASK*.

Note: *FDF*, *BRS*, and *ESI* require CAN FD.

RTR cannot be set for CAN FD messages.

Not all hardware platforms can detect the difference between hardware overruns and software overruns, so your application should test for both conditions.

ABL = 67108864

Single shot message was not sent because arbitration was lost.

BIT = 49152

All bit error.

BIT0 = 16384

Sent dominant bit, read recessive bit

BIT1 = 32768

Sent recessive bit, read dominmant bit

BRS = 131072

Message is sent/received with bit rate switch (CAN FD)

BUSERR = 63488

All RX error.

CRC = 8192
CRC error.

EDL = 65536
obsolete

ERROR_FRAME = 32
Message represents an error frame.

ESI = 262144
Sender of the message is in error passive mode (CAN FD)

EXT = 4
Message has an extended (29-bit) identifier.

FDF = 65536
Message is a CAN FD message.

FDMSG_MASK = 16711680
obsolete

FORM = 4096
Form error.

HW_OVERRUN = 512
Hardware buffer overrun.

LOCAL_TXACK = 268435456
Message is a LOCAL TX ACK (was transmitted from another handle on the same channel)

MSGERR_MASK = 65280
Used to mask the non-error bits

MSG_MASK = 255
Used to mask the non-info bits.

NERR = 16
NERR was active during the message (TJA1054 hardware)

OVERRUN = 1536
Both SW and HW overrun conditions

RTR = 1
Message is a remote request.

SINGLE_SHOT = 16777216
Message is Single Shot, try to send once, no retransmission.

STD = 2
Message has a standard (11-bit) identifier.

STUFF = 2048
Stuff error.

SW_OVERRUN = 1024
Software buffer overrun.

TXACK = 64
Message is a TX ACK (msg has really been sent)

TXNACK = 33554432
Message is a failed Single Shot, message was not sent.

TXRQ = 128
Message is a TX REQUEST (msg was transfered to the chip)

WAKEUP = 8
Message is a WAKEUP message, Single Wire CAN.

Notify

class canlib.canlib.**Notify**(*value*)
canNOTIFY_xxx

These notify flags are used in *Channel.set_callback* to indicate different kind of events.

BUSONOFF = 32
Notify on bus on/off status changed

ENVVAR = 16
An environment variable was changed by a script. Note that you will not be notified when an environment variable is updated from the Canlib API.

ERROR = 4
CAN bus error notification

NONE = 0
Turn notifications off.

REMOVED = 64
Notify on device removed

RX = 1
CAN message reception notification

STATUS = 8
CAN chip status change

TX = 2
CAN message transmission notification

Open

class canlib.canlib.**Open**(*value*)
Flags used in the flags argument to *canlib.openChannel()*.

ACCEPT_LARGE_DLC = 512
DLC can be greater than 8.

The channel will accept CAN messages with DLC (Data Length Code) greater than 8. If this flag is not used, a message with DLC > 8 will always be reported or transmitted as a message with DLC = 8. When the *ACCEPT_LARGE_DLC* flag is used, the message will be sent and/or received with the true DLC, which can be at most 15. The length of the message is always at most 8.

ACCEPT_VIRTUAL = 32

Allow opening of virtual channels as well as physical channels.

CAN_FD = 1024

The channel will use the CAN FD protocol, ISO compliant.

This also means that messages with *MessageFlag.FDF*, *MessageFlag.BRS* and *MessageFlag.ESI* can now be used.

CAN_FD_NONISO = 2048

The channel will use the CAN FD NON-ISO protocol.

Use this if you want to configure the can controller to be able to communicate with a can controller designed prior to the release of the CAN FD ISO specification.

Non ISO mode implies:

1. The stuff bit counter will not be included in the frame format.
2. Initial value for CRC17 and CRC21 will be zero.

This also means that messages with *MessageFlag.FDF*, *MessageFlag.BRS* and *MessageFlag.ESI* can now be used.

EXCLUSIVE = 8

Don't allow sharing between multiple *Channel* objects.

Two or more threads or applications can share the same CAN channel by opening the same CANlib channel multiple times. If this is not desired you can open a CANlib channel exclusively once by passing the *EXCLUSIVE* flag. If the CANlib channel is aleady open, the call to *canlib.openChannel()* will fail.

NOFLAG = 0

NO_INIT_ACCESS = 256

Don't open the handle with init access.

Note: A handle opened without init access will still set default bitrate when going on bus, if no other handle has opened the channel with init access at the time of the buson.

OVERRIDE_EXCLUSIVE = 64

Open the channel even if it is opened for exclusive access already.

REQUIRE_EXTENDED = 16

This flag causes two things to happen:

1. The call will fail if the specified circuit doesn't allow extended CAN (CAN 2.0B).
2. If no frame-type flag is specified in a call to *Channel.write()*, it is assumed that extended CAN should be used.

REQUIRE_INIT_ACCESS = 128

Fail the call if the channel cannot be opened with init access.

Init access means that the CAN handle can set bit rate and CAN driver mode. At most one CAN handle may have init access to any given channel. If you try to set the bit rate or CAN driver mode for a handle to which you don't have init access, the call will silently fail (i.e. `canOK` is returned although the call had no effect), unless you enable "access error reporting" by using `Channel.IOControlItem.SET_REPORT_ACCESS_ERRORS`. Access error reporting is by default off. Init access is the default.

OperationalMode

class canlib.canlib.**OperationalMode**(*value*)

canCHANNEL_OPMODE_xxx

Current WLAN operational mode.

ADH = 4

Adhoc mode

INFRASTRUCTURE = 2

Infrastructure mode

NONE = 1

Not applicable, or unknown mode.

RESERVED = 3

Reserved

RemoteType

class canlib.canlib.**RemoteType**(*value*)

kvREMOTE_TYPExxx

Remote type, returned when using canCHANNELDATA_REMOTE_TYPE

LAN = 2

NOT_REMOTE = 0

WLAN = 1

ScriptRequest

class canlib.canlib.**ScriptRequest**(*value*)

kvSCRIPT_REQUEST_TEXT_xxx

These defines are used in kvScriptRequestText() for printf message subscribe/unsubscribe.

ALL_SLOTS = 255

SUBSCRIBE = 2

UNSUBSCRIBE = 1

ScriptStatus

class canlib.canlib.**ScriptStatus**(*value*)

Status of t program

New in version 1.6.

Changed in version 1.7: Is now based on `IntFlag` instead of `IntEnum`

IDLE = 0

LOADED = 1

RUNNING = 2

ScriptStop

class canlib.canlib.**ScriptStop**(*value*)

An enumeration.

FORCED = -9

NORMAL = 0

Stat

class canlib.canlib.**Stat**(*value*)

canSTAT_xxx

The following circuit status flags are returned by *Channel.readStatus*. Note that more than one flag might be set at any one time.

Note: Usually both canSTAT_HW_OVERRUN and canSTAT_SW_OVERRUN are set when overrun has occurred. This is because the kernel driver can't see the difference between a software overrun and a hardware overrun, but the code should always test for both types of overruns.

BUS_OFF = 2

The circuit is Off Bus

ERROR_ACTIVE = 8

The circuit is error active.

ERROR_PASSIVE = 1

The circuit is error passive

ERROR_WARNING = 4

At least one error counter > 96

HW_OVERRUN = 512

There has been at least one HW buffer overflow

OVERRUN = 1536

Both *HW_OVERRUN* and *SW_OVERRUN* is active

RESERVED_1 = 64

RXERR = 256

There has been at least one RX error of some sort

RX_PENDING = 32

There are messages in the receive buffer

SW_OVERRUN = 1024

There has been at least one SW buffer overflow

TXERR = 128

There has been at least one TX error

TX_PENDING = 16

There are messages pending transmission

TransceiverType

class canlib.canlib.**TransceiverType**(*value*)

Transceiver (logical) types

The following constants can be returned from canGetChannelData(), using the canCHANNELDATA_TRANS_TYPE item code. They identify the bus transceiver type for the channel specified in the call to canGetChannelData.

Note: If the type starts with a number `T_` has been prepended to the name.

They indicate a hardware type, but not necessarily a specific circuit or product.

CANFD = 22

CANFD_LIN = 24

HYBRID CAN-FD/LIN

DNOPTO = 3

Optoisolated 82C251

EVA = 7

FIBER = 8

82c251 with fibre extension

K = 10

K-line, without CAN.

K251 = 9

K-line + 82c251

KONE = 20

LIN = 19

LINX_J1708 = 66

LINX_K = 68

LINX_LIN = 64

LINX_LS = 72

LINX_SWC = 70

RS485(*i.e. J1708*) **= 18**
RS485 (i.e. J1708)

SWC = 6
AU5790

SWC_OPTO = 12
AU5790 with optical isolation

SWC_PROTO = 5
AU5790 prototype

TT = 13
B10011S Truck-And-Trailer

T_1041 = 16
TJA1041

T_1041_OPTO = 17
TJA1041 with optical isolation

T_1050 = 14
TJA1050

T_1050_OPTO = 15
TJA1050 with optical isolation

T_1054_OPTO = 11
TJA1054 with optical isolation

T_251 = 1
82c251

T_252 = 2
82c252, TJA1053, TJA1054

UNKNOWN = 0
Unknown or undefined

W210 = 4

TxeDataItem

class canlib.canlib.**TxeDataItem**(*value*)
An enumeration.

COMPILER_VERSION = 2
The three part version number of the compiler used to create the compiled script file (.txe).

DATE = 3
Compilation date in Coordinated Universal Time (UTC) of the compiled script file (.txe).

Contents: 0. Year, 1. Month, 2. Day, 3. Hour, 4. Minute, 5. Second.

DESCRIPTION = 4
Description of the compiled script file (.txe).

FILE_VERSION = 1

The three part version number of the compiled script file (.txe) file format.

IS_ENCRYPTED = 7

Non-zero value if the compiled script file (.txe) contents is encrypted.

SIZE_OF_CODE = 6

The size of the compiled code in the .txe file.

SOURCE = 5

The name followed by the content of each unencrypted source file

IOControl

class canlib.canlib.**IOControl**(*channel*)

Helper object for using `canIoCtl`

Provides a variety of functionality, some of which are represented as attributes of this object and some as functions. See the respective entries below for more information.

Variables

- **brlimit** – An `int` with the hardware bitrate limit, or zero for the device's default. Write-only.
- **bus_type** – A member of the *BusTypeGroup* enum. Not implemented in Linux. Read-only.
- **buson_time_auto_reset** – A `bool` for whether the CAN clock is reset at bus-on. Not implemented in Linux. Write-only.
- **channel_quality** – An `int` between 0 and 100 (inclusively) with the quality of the channel in percent. Not implemented in Linux. Read-only.
- **devname_ascii** – A `str` with the current device name. Not implemented in Linux. Read-only.
- **driverhandle** – The windows handle related to the CANlib handle. Not implemented in Linux. Read-only.
- **error_frames_reporting** – A `bool` for whether error frames are reported. Not implemented in Linux. Write-only.
- **eventhandle** – An `int` with the windows event handle. Not implemented in Linux. Read-only.
- **local_txack** – A `bool` for whether local transmit acknowledge is turned on. Write-only.
- **local_txecho** – A `bool` for whether local transmit echo is turned on. Write-only.
- **report_access_errors** – A `bool` for whether Access Reporting is turned on. Default is `False` (off).
- **roundtrip_time** – An `int` with the roundtrip time in milliseconds. Not implemented in Linux. Read-only.
- **rx_buffer_level** – An `int` with the approximate receive queue level. Read-only.
- **rx_queue_size** – An `int` with the size of the receive buffer. Can only be used off-bus. Not implemented in Linux. Write-only.

- **throttle_scaled** – An `int` between 0 and 100 (inclusively) where 0 means the device is very responsive but generates more CPU load and 100 means the device is less responsive with less CPU load. Note that not all devices support setting this. Some hardware will accept this command but neglect it. Not implemented in Linux.
- **time_since_last_seen** – An `int` with the time in milliseconds since the last communication occured. Not implemented in Linux. Read-only.
- **timer_scale** – An `int` with the time-stamp clock resolution in microseconds. Used e.g. in *Channel.read()*. Default is 1000, i.e. 1 ms.
- **tx_buffer_level** – An `int` with the approximate transmit queue level. Read-only.
- **tx_interval** – An `int` with the number of microseconds with the minimum CAN message transmit interval.
- **txack** – 0 for Transmit Acknowledges off, 1 for Transmit Acknowledges on, and 2 for Transmit Acknowledges off, even for the driver's internal usage (this will break parts of the library).
- **txrq** – A `bool` for whether Transmit Requests are turned on. Write-only.

clear_error_counters()

Tells CANlib to clear the CAN error counters. CAN error counters on device side are NOT updated. It is recommended to use *reset_overrun_count* to reset overrun status. Not implemented in Linux.

connect_to_virtual_bus(*value*)

Connects the channel to the virtual bus number `value`.

disconnect_from_virtual_bus(*value*)

Disconnects the channel from the virtual bus number `value`.

flush_rx_buffer()

Discard the current contents of the RX queue.

flush_tx_buffer()

Discard the current contents of the TX queue.

prefer_ext()

Tells CANlib to assume *MessageFlag.EXT* when sending messages if neither *MessageFlag.EXT* or *MessageFlag.STD* is specified. Not implemented in Linux.

prefer_std()

Tells CANlib to assume *MessageFlag.STD* when sending messages if neither *MessageFlag.EXT* or *MessageFlag.STD* is specified. Not implemented in Linux.

reset_overrun_count()

Resets overrun count and flags.

raw(*item*, *value=None*, *ctype=<class 'ctypes.c_uint'>*)

A raw call to `canIoCtl`

Parameters

- **item** (*IOControlItem*) – The "function code" to be passed to `canIoCtl`.
- **value** – The value sent to `canIoCtl` or `None` if no value should be given. Must be compatible with the `ctype` argument.
- **ctype** – The `ctypes` type that should be used to when sending the `value` argument and when interpreting the result of `canIoCtl`.

I/O pin

Experimental support for accessing IO-pins on sub modules of the Kvaser DIN Rail SE 400S and variants that was added to CANlib v5.26.

New in version 1.8.

AddonModule

class canlib.canlib.iopin.**AddonModule**(*module_type*, *fw_version=None*, *serial=None*, *first_pin_index=None*)

Contains information about one add-on module

Parameters

- **module_type** (*ModuleType*) – The type of the add-on module.
- **sw_version** (*canlib.VersionNumber*) – The software version in the add-on module.
- **serial** (*int*) – The serial number of the add-on module.
- **first_index** (*int*) – The index of the add-on modules first pin.

New in version 1.9.

issubset(*spec*)

Check if current attributes are fulfilling attributes in *spec*.

Any attribute in spec that is set to None is automatically considered fulfilled.

The `fw_version` attribute is considered fulfilled when `self.fw_version >= spec.fw_version`.

This can be used to check if a specific module fulfills a manually created specification:

```
>>> module_spec = [iopin.AddonModule(module_type=iopin.ModuleType.DIGITAL)]
... config = iopin.Configuration(channel)
>>> config.modules
[AddonModule(module_type=<ModuleType.DIGITAL: 1>, fw_
↪version=VersionNumber(major=2, minor=5, release=None, build=None),␣
↪serial=2342), first_pin_index=0]
>>> config.issubset(module_spec)
True

>>> module_spec = [iopin.AddonModule(
        module_type=iopin.ModuleType.DIGITAL,
        fw_version=VersionNumber(major=3, minor=1),
        serial=2342)]

>>> config.issubset(module_spec)
False

>>> module_spec = [
    iopin.AddonModule(module_type=iopin.ModuleType.ANALOG),
    iopin.AddonModule(module_type=iopin.ModuleType.DIGITAL,
                      fw_version=VersionNumber(major=3, minor=1),
                      serial=2342)]
```

(continues on next page)

(continued from previous page)

```
>>> config.issubset(module_spec)
False
```

AnalogIn

class canlib.canlib.iopin.**AnalogIn**(*channel*, *pin*)

Bases: *IoPin*

property hysteresis

The hysteresis in Volt for analog input pin

property lp_filter_order

The low-pass filter order for analog input pin

property value

Voltage level on the Analog input pin

AnalogOut

class canlib.canlib.iopin.**AnalogOut**(*channel*, *pin*)

Bases: *IoPin*

property value

Voltage level on the Analog output pin

Configuration

class canlib.canlib.iopin.**Configuration**(*channel*)

Contains I/O pins and the *canlib.Channel* to find them on

Creating this object may take some time depending on the number of I/O pins availably on the given *canlib.Channel*.

Parameters

channel (*Channel*) – The channel where the discovery of I/O pins should take place.

Variables

- **io_pins** (list(*IoPin*)) – All discovered I/O pins.
- **modules** (list(*AddonModule*)) – All included add-on-modules.
- **pin_names** (*list(str)*) – List of label I/O pin names.
- **(dict(str** (*pin_index*) – int)): Dictionary with I/O pin label name as key, and pin index as value.

To create an *iopin.Configuration* you need to supply the *canlib.Channel*, which is were we look for I/O pins:

```
>>> from canlib.canlib import iopin
... from canlib import canlib, Device, EAN
... device = Device.find(ean=EAN('01059-8'), serial=225)
... channel = canlib.openChannel(device.channel_number(), canlib.Open.EXCLUSIVE)
... config = iopin.Configuration(channel)
```

Now we can investigate a specific pin by index:

```
>>> config.pin(index=80)
Pin 80: <PinType.ANALOG: 2> <Direction.OUT: 8> bits=12 range=0.0-10.0 (<ModuleType.
↪ANALOG: 2>)
```

It is also possible to find the label name from the index and vice verse for a pin, as well as access the pin using the label name:

```
>>> config.name(80)
'4:AO1'

>>> config.index('4:AO1')
80

>>> config.pin(name='4:AO1')
Pin 80: <PinType.ANALOG: 2> <Direction.OUT: 8> bits=12 range=0.0-10.0 (<ModuleType.
↪ANALOG: 2>)
```

Note: A configuration needs to be confirmed using *iopin.Configuration.confirm* (which calls *Channel.io_confirm_config*) before accessing pin values:

```
>>> config.pin(name='4:AO1').value = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "...\canlib\canlib\iopin.py", line 271, in value
  File "...\canlib\canlib\dll.py", line 94, in _error_check
    raise can_error(result)
canlib.canlib.exceptions.IoPinConfigurationNotConfirmed: I/O pin configuration is␣
↪not confirmed (-45)
I/O pin configuration is not confirmed (-45)

>>> config.confirm()

>>> config.pin(name='4:AO1').value = 4
```

A *Configuration* may be compared with an expected ordered list of *AddonModule* before confirming using *AddonModule.issubset*

Changed in version 1.9: `Configuration.modules` is now an attribute, containing an ordered list of *AddonModule* objects.

confirm()

Confirm current configuration

Convenience function that calls *Channel.io_confirm_config*.

index(*name*)

Return index for pin with the given label name

issubset(*spec*)

Check if attributes of modules in self is fulfilled by given spec

This is a convenience method that calls *AddonModule.issubset* on all modules given by `self.modules` which can be used to check if the current configuration fulfills a manually created specification:

```
>>> config = iopin.Configuration(channel)
>>> config_spec = [iopin.AddonModule(module_type=iopin.ModuleType.ANALOG),
                   iopin.AddonModule(module_type=iopin.ModuleType.DIGITAL,
                                     fw_version=VersionNumber(major=3, minor=1),
                                     serial=2342)]

>>> config.issubset(config_spec)
False
```

New in version 1.9.

name(*index*)

Return label name for pin with given index

pin(*index=None*, *name=None*)

Return *IoPin* object using index or name

Either *index* or *name* must be given, if both are given, the name will be used.

Parameters

- **index** (*int*) – I/O pin index
- **name** (*str*) – I/O pin name

DigitalIn

class canlib.canlib.iopin.**DigitalIn**(*channel*, *pin*)

Bases: *IoPin*

property high_low_filter

Filter time in micro seconds when a digital pin goes from HIGH to LOW

property low_high_filter

Filter time in micro seconds when a digital pin goes from LOW to HIGH

property value

Value on digital input pin (0 or 1)

DigitalOut

class canlib.canlib.iopin.**DigitalOut**(*channel*, *pin*)

Bases: *IoPin*

property value

Value on digital output pin (0 or 1)

DigitalValue

class canlib.canlib.iopin.**DigitalValue**(*value*)

Enum used digital values

HIGH = 1

LOW = 0

Direction

class canlib.canlib.iopin.**Direction**(*value*)

Enum used for values in *Info*

IN = 4

Input

OUT = 8

Output

Info

class canlib.canlib.iopin.**Info**(*value*)

Enum used internally in *IoPin* for calls to `kvIoPinGetInfo` and `kvIoPinSetInfo`

AI_HYSTERESIS = 11

The hysteresis in volt.

The hysteresis in volt for an analog input pin, i.e. the amount the input have to change before the sampled value is updated.

0.0 - 10.0, default 0.3

AI_LP_FILTER_ORDER = 10

The low-pass filter order for an analog input pin.

0 - 16, default 3 (sample time is 1 ms)

DIRECTION = 2

One of *Direction*

DI_HIGH_LOW_FILTER = 9

Time when a digital input pin goes from LOW to HIGH.

Filter time in micro seconds when a digital input pin goes from LOW to HIGH.

Range: 0 - 65000, Default 5000 us

DI_LOW_HIGH_FILTER = 8

Time when a digital input pin goes from HIGH to LOW.

Filter time in micro seconds when a digital input pin goes from HIGH to LOW. Range: 0 - 65000, Default 5000 us

FW_VERSION = 16

Software version number of the submodule the pin belongs to. Read-only.

MODULE_NUMBER = 14

The module number the pin belongs to. The number starts from 0. Read-only.

MODULE_TYPE = 1

One of *ModuleType*

NUMBER_OF_BITS = 5

Resolution in number of bits. Read-only.

PIN_TYPE = 4

One of *PinType*

RANGE_MAX = 7

A float that contains the upper range limit in volts. Read-only.

RANGE_MIN = 6

A float that contains the lower range limit in volts. Read-only.

SERIAL_NUMBER = 15

Serial number of the submodule the pin belongs to. Read-only.

IoPin

class canlib.canlib.iopin.**IoPin**(*channel*, *pin*)

Base class of I/O ports

property direction

Pin direction (Read-only)

Type
Direction

property fw_version

Firmware version in module (Read-only)

Type
VersionNumber

property hysteresis

Base class does not implement hysteresis attribute

property lp_filter_order

Base class does not implement lp_filter_order attribute

property module_type

Type of module (Read-only)

Type
ModuleType

property number_of_bits

Resolution in number of bits (Read-only)

Type
int

property pin_type

Type of pin (Read-only)

Type
PinType

property range_max

Upper range limit in volts (Read-only)

Type
float

property range_min

Lower range limit in volts (Read-only)

Type
float

property serial

Module serial number (Read-only)

Type
int

property value

Base class does not implement value attribute

ModuleType

class canlib.canlib.iopin.**ModuleType**(*value*)

Enum used for return values in `kvIoPinGetInfo`

ANALOG = 2

Analog Add-on (4 inputs, 4 outputs).

DIGITAL = 1

Digital Add-on (16 inputs, 16 outputs).

INTERNAL = 4

Internal Digital module (1 input, 1 output).

RELAY = 3

Relay Add-on (8 inputs, 8 outputs).

PinType

class canlib.canlib.iopin.**PinType**(*value*)

Enum used for values in *Info*

ANALOG = 2

DIGITAL = 1

RELAY = 3

Relay

class canlib.canlib.iopin.**Relay**(*channel*, *pin*)

Bases: *IoPin*

property value

Value on relay, `0` for off, `1` for on

Script Container

SourceElement

class canlib.canlib.**SourceElement**(*name*, *contents*)

property contents

Alias for field number 1

property name

Alias for field number 0

Txe

class canlib.canlib.**Txe**(*path*)

The Txe class provides an interface to compiled t programs (.txe) files.

New in version 1.6.

property compiler_version

t compiler version number.

Type
VersionNumber

property date

Compilation date and time.

Type
datetime.datetime

property description

t program description.

Type
str

property file_version

.txe binary format version number.

Type
VersionNumber

property is_encrypted

true if the source and byte-code sections of the .txe binary have been encrypted.

Type
bool

property path

Path of compiled t program (.txe) file.

Type
str

property size_of_code

Size in bytes of byte-code section.

Type
int

property source

Yields name and content of the source files used to create the .txe binary file.

If the t source files used to create the .txe binary was included at compilation time, then this attribute will yield *SourceElement* tuples containing the name and content of the individual source files.

Sample usage:

```
for name, contents in txe.source:
  print('file name:{} contents:{}'.format(name, contents))
```

If the source and byte-code sections of the .txe binary have been encrypted then it's not possible to parse the source list and a *TxeFileIsEncrypted* exception will be raised.

If no source files have been included in the .txe binary then an empty iterator is returned.

Yields
SourceElement – Name and contents tuple.

Raises
TxeFileIsEncrypted – If the source and byte-code sections of the .txe binary have been encrypted.

Object Buffers

Support for accessing Object Buffers.

New in version 1.22.

class canlib.canlib.objbuf.**MessageFilter**(*code*, *mask*)

A message reception filter.

First set the mask bit to '1' for the bits you would like to filter on. Then set the code to the desired bitpattern. See *Code and Mask Format* for an explanation of the *code and mask* format.

Calling the *MessageFilter* with an id returns `True` if the id passes the filter.

```
>>> mf = MessageFilter(code=0b0100, mask=0b0110)
>>> mf(0b0110)
False
```

```
>>> mf(0b0101)
True
```

```
>>> mf(0b0010)
False
```

For use with *Response* object buffers.

New in version 1.22.

class canlib.canlib.objbuf.**Periodic**(*ch*, *period_us*, *frame=None*)

Periodic object buffer (also known as auto transmit buffer).

Returned from *canlib.Channel.allocate_periodic_objbuf()*

disable()

Disable this object buffer.

enable()

Enable this object buffer.

free()

Deallocate this object buffer.

This object buffer can not be referenced after this operation. To free all allocated object buffers, use *canlib.Channel.free_objbuf()*.

send_burst(*length*)

Send a burst of CAN frames from this object buffer.

The frames will be sent as fast as possible from the hardware. This function is intended for certain diagnostic applications.

Parameters
length (`int`) – Number of CAN frames to send.

set_frame(*frame*)

Define the CAN frame to be sent by the object buffer.

Parameters
frame (*Frame*) – The CAN frame to send.

set_msg_count(*count*)

Limit the total number of CAN frames sent.

When this periodic buffer is enabled, only `count` number of CAN frames will be sent (i.e. the periodic buffer will only be active for `count` number of periods).

When all frames have been sent, the msg_count is set to zero. If you would like to send five more frames, you need to make two calls:

```
>>> periodic_buffer.set_msg_count(5)
>>> periodic_buffer.enable()
```

Parameters
count (`int`) – Total number of CAN frames to send, Zero means infinite.

set_period(*period_us*)

Set interval in microseconds between each sent CAN frame.

Parameters
period_us (`int`) – Interval in microseconds between each sent CAN frames.

class canlib.canlib.objbuf.**Response**(*ch*, *filter=None*, *frame=None*, *rtr_only=False*)

Auto response object buffer.

Returned from *canlib.Channel.allocate_response_objbuf()*

The following example responds with a CAN frame with CAN ID 200 when a CAN frame with CAN ID 100 is received.:

```
>>> from canlib import canlib, Frame
>>> ch = canlib.openChannel(0)
>>> msg_filter = canlib.objbuf.MessageFilter(code=100, mask=0xFFFF)
>>> msg_filter(100)
True
>>> frame = Frame(id_=200, data=[1, 2, 3, 4])
>>> response_buf = ch.allocate_response_objbuf(filter=msg_filter, frame=frame)
>>> response_buf.enable()
```

New in version 1.22.

disable()

Disable this object buffer.

enable()

Enable this object buffer.

free()

Deallocate this object buffer.

This object buffer can not be referenced after this operation. To free all allocated object buffers, use *canlib.Channel.free_objbuf()*.

set_filter(*filter*)

Set message reception filter.

If no filter is set, any CAN ID will trigger the auto response.

Parameters
filter (*MessageFilter*) – Messages not matching the filter is ignored.

set_frame(*frame*)

Define the CAN frame to be sent by the object buffer.

Parameters
frame (*Frame*) – The CAN frame to send.

set_rtr_only(*value*)

Filter on CAN RTR (remote transmission request).

This complements the message reception filter (see *set_filter()*).

When set to `True`, the auto response buffer will only respond to remote requests (that also passes the message reception filter). When set to `False`, the auto response buffer will respond to both remote requests and ordinary data frames (that also passes the message reception filter).

Miscellaneous

dllversion()

canlib.canlib.**dllversion**()

Get the CANlib DLL version number

Parameters
None

Returns a *canlib.BetaVersionNumber* if the CANlib DLL is marked as beta (preview), otherwise returns *canlib.VersionNumber*.

Changed in version 1.6.

getErrorText()

canlib.canlib.**getErrorText**(*error_code*)

Return error text for supplied error_code

getNumberOfChannels()

canlib.canlib.**getNumberOfChannels**(*driver=False*)

Get number of available CAN channels.

Returns the number of available CAN channels in the computer. The virtual channels are included in this number.

In order to manually re-enumerate connected devices when a device has been added or removed, use *enumerate_hardware*.

Parameters
None

Returns
chanCount (int) – Number of available CAN channels

getVersion()

canlib.canlib.**getVersion**()

Get the CANlib DLL version number as a `str`

Deprecated since version 1.5: Use *dllversion* instead.

prodversion()

canlib.canlib.**prodversion**()

Get the CANlib Product version number

Parameters
None

Returns a *canlib.BetaVersionNumber* if the CANlib driver/DLL is marked as beta (preview), otherwise returns *canlib.VersionNumber*.

New in version 1.6.

initializeLibrary()

canlib.canlib.**initializeLibrary**()

Initialize CANlib library

Note: This initializes the driver and must be called before any other function in the CANlib DLL is used. This is handled in most cases by the Python wrapper. If you want to trigger a re-enumeration of connected devices, you should call *enumerate_hardware* instead.

Any errors encountered during library initialization will be "silent" and an appropriate error code will be returned later on when an API call that requires initialization is called.

reinitializeLibrary()

canlib.canlib.**reinitializeLibrary**()

Reinitializes the CANlib driver.

Convenience function that calls *unloadLibrary* and *initializeLibrary* in succession.

Warning: Calling *reinitializeLibrary* invalidates every canlib-object. Use at your own risk. You most likely would like to call *enumerate_hardware()* instead.

enumerate_hardware()

canlib.canlib.**enumerate_hardware**()

Enumerate connected Kvaser devices and rebuild list of available channels

Returns
The number of CAN channels enumerated.

New in version 1.13.

ScriptText

class canlib.canlib.**ScriptText**(*text*, *slot*, *time*, *flags*)

Text returned by *Channel.scriptGetText*

Subclass of built-in `str`, so it can be used just like a normal string.

It also has the following attributes:

Parameters

- **text** (`str`) – Text content.
- **slot** (`int`) – Which script-slot the text came from.
- **time** (`int`) – Timestamp of when the text was printed.
- **flags** (*Stat*) – Any status flags associated with the text.

New in version 1.7.

translateBaud()

canlib.canlib.**translateBaud**(*freq*)

Translate bitrate constant

This function translates the *canlib.Bitrate* and *canlib.BitrateFD* enums to their corresponding bus parameter values.

Parameters
freq – Any of the predefined *canlib.Bitrate* or *canlib.BitrateFD*.

Returns
A *BitrateSetting* object containing the actual values of frequency, tseg1, tseg2 etc.

unloadLibrary()

canlib.canlib.**unloadLibrary**()

Unload CANlib

Unload canlib and release all internal handles.

> **Warning:** Calling *unloadLibrary* invalidates every canlib-object. Use at your own risk.

1.6.7 kvadblib

Exceptions

KvdError

exception canlib.kvadblib.**KvdError**

Base class for exceptions raised by the kvadblib dll

KvdBufferTooSmall

exception canlib.kvadblib.**KvdBufferTooSmall**

Bases: *KvdError*

The buffer provided was not large enough to contain the requested data.

New in version 1.10.

status = -14

KvdDbFileParse

exception canlib.kvadblib.**KvdDbFileParse**

Bases: *KvdError*

The database file could not be parsed.

More information can be obtained by running *get_last_parse_error*.

New in version 1.10.

status = -15

KvdErrInParameter

exception canlib.kvadblib.**KvdErrInParameter**

Bases: *KvdError*

One or more of the parameters in call is erronous.

status = -3

KvdInUse

exception canlib.kvadblib.**KvdInUse**

Bases: *KvdError*

An item is in use.

status = -13

KvdNoAttribute

exception canlib.kvadblib.**KvdNoAttribute**

Bases: *KvdNotFound*

No attribute found.

status = -10

KvdNoMessage

exception canlib.kvadblib.**KvdNoMessage**

Bases: *KvdNotFound*

No message was found.

status = -4

KvdNoSignal

exception canlib.kvadblib.**KvdNoSignal**

Bases: *KvdNotFound*

No signal was found.

status = -5

KvdNoNode

exception canlib.kvadblib.**KvdNoNode**

Bases: *KvdNotFound*

Could not find the database node.

status = -9

KvdNotFound

exception canlib.kvadblib.**KvdNotFound**

Bases: *KvdError*

KvdOnlyOneAllowed

exception canlib.kvadblib.**KvdOnlyOneAllowed**

Bases: *KvdError*

An identical kvaDbLib structure already exists.

Only one database at a time can be used).

status = -11

SignalNotFound

exception canlib.kvadblib.**SignalNotFound**(*text*)

Bases: *CanlibException*

Attribute

class canlib.kvadblib.**Attribute**(*db*, *handle*)

Factory for creating different types of attributes.

This class is also the base class and thus contains all common properties.

property name

Name of attribute.

Type
str

property value

Attribute value

Attribute Definitions

AttributeDefinition

class canlib.kvadblib.**AttributeDefinition**(*db*, *handle*, *definition=None*)

Factory for creating different types of attribute definitions.

This class is also the base class and thus contains all common properties.

property name

Name of attribute definition.

Type
str

property owner

Return attribute owner

Type
AttributeOwner

DefaultDefinition

class canlib.kvadblib.**DefaultDefinition**(*default*)

Attribute definition for attributes defined using only default.

property default

Alias for field number 0

EnumDefaultDefinition

class canlib.kvadblib.**EnumDefaultDefinition**(*default*, *enums*)

Attribute definition for enumeration attributes.

Holds a definition using default and key-value pairs.

property default

Alias for field number 0

property enums

Alias for field number 1

EnumDefinition

class canlib.kvadblib.**EnumDefinition**(*db*, *handle*, *definition=None*)

Definition of an enum attribute.

Parameters

- **db** (*Dbc*) – Database that holds attribute definitions
- **definition** (*EnumDefaultDefinition*) – default value and enums

add_enum_definition(*enums*)

Add enum definitions.

Parameters
enums (*dict*) – key - value pair(s), example: {'empty': 0}

property definition

Return attribute definition

Changed in version 1.6.

Type
EnumDefaultDefinition

FloatDefinition

class canlib.kvadblib.**FloatDefinition**(*db*, *handle*, *definition=None*)

Definition of a float attribute.

Parameters

- **db** (*Dbc*) – Database that holds attribute definitions
- **definition** (*MinMaxDefinition*) – Min, max and default values

property definition

Attribute definition

Type
MinMaxDefinition

IntegerDefinition

class canlib.kvadblib.**IntegerDefinition**(*db*, *handle*, *definition=None*)

Definition of an integer attribute.

Parameters

- **db** (*Dbc*) – Database that holds attribute definitions
- **definition** (*MinMaxDefinition*) – Min, max and default values

property **definition**

Return attribute definition

Type
MinMaxDefinition

HexDefinition

class canlib.kvadblib.**HexDefinition**(*db*, *handle*, *definition=None*)

Definition of an hex attribute.

Parameters

- **db** (*Dbc*) – Database that holds attribute definitions
- **definition** (*MinMaxDefinition*) – Min, max and default values

New in version 1.20.

MinMaxDefinition

class canlib.kvadblib.**MinMaxDefinition**(*default*, *min*, *max*)

Attribute definition for attributes defined using default, min and max.

property **default**

Alias for field number 0

property **max**

Alias for field number 2

property **min**

Alias for field number 1

StringDefinition

class canlib.kvadblib.**StringDefinition**(*db*, *handle*, *definition=None*)

Definition of a string attribute.

Parameters

- **db** (*Dbc*) – Database that holds attribute definitions
- **definition** (*DefaultDefinition*) – default value

property definition

Return attribute definition

Type
DefaultDefinition

dbc

dbc.**DATABASE_FLAG_J1939 = 1**

Dbc

class canlib.kvadblib.**Dbc**(*filename=None*, *name=None*, *protocol=None*)

Holds a dbc database.

There are three ways to create a database:

1. To load data from an existing database file, only set filename to the database filename.
2. To add an empty database, set only name.
3. To load data from an existing database file and give it a new name, set name to the new name and filename to the existing database filename.

Either a name or a filename must be given.

Parameters

- **filename** (*str, optional*) – The existing database file is read.
- **name** (*str, optional*) – The database name will be set.

Raises
KvdDbFileParse – If the database file can't be parsed.

When parsing a database file fails, *get_last_parse_error* can be called:

```
try:
    kvadblib.Dbc(filename="database.dbc")
except kvadblib.KvdDbFileParse:
    print(kvadblib.get_last_parse_error())
```

Changed in version 1.10: *Dbc* now raises specific errors depending on the situation.

attribute_definitions()

Return a generator over all database attribute definitions.

attributes()

Return a generator over all database attributes.

New in version 1.6.

close()

Close an open database handle.

delete_attribute(*name*)

Delete attribute from database.

New in version 1.6.

delete_attribute_definition(*name*)

Delete attribute definition from database.

New in version 1.7.

delete_message(*message*)

Delete message from database.

Parameters
message (*Message*) – message to be deleted

delete_node(*node*)

Delete node from database.

Parameters
node (*Node*) – node to be deleted

property flags

Get the database flags.

E.g. DATABASE_FLAG_J1939

get_attribute_definition_by_name(*name*)

Find attribute definition using name.

Parameters
name (*str*) – name of attribute definition

Returns an attribute definition object depending on type, e.g. if the type is AttributeType.INTEGER, an `IntegerAttributeDefinition` is returned.

get_attribute_value(*name*)

Return attribute value

If the attribute is not set on the database, we return the attribute definition default value.

New in version 1.6.

get_message(*id=None*, *name=None*)

Find message by id or name

If both id and name is given, both most match.

Note that bit 31 of the id indicates if the message has an extended id or not.

Parameters

- **id** (*int*) – message id to look for
- **name** (*str*) – message name to look for

Returns
Message

Raises
KvdNoMessage – If no match was found, or if none of `id` and *name* were given.

get_message_by_id(*id*, *flags*)

Find message by id.

Parameters

- **id** (*int*) – message id to look for

- **flags** (*int*) – message flags, e.g. `kvadblib.MessageFlag.EXT` (Note that `kvadblib.MessageFlag.EXT` != `canlib.MessageFlag.EXT`)

Returns
Message

Raises
KvdNoMessage – If no match was found.

get_message_by_name(*name*)

Find message by name

Parameters
name (*str*) – message name to look for

Returns
Message

Raises
KvdNoMessage – If no match was found.

get_message_by_pgn(*can_id*)

Find message using the PGN part of the given CAN id.

Note: The `can_id` must have the *MessageFlag.EXT* flag bit set (bit 31, 0x80000000)

```
dbc = kvadblib.Dbc("My_j1939_database.dbc")
can_id = 0xff4200
can_id |= kvadblib.MessageFlag.EXT
dbc.get_message_by_pgn(can_id)
```

Parameters
can_id (*int*) – CAN id containing PGN to look for

Returns
Message

Raises
KvdNoMessage – If no match was found.

New in version 1.18.

get_node_by_name(*name*)

Find node by name

Parameters
name (*str*) – node name to look for

Returns
Node

Raises
KvdNoNode – If no match was found.

interpret(*frame*, *j1939=False*)

Interprets a given *canlib.Frame* object, returning a *BoundMessage*.

messages(*show_all=True*)

Return a generator of all database messages.

If you would like to omit the special message 'VECTOR__INDEPENDENT_SIG_MSG', which is used to collect removed signals under, set show_all to False or use the Dbc iterator directly:

```
db = kvadblib.Dbc(filename='mydb.dbc')
for message in db:
    print(msg)
```

Parameters

show_all (bool) – If true, all messages, including any special message such as 'VECTOR__INDEPENDENT_SIG_MSG' will be returned

Changed in version 1.8: Added argument show_all.

property name

The current database name (read-only)

Type

str

new_attribute_definition(*name*, *owner*, *type*, *definition*)

Create a new attribute definition in the database.

The owner specify where the attribute is applicable, e.g. *AttributeOwner.MESSAGE* specifies that this attribute is only applicable on messages (*Message*).

Parameters

- **name** (*str*) – a unique name.
- **owner** (*AttributeOwner*) – the owner type

Returns

AttributeDefinition

new_message(*name*, *id*, *flags=0*, *dlc=None*, *comment=None*)

Create a new message in the database.

Parameters

- **name** (*str*) – name of message
- **id** (*int*) – message id
- **flags** (*int, optional*) – message flags, e.g. kvadblib.MessageFlag.EXT

Returns

Message

new_node(*name*, *comment=None*)

Create a new node in the database.

Parameters

- **name** (*str*) – name of message
- **comment** (*str, optional*) – message comment

Returns

Node

node_in_signal(*node*, *signal*)

Check if signal has been added to node.

Returns
True – signals contains node False: otherwise

nodes()

Return a generator containing all database nodes.

property protocol

The database protocol

Type
ProtocolType

set_attribute_value(*name*, *value*)

Set value of attribute *name* on database.

If no attribute called *name* is set on database, attach a database attribute from the database attribute definition first.

New in version 1.6.

write_file(*filename*)

Write a database to file.

Parameters
filename (*str*) – file to write database to

get_last_parse_error()

canlib.kvadblib.**get_last_parse_error**()

Can be used to get the specific reason why *KvdDbFileParse* was raised.

Returns
`str` – Error message from the parser.

New in version 1.10.

Enumerations

AttributeOwner

class canlib.kvadblib.**AttributeOwner**(*value*)

An enumeration.

DB = 1

Database owner

ENV = 5

Environment owner

INVALID = 0

Invalid owner

MESSAGE = 2
Message owner

NODE = 3
Node owner

SIGNAL = 4
Signal owner

AttributeType

class canlib.kvadblib.**AttributeType**(*value*)
An enumeration.

ENUM = 4

FLOAT = 3

HEX = 2

INTEGER = 1

INVALID = 0

STRING = 5

Error

class canlib.kvadblib.**Error**(*value*)
kvaDbErr_xxx

BUFFER_TOO_SMALL = -14
The buffer provided was not large enough to contain the requested data.

DATABASE_INTERNAL = -8
An internal error occured in the database handler.

DB_FILE_OPEN = -7
Could not open the database file.

DB_FILE_PARSE = -15
Could not parse the database file

FAIL = -1
General failure.

INTERNAL = -6
An internal error occured in the library.

IN_USE = -13
An item is in use

NO_ATTRIB = -10
No attribute found

NO_DATABASE = -2

No database was found.

NO_MSG = -4

No message was found.

NO_NODE = -9

Could not find the database node.

NO_SIGNAL = -5

No signal was found.

ONLY_ONE_ALLOWED = -11

An identical kvaDbLib structure already exists (and only one database at a time can be used).

PARAM = -3

One or more of the parameters in call is erronous.

WRONG_OWNER = -12

Wrong owner for attribute

MessageFlag

class canlib.kvadblib.**MessageFlag**(*value*)

An enumeration.

EXT = 2147483648

Message is an extended CAN message

J1939 = 1

Message uses J1939 protocol

WAKEUP = 2

Message is a wakeup frame, currently not used

ProtocolType

class canlib.kvadblib.**ProtocolType**(*value*)

An enumeration.

AFDX = 7

BEAN = 5

CAN = 0

CANFD = 9

ETHERNET = 6

FLEXRAY = 4

J1708 = 8

LIN = 2

MOST = 3

UNKNOWN = 10

Unknown or not specified protocol

VAN = 1

SignalByteOrder

class canlib.kvadblib.**SignalByteOrder**(*value*)

An enumeration.

INTEL = 0

MOTOROLA = 1

SignalMultiplexMode

class canlib.kvadblib.**SignalMultiplexMode**(*value*)

An enumeration.

MUX_INDEPENDENT = -1

Multiplex mode value of an independent signal

MUX_SIGNAL = -2

Multiplex mode value of a multiplexer signal

SIGNAL = 0

SignalType

class canlib.kvadblib.**SignalType**(*value*)

An enumeration.

DOUBLE = 4

Double, strictly 64 bit long

ENUM_SIGNED = 101

ENUM_UNSIGNED = 102

FLOAT = 3

Float, strictly 32 bit long

INVALID = 0

Invalid representation

SIGNED = 1

Signed integer

UNSIGNED = 2

Unsigned integer

Frame Box

FrameBox

class canlib.kvadblib.**FrameBox**(*db*, *messages=()*)

Helper class for sending signals

This class allows sending signals without worrying about what message they are defined in. It does this by binding a message and all its signals to the same *canlib.Frame* object.

Objects are created by giving them a *Dbc* database, and optionally a list of messages (either names or *Message* objects):

```
db = Dbc(...)
framebox = FrameBox(db, messages=('Msg0','Msg1'))
```

Messages can also be added after instantiation with *add_message*:

```
framebox.add_message('Msg0', 'Msg1')
```

Then setting signal values for any added message is done with:

```
framebox.signal('Sig0').phys = 7
framebox.signal('Sig1').phys = 20
```

Once all values are set, they can easily be sent via the channel `channel` with:

```
for frame in framebox.frames():
  channel.write(frame)
```

Any `Framebox` methods that return messages requires the message to have been added to the framebox, either with the `messages` constructor argument or with *add_message*. Likewise, any methods that return signals require the signal's message to have been added.

add_message(*message*)

Add a message to the framebox

The message will be available for all future uses of *FrameBox.message* and *FrameBox.messages*, and all its signals will be available for uses of *FrameBox.signal* and *FrameBox.signals*.

The `message` argument can either be a message name, or a *canlib.kvadblib.Message* object.

frames()

Iterate over all frames of the signals/messages from this *FrameBox*

message(*name*)

Retrieves a message by name

Returns a *BoundMessage* that shares its *canlib.Frame* object with its child signals.

messages()

Iterator over all messages that this *FrameBox* is aware of

signal(*name*)

Retrieves a signal by name

Returns a *BoundSignal* that shares its *canlib.Frame* object with its parent message and sibling signals.

signals()

Iterator over all signals that this *FrameBox* is aware of

BoundMessage

class canlib.kvadblib.**BoundMessage**(*message*, *frame*)

A CAN data object that manipulates data through signals.

BoundSignal

class canlib.kvadblib.**BoundSignal**(*signal*, *frame*)

property is_enum

Whether this signal is an enum-signal

New in version 1.7.

Type
bool

property name

Signal's name string

Type
str

property phys

Signal's physical value

Type
int or float

property raw

Signal's raw value

Type
int

property unit

Signal's unit string

Type
str

property value

Signal's value

If the signal is an enum-signal (i.e. the signal has a defined value table), then the enum name is returned if found, otherwise the signals raw value (*raw*) is returned. If the signal is not an enum-signal, the signals physical value (*phys*) is returned.

New in version 1.7.

Message

class canlib.kvadblib.**Message**(*db*, *handle*, *name=None*, *id=None*, *flags=None*, *dlc=None*, *comment=None*)

Database message, holds signals.

asframe()

Returns a *Frame* object with flags and empty data matching this message

Changed in version 1.21: Now includes support for CAN FD signals in dbc file introduced in CANlib SDK v5.39 by using the *canflags* properties.

attributes()

Return a generator over all message attributes.

bind(*frame=None*)

Bind this message to a frame

Creates a new BoundMessage object representing this message bound to the given Frame object, or a new Frame object if `frame` is `None`.

property canflags

Relevant message attributes expressed as *canlib.MessageFlag*

Note that *canlib.MessageFlag.BRS* will never be returned for non-CAN FD frames, even though the CANFD_BRS attribute was set in the .dbc file.

New in version 1.21.

Type
canlib.MessageFlag

property comment

Message comment

Type
`str`

delete_attribute(*name*)

Delete attribute from message.

delete_signal(*signal*)

Delete signal from message.

Parameters
signal (*Signal*) – signal to be deleted

property dlc

The message dlc

Type
`int`

property flags

The message flags

Type
MessageFlag

get_attribute_value(*name*)

Return attribute value

If the attribute is not set on the message, we return the attribute definition default value.

Changed in version 1.18: When an EnumAttribute is not set, the default value will now be returned as `int` (instead of `EnumValue` with empty *name*).

get_signal(*name*)

Find signal in message by name.

get_signal_by_name(*name*)

Find signal in message by name.

property id

The message identifier

Type
`int`

property name

The message name

Type
`str`

new_signal(*name*, *type=SignalType.UNSIGNED*, *byte_order=SignalByteOrder.INTEL*, *mode=SignalMultiplexMode.SIGNAL*, *representation=None*, *size=None*, *scaling=None*, *limits=None*, *unit=None*, *comment=None*, *enums=None*)

Create and add a new signal to the message.

property qualified_name

The qualified message name

Returns database and message names separated by a dot.

Type
`str`

property send_node

The send node for this message.

Type
Node

set_attribute_value(*name*, *value*)

Set value of attribute 'name' on message.

If no attribute called 'name' is set on message, attach a message attribute from the database attribute definition first.

signals()

Return a generator of all signals in message.

Node

class canlib.kvadblib.**Node**(*db*, *handle*, *name=None*, *comment=None*)

Database Node

attributes()

Return a generator over all message attributes.

property comment

The node's comment

Type
str

delete_attribute(*name*)

Delete attribute from node.

get_attribute_value(*name*)

Return attribute value

If the attribute is not set on the message, we return the attribute definition default value.

property name

The node's name

Type
str

set_attribute_value(*name*, *value*)

Set value of attribute 'name' on node.

If no attribute called 'name' is set on node, attach a node attribute from the database attribute definition first.

Signals

EnumSignal

class canlib.kvadblib.**EnumSignal**(*db*, *message*, *sh*, *name=None*, *type=None*, *byte_order=None*, *mode=None*, *size=None*, *scaling=None*, *limits=None*, *unit=None*, *comment=None*, *enums=None*)

Database signal of type enum, holds meta data about a signal.

Changed in version 1.17: default settings byte_order=SignalByteOrder.INTEL and mode=SignalMultiplexMode.SIGNAL chaged to None.

add_enum_definition(*enums*)

Add enums dictionary to definition.

property enums

Signal enum definition dictionary

Type
dict

Signal

class canlib.kvadblib.**Signal**(*db*, *message*, *sh*, *name=None*, *type=None*, *byte_order=None*, *mode=None*, *representation=None*, *size=None*, *scaling=None*, *limits=None*, *unit=None*, *comment=None*)

Database signal, holds meta data about a signal

add_node(*node*)

Add receiving node to signal.

attributes()

Return a generator over all signal attributes.

bind(*frame=None*)

Bind this signal to a frame

Creates a new BoundSignal object representing this signal bound to the given Frame object, or a new Frame object if `frame` is `None`..

property byte_order

Get the signal byte order encoding.

Type
SignalByteOrder

property comment

Get the signal comment.

Type
str

data_from(*can_data*, *phys=None*, *raw=None*)

Convert a raw or physical value into CAN data bytes.

delete_attribute(*name*)

Delete attribute from signal.

get_attribute_value(*name*)

Return attribute value

If the attribute is not set on the signal, we return the attribute definition default value.

property limits

Get message min and max values

Type
ValueLimits

property mode

property name

Get the signal name.

Type
str

nodes()

Return a generator over all receiving nodes of the signal.

phys_from(*can_data*)

Return signals physical value from data

property qualified_name

Get the qualified signal name.

Returns database, message and signal names separated by dots.

Type
str

raw_from(*can_data*)

Return signals raw value from data

remove_node(*node*)

Remove receiving node from signal.

property scaling

Get the signals factor and offset

Type
ValueScaling

set_attribute_value(*name*, *value*)

Set value of attribute 'name' on signal.

If no attribute called 'name' is set on signal, attach a signal attribute from the database attribute definition first.

property size

Get the signals startbit and length

Type
ValueSize

property type

Get the signal representation type.

Type
SignalType

property unit

Get the signal unit

Type
str

ValueLimits

class canlib.kvadblib.**ValueLimits**(*min*, *max*)

property max

Alias for field number 1

property min

Alias for field number 0

ValueScaling

class canlib.kvadblib.**ValueScaling**(*factor*, *offset*)

property factor
Alias for field number 0

property offset
Alias for field number 1

ValueSize

class canlib.kvadblib.**ValueSize**(*startbit*, *length*)

property length
Alias for field number 1

property startbit
Alias for field number 0

Miscellaneous

bytes_to_dlc()

canlib.kvadblib.**bytes_to_dlc**(*num_bytes*, *protocol*)
Convert number of bytes to DLC for given protocol.

dlc_to_bytes()

canlib.kvadblib.**dlc_to_bytes**(*dlc*, *protocol*)
Convert DLC to number of bytes for given protocol.

dllversion()

canlib.kvadblib.**dllversion**()
Get the kvaDbLib DLL version number as a `VersionNumber`

get_protocol_properties()

canlib.kvadblib.**get_protocol_properties**(*prot*)
Get the signal protocol_properties.

1.6.8 kvamemolibxml

Exceptions

KvaError

exception canlib.kvamemolibxml.**KvaError**

Base class for exceptions raised by the kvamemolibxml library

Looks up the error text in the kvamemolibxml dll and presents it together with the error code.

Confiurations

Configuration

class canlib.kvamemolibxml.**Configuration**(*xml=None*, *lif=None*)

Configuration data for Kvaser devices

It is usually preferred to create objects of this class with one of the functions:

- *load_xml*
- *load_xml_file*
- *load_lif*
- *load_lif_file*

The XML and param.lif representation of this configuration can be accessed with the *xml* and *lif* attributes, respectively.

Two *Configuration* objects can be tested for equality:

```
config1 == config2
```

This will test whether the objects are equivalent: whether they have the same param.lif representation.

Finally, the configuration can be validated with *Configuration.validate*:

```
errors, warnings = configuration.validate()
for error in errors:
    print(error)
for warning in warnings:
    print(warning)
if errors:
    raise ValueError("Invalid configuration")
```

property lif

The param.lif representation of this configuration

Type
bytes

validate()

Validate this configuration

Validates the XML representation of this configuration, and returns a tuple `(errors, warnings)` where `errors` is a `list` of *ValidationError* and `warnings` is a `list` *ValidationWarning*.

property xml

The XML representation of this configuration

Type
`str`

load_lif()

canlib.kvamemolibxml.**load_lif**(*lif_bytes*)

Create a *Configuration* from a param.lif `bytes`

Parameters
lif_bytes (`bytes`) – Binary configuration in param.lif format

load_lif_file()

canlib.kvamemolibxml.**load_lif_file**(*filepath*)

Like *load_lif* but takes a path to a file containing the lif configuration

load_xml()

canlib.kvamemolibxml.**load_xml**(*xml_string*)

Create a *Configuration* from an xml `string`

Parameters
xml_string (`str`) – XML configuration

load_xml_file()

canlib.kvamemolibxml.**load_xml_file**(*filepath*)

Like *load_lif* but takes a path to a file containing the XML configuration

ValidationMessage

class canlib.kvamemolibxml.**ValidationMessage**(*code*, *text*)

Validation code and message.

Parameters

- **code** (`int`) – Valdiation status code.
- **text** (`str`) – Validation message.

property code

Alias for field number 0

property text

Alias for field number 1

ValidationErrorMessage

class canlib.kvamemolibxml.**ValidationErrorMessage**(*code*, *text*)

An error found during validation

ValidationWarningMessage

class canlib.kvamemolibxml.**ValidationWarningMessage**(*code*, *text*)

A warning found during validation

Enumerations

Error

class canlib.kvamemolibxml.**Error**(*value*)

An enumeration.

ATTR_NOT_FOUND = -3

Failed to find an attribute in a node

ATTR_VALUE = -4

The attribute value is not correct, e.g. whitespace after a number.

DTD_VALIDATION = -11

The XML settings do not follow the DTD.

ELEM_NOT_FOUND = -5

Could not find a required element

FAIL = -1

Generic error

INTERNAL = -20

Internal errors, e.g. null pointers.

OK = 0

OK

POSTFIXEXPR = -9

The trigger expression could not be parsed

SCRIPT_ERROR = -12

t-script related errors, e.g. file not found.

VALUE_CONSECUTIVE = -8

The values are not consecutive; usually idx attributes

VALUE_RANGE = -6

The value is outside the allowed range

VALUE_UNIQUE = -7

The value is not unique; usually idx attributes

XML_PARSER = -10

The XML settings contain syntax errors.

classmethod from_number(*number*)

Create *Error* object from number.

If the number is not defined in the class, return the number instead. This is used in order to be forward compatible with new codes in the dll.

New in version 1.19.

ValidationError

class canlib.kvamemolibxml.**ValidationError**(*value*)

An enumeration.

ABORT = -2

Too many errors, validation aborted.

DISK_FULL_STARTS_LOG = -7

A trigger on disk full starts the logging.

ELEMENT_COUNT = -13

Too many or too few elements of this type.

EXPRESSION = -16

A general trigger expression found during parsing.

FAIL = -1

Generic error.

MULTIPLE_EXT_TRIGGER = -5

There are more than one external trigger defined.

MULTIPLE_START_TRIGGER = -6

There are more than one start up trigger defined.

NUM_OUT_OF_RANGE = -8

A numerical value is out of range.

OK = 0

OK

PARSER = -14

A general error found during parsing.

SCRIPT = -15

A general t-script error found during parsing.

SCRIPT_CONFLICT = -12

More than one active script is set as 'primary'.

SCRIPT_NOT_FOUND = -9

A t-script file could not be opened.

SCRIPT_TOO_LARGE = -10

A t-script is too large for the configuration.

SCRIPT_TOO_MANY = -11

Too many active t-scripts for selected device.

SILENT_TRANSMIT = -3

Transmit lists used in silent mode.

UNDEFINED_TRIGGER = -4

An undefined trigger is used in an expression.

classmethod from_number(*number*)

Create *ValidationError* object from number.

If the number is not defined in the class, return the number instead. This is used in order to be forward compatible with new codes in the dll.

New in version 1.19.

ValidationWarning

class canlib.kvamemolibxml.**ValidationWarning**(*value*)

An enumeration.

ABORT = -100

Too many warnings, validation aborted.

DISK_FULL_AND_FIFO = -102

A trigger on disk full used with FIFO mode.

IGNORED_ELEMENT = -103

This XML element was ignored.

MULTIPLE_EXT_TRIGGER = -104

Using more than one external trigger requires firmware version 3.7 or better.

NO_ACTIVE_LOG = -101

No active logging detected.

classmethod from_number(*number*)

Create *ValidationWarning* from number.

If the number is not defined in the class, return the number instead. This is used in order to be forward compatible with new codes in the dll.

New in version 1.19.

Miscellaneous

dllversion()

canlib.kvamemolibxml.**dllversion**()

Get the kvaMemoLibXML DLL version number.

Returns

canlib.VersionNumber

kvaBufferToXml()

canlib.kvamemolibxml.**kvaBufferToXml**(*conf_lif*)

Convert a buffer containg param.lif to XML settings.

Scripts from the param.lif will be written to current working directory.

Parameters
conf_lif (str) – Buffer containing param.lif to convert.

Returns
str – Buffer containing converted XML settings.

kvaXmlToBuffer()

canlib.kvamemolibxml.**kvaXmlToBuffer**(*conf_xml*)

Convert XML settings to param.lif buffer.

Parameters
conf_xml (str) – XML settings to convert.

Returns
str – Buffer containing converted param.lif.

kvaXmlToFile()

canlib.kvamemolibxml.**kvaXmlToFile**(*xml_filename*, *binary_filename*)

Convert XML file to binary configuration file.

Parameters

- **xml_filename** (str) – Filename of file containing the XML settings.
- **binary_filename** (str) – Filename of binary configuration.

kvaXmlValidate()

canlib.kvamemolibxml.**kvaXmlValidate**(*conf_xml*)

Validate a buffer with XML settings.

Parameters
conf_xml (str) – string containing the XML settings to validate.

Returns

tuple – containing

- int: Number of XML validation errors.
- int: Number of XML validation warnings.

xmlGetLastError()

canlib.kvamemolibxml.**xmlGetLastError**()

Get the last error message (if any).

Returns

tuple – containing

- str: Error message associated with the error code.
- *Error*: Error code.

Changed in version 1.19: Returned error code is now an enum.

xmlGetValidationError()

canlib.kvamemolibxml.**xmlGetValidationError**()

Get validation errors (if any).

Call after kvaXmlValidate() until return status is KvaXmlValidationStatusOK.

Returns

tuple – containing

- int: Validation error status code.
- str: Validation error status message.

Changed in version 1.19: Returned status code is now an enum.

xmlGetValidationStatusCount()

canlib.kvamemolibxml.**xmlGetValidationStatusCount**()

Get the number of validation statuses (if any).

Call after kvaXmlValidate().

Returns

tuple – containing

- int: Number of XML validation errors.
- int: Number of XML validation warnings.

xmlGetValidationWarning()

canlib.kvamemolibxml.**xmlGetValidationWarning**()

Get validation warnings (if any).

Call after kvaXmlValidate() until return status is KvaXmlValidationStatusOK.

Returns

tuple – containing

- int: Validation warning status code.

- str: Validation warning status message.

Changed in version 1.19: Returned status code is now an enum.

1.6.9 kvlclib

Exceptions

KvlcError

exception canlib.kvlclib.**KvlcError**

Bases: *DllException*

Base class for exceptions raised by the kvlclib module.

Looks up the error text in the kvlclib dll and presents it together with the error code and the wrapper function that triggered the exception.

KvlcEndOfFile

exception canlib.kvlclib.**KvlcEndOfFile**

Bases: *KvlcError*

Exception used when kvlclib returns *Error.EOF*.

Exception used when end of file is reached on input file.

status = -3

KvlcFileExists

exception canlib.kvlclib.**KvlcFileExists**

Bases: *KvlcError*

Exception used when kvlclib returns *Error.FILE_EXISTS*.

File exists, set *Property.OVERWRITE* to overwrite

New in version 1.17.

status = -6

KvlcNotImplemented

exception canlib.kvlclib.**KvlcNotImplemented**

Bases: *KvlcError*, NotImplementedError

Exception used when kvlclib returns *Error.NOT_IMPLEMENTED*.

status = -4

Converter

class canlib.kvlclib.**Converter**(*filename*, *file_format*)

A kvlclib converter

This class wraps all kvlclib functions related to converters, and saves you from keeping track of a handle and passing that to the functions.

`kvlcCreateConverter` and `kvlcDeleteConverter` are not wrapped as they are called when Converter objects are created and deleted, respectively. However, if it is necessary to force the converter to write its files, *flush* can be used to simulate destroying and recreating the converter object.

Parameters

- **filename** (`str`) – Name of output file
- **file_format** (*FileFormat* | *WriterFormat*) – A supported output format

---

Note: No more than 128 converters can be open at the same time.

Changed in version 1.18: The `file_format` parameter now accepts *WriterFormat* as well.

IsDataTruncated()

Get truncation status.

Deprecated since version 1.5: Use *isDataTruncated* instead.

IsOutputFilenameNew()

Check if the converter has created a new file.

Deprecated since version 1.5: Use *isOutputFilenameNew* instead.

IsOverrunActive()

Get overrun status.

Deprecated since version 1.5: Use *isOverrunActive* instead.

addDatabaseFile(*filename*, *channel_mask*)

Add a database file.

Converters with the property *Property.SIGNAL_BASED* will match events against all entries in the database and write signals to the output file.

Parameters

- **filename** (`str`) – Path to database file (.dbc)
- **channel_mask** (*ChannelMask*) – Channels to use database on

attachFile(*filename*)

Attach file to be included in the output file.

E.g. used to add a database or a movie to the output.

Note that the output format must support the property *Property.ATTACHMENTS*.

Parameters
filename (`str`) – Path to file to be attached

convertEvent()

Convert next event.

Convert one event from input file and write it to output file.

eventCount()

Get extimated number of events left.

Get the estimated number of remaining events in the input file. This can be useful for displaying progress during conversion.

feedLogEvent(*event*)

Feed one event to the converter.

Used when reading log files directly from device.

feedNextFile()

Prepare for new file

Notify the converter that next event in *feedLogEvent()* will come from another file. Used when reading log files directly from device.

E.g. use this function with *FileFormat.MEMO_LOG* when using KVMLIB to read events from a Kvaser Memorator connected to USB.

New in version 1.18.

flush()

Recreate the converter so changes are saved to disk

Converters do not write changes to disk until they are deleted. This method deletes and recreates the underlying C converter, without needing to recreate the Python object.

getDlcMismatch()

Return a dictionary with id, DLC with number of mismatched messages

If any DLC mismatch occurred during conversion (which can be seen using *isDlcMismatch*) this function returns a dictonary with the tuple message id and message DLC as key, and the number of times that triggered the mismatch as value.

getOutputFilename()

Get the filename of the current output file.

getProperty(*wr_property*)

Get current value for a writer property.

Parameters
wr_property (*Property*) – Writer property to get

getPropertyDefault(*wr_property*)

Get default property.

Deprecated since version 1.5: Use *WriterFormat.getPropertyDefault* instead.

isDataTruncated()

Get truncation status.

Truncation occurs when the selected output converter can't write all bytes in a data frame to file. This can happen if CAN FD data is extracted to a format that only supports up to 8 data bytes, e.g. *FileFormat.KME40*.

Truncation can also happen if *Property.LIMIT_DATA_BYTES* is set to limit the number of data bytes in output.

Returns
True if data has been truncated

`isDlcMismatch`()

Get DLC mismatch status.

DLC mismatch occurs when a CAN id is found in the database but the DLC differs from the DLC in the message.

`isOutputFilenameNew`()

Check if the converter has created a new file.

This is only true once after a a new file has been created. Used when splitting output into multiple files.

`isOverrunActive`()

Get overrun status.

Overruns can occur during logging with a Memorator if the bus load exceeds the logging capacity. This is very unusual, but can occur if a Memorator runs complex scripts and triggers.

`isPropertySupported`(*wr_property*)

Check if specified wr_property is supported by the current format.

Deprecated since version 1.5: Use *WriterFormat.isPropertySupported* instead.

`nextInputFile`(*filename*)

Select next input file.

`resetDlcMismatch`()

Reset DLC mismatch status.

`resetOverrunActive`()

Reset overrun status.

`resetStatusTruncated`()

Reset data trunctation status.

`setInputFile`(*filename*, *file_format*)

Select input file.

Parameters

- **filename** (*string*) – Name of input file
- **file_format** (*FileFormat* | *ReaderFormat*) – A supported input format

Changed in version 1.16: The `file_format` parameter now accepts *ReaderFormat* as well.

Changed in version 1.18: If `filename` is `None`, the format for *feedLogEvent* is set.

`setProperty`(*wr_property*, *value*)

Set a property value.

Parameters
wr_property (*Property*) – Writer property to be set

Enumerations

ChannelMask

class canlib.kvlclib.**ChannelMask**(*value*)

Masking channels

The *ChannelMask* is used in *Converter.addDatabaseFile* to indicate which channels to use.

Multiple channels may be specified using |, e.g. to specify channel one and three use:

```
channel_one_and_three = ChannelMask.ONE | ChannelMask.THREE
```

Changed in version 1.20: Added *ALL* as a convenience.

ALL = 65535

Mask for all channels

FIVE = 16

Mask for fifth channel

FOUR = 8

Mask for fourth channel

ONE = 1

Mask for first channel

THREE = 4

Mask for third channel

TWO = 2

Mask for second channel

Error

class canlib.kvlclib.**Error**(*value*)

An enumeration.

BUFFER_SIZE = -15

Supplied buffer too small to hold the result.

CONVERTING = -14

Call failed since conversion is running.

EOF = -3

End of input file reached.

FAIL = -1

Generic error.

FILE_ERROR = -5

File I/O error.

FILE_EXISTS = -6

Output file already exists.

FILE_TOO_LARGE = -10

File size too large for specified format.

INTERNAL_ERROR = -7

Unhandled internal error.

INVALID_LOG_EVENT = -30

Event is unknown to converter.

MIXED_ENDIANESS = -33

Wrong data type in MDF.

NOT_IMPLEMENTED = -4

Not implemented.

NO_FREE_HANDLES = -12

Too many open KvlcHandle handles.

NO_INPUT_SELECTED = -13

Missing call to kvlcSetInputFile or kvlcFeedSelectFormat.

NO_TIME_REFERENCE = -31

Required timestamp missing.

NULL_POINTER = -8

Unexpected null pointer.

PARAM = -2

Error in supplied parameters.

RESULT_TOO_BIG = -34

Result is too big for an out-parameter

TIME_DECREASING = -32

Decreasing time between files.

TYPE_MISMATCH = -11

Supplied parameter has incorrect type.

FileFormat

class canlib.kvlclib.**FileFormat**(*value*)

FILE_FORMAT_xxx

Format used for input and output, used in *Converter.setInputFile()*.

Note: Not all formats are valid as both output and input format.

CSV = 4

Output file format.

CSV_SIGNAL = 100

Output file format.

DEBUG = 1000

Reserved for debug.

DIADEM = 110

Output file format.

FAMOS = 105

Output file format.

FAMOS_XCP = 201

Output file format.

INVALID = 0

Invalid file format

J1587 = 103

Output file format.

J1587_ALT = 104

Obsolete.

KME24 = 1

Input and output file format.

KME25 = 2

Input and output file format.

KME40 = 7

Input and output file format.

KME50 = 9

Input and output file format.

KME60 = 10

MATLAB = 102

Output file format.

MDF = 101

Output file format.

MDF_4X = 107

Output file format.

MDF_4X_SIGNAL = 108

Output file format.

MDF_SIGNAL = 106

Output file format.

MEMO_LOG = 6

Input (internal device logfile format).

PLAIN_ASC = 5

Output file format.

RPCIII = 111

Output file format.

VECTOR_ASC = 3

Output file format.

VECTOR_BLF = 8

Output file format.

VECTOR_BLF_FD = 109

Input and output format.

XCP = 200

Output file format.

Property

class canlib.kvlclib.**Property**(*value*)

An enumeration.

ATTACHMENTS = 1003

Can attach files to converted data.

It is possible to use *Converter.attachFile()* to add a file. Used only with *WriterFormat.isPropertySupported()* and *ReaderFormat.isPropertySupported()*.

CALENDAR_TIME_STAMPS = 7

Write calendar time stamps.

CHANNEL_MASK = 5

Bitmask of the channels that should be used during conversion.

COMPRESSION_LEVEL = 32

ZLIB compression level for writers that use ZLIB for compression. [-1, 9].

CREATION_DATE = 27

File creation date/time as seconds in standard UNIX format. Used in file headers if not zero.

CROP_PRETRIGGER = 22

Crop pre-triggers.

DATA_IN_HEX = 12

Write data in hexadecimal format.

DECIMAL_CHAR = 10

Use token as decimal separator.

ENUM_VALUES = 23

Replace integer values in signals with strings from database.

FILL_BLANKS = 15

Propagate values down to next row in csv-files.

FIRST_TRIGGER = 2

Use first trigger as time reference.

FULLY_QUALIFIED_NAMES = 30

Write fully qualified signal names

HLP_J1939 = 6

Interpret events as J1939.

ID_IN_HEX = 11

Write id in hexadecimal format.

ISO8601_DECIMALS = 17

Number of time decimals (0-9) to print in the calendar timestamps using ISO8601.

LIMIT_DATA_BYTES = 26

Number of data bytes that a converter will write.

MERGE_LINES = 18

Merge two lines if their signal values are equal.

NAME_MANGLING = 14

Make signal names safe for use in Matlab.

NUMBER_OF_DATA_DECIMALS = 31

Number of data decimals (0-50)

NUMBER_OF_TIME_DECIMALS = 13

Number of time decimals (0-9).

OFFSET = 4

Time reference offset.

OVERWRITE = 28

Overwrite existing output files

RESAMPLE_COLUMN = 19

Only print a line when the given column has been accessed.

SAMPLE_AND_HOLD_TIMESTEP = 33

Time step in microseconds.

Used for format where time is implicit and defined by start time and the selected time step in microseconds. Signal values are interpolated with sample and hold. Used with e.g. DIAdem and RPCIII.

SEPARATOR_CHAR = 9

Use token as separator.

SHOW_COUNTER = 21

Add a counter to the output.

SHOW_SIGNAL_SELECT = 1002

Format requires a database.

Used only with *WriterFormat.isPropertySupported()* and *ReaderFormat.isPropertySupported()*.

SHOW_UNITS = 16

Show units on their own row.

SIGNAL_BASED = 1001

Writes signals and not data frames.

Used only with *WriterFormat.isPropertySupported()* and *ReaderFormat.isPropertySupported()*.

SIZE_LIMIT = 24

Maximum file size in megabytes before starting a new output file.

START_OF_MEASUREMENT = 1

Use start of measurement as time reference.

TIMEZONE = 29

Timezone for absolute timestamps

TIME_LIMIT = 25

Maximum delta time in seconds between first and last event before starting a new output file.

USE_OFFSET = 3

Use offset as time reference.

VERSION = 20

File format version.

WRITE_HEADER = 8

Write informational header.

Reader Formats

reader_formats()

canlib.kvlclib.**reader_formats**()

Return a generator of all reader formats.

You may list available Readers using:

```
>>> from canlib import kvlclib
>>> for format in kvlclib.reader_formats():
...     print(format)
KME24 (.kme): Reader, Kvaser binary format (KME 2.4)
KME25 (.kme25): Reader, Kvaser binary format (KME 2.5)
KME40 (.kme40): Reader, Kvaser binary format (KME 4.0)
KME50 (.kme50): Reader, Kvaser binary format (KME 5.0)
MDF (.log): Reader, CAN frames in Vector Mdf
MDF_4X (.mf4): Reader, CAN frames in MDF v4.1 for Vector CANalyzer
PLAIN_ASC (.txt): Reader, CAN frames in plain text format
VECTOR_ASC (.asc): Reader, CAN frames in Vector ASCII format
VECTOR_BLF_FD (.blf): Reader, CAN frames in Vector BLF format
CSV (.csv): Reader, CAN frames in CSV format
   ...
```

Note: CANlib before v5.37 incorrectly reported `.mke` as the file suffix for KME 2.4.

New in version 1.7.

ReaderFormat

class canlib.kvlclib.**ReaderFormat**(*id_*)

Helper class that encapsulates a Reader.

You may use *reader_formats()* to list available Readers.

New in version 1.7.

Changed in version 1.19: Updated formating in `__str__`.

getPropertyDefault(*rd_property*)

Get default value for property.

isPropertySupported(*rd_property*)

Check if specified read property is supported.

Retuns True if the property is supported by input format.

Parameters
rd_property (*Property*) – Reader property

Writer Formats

writer_formats()

canlib.kvlclib.**writer_formats**()

Return a generator of all writer formats.

You may list available Writers using:

```
>>> from canlib import kvlclib
>>> for format in kvlclib.writer_formats():
...     print(format)
CSV (.csv): Writer, CAN frames in CSV format
CSV_SIGNAL (.csv): Writer, Selected signals in CSV format
XCP (.csv): Writer, CCP/XCP calibration in CSV format
MATLAB (.mat): Writer, Selected signals in Matlab format for ATI Vision
KME24 (.kme): Writer, Kvaser binary format (KME 2.4) - used for Vector CANalyzer
KME25 (.kme25): Writer, Kvaser binary format (KME 2.5)
KME40 (.kme40): Writer, Kvaser binary format (KME 4.0)
KME50 (.kme50): Writer, Kvaser binary format (KME 5.0)
PLAIN_ASC (.txt): Writer, CAN frames in plain text format
   ...
```

New in version 1.7.

WriterFormat

class canlib.kvlclib.**WriterFormat**(*id_*)

Helper class that encapsulates a Writer.

You may use *writer_formats()* to list available Writers.

Changed in version 1.19: Updated formating in `__str__`.

classmethod getFirstWriterFormat()

Get the first supported output format.

classmethod getNextWriterFormat(*previous_id*)

Get the next supported output format.

getPropertyDefault(*wr_property*)

Get default value for property.

isPropertySupported(*wr_property*)

Check if specified write property is supported.

Retuns True if the property is supported by output format.

Parameters
wr_property (*Property*) – Writer property

Returns
bool

Miscellaneous

dllversion

canlib.kvlclib.**dllversion**()

Get the kvlclib version number as a *canlib.VersionNumber*

1.6.10 kvmlib

Exceptions

KvmError

exception canlib.kvmlib.**KvmError**

Base class for exceptions raised by the kvmlib dll

KvmDiskError

exception canlib.kvmlib.**KvmDiskError**

General disk error

status = -24

KvmDiskNotFormated

exception canlib.kvmlib.**KvmDiskNotFormated**

Disk not formatted

status = -34

KvmNoDisk

exception canlib.kvmlib.**KvmNoDisk**

No disk found

status = -13

KvmNoLogMsg

exception canlib.kvmlib.**KvmNoLogMsg**

No log message found

status = -10

LockedLogError

exception canlib.kvmlib.**LockedLogError**

Raised when trying to mount a log file to a locked log

Only one log file can be "mounted" internally at time. When a *LogFile* object requires its log file to be kept mounted for an extended period of time (such as when iterating over it) it will lock its containing *MountedLog* object. If during this time an attempt is made to mount a log file, this error will be raised.

Enumerations

Device

class canlib.kvmlib.**Device**(*value*)

kvmDEVICE_xxx

Device type, used to connect to a Memorator device.

MHYDRA = 0

Kvaser Memorator (2nd generation)

MHYDRA_EXT = 1

Kvaser Memorator (2nd generation) with extended data capabilities.

Error

class canlib.kvmlib.**Error**(*value*)

An enumeration.

CONFIG_ERROR = -41

Configuration error.

CRC_ERROR = -21

CRC check failed.

DEVICE_COMM_ERROR = -37

Device communication error.

DISKFULL_DATA = -26

Disk full (data).

DISKFULL_DIR = -25

Disk full (directory).

DISK_ERROR = -24

General disk error.

EOF = -12

End of file found.

FAIL = -1

Generic error.

FATAL_ERROR = -31

Fatal error.

FILE_ERROR = -23

File I/O error.

FILE_NOT_FOUND = -33

File not found.

FILE_SYSTEM_CORRUPT = -28

File system corrupt.

FIRMWARE = -40

Firmware error.

ILLEGAL_REQUEST = -32

Illegal request.

LOGFILEOPEN = -8

Can't find/open log file.

LOGFILEREAD = -14

Error while reading log file.

LOGFILEWRITE = -11

Error writing log file.

NOLOGMSG = -10

No log message found.

NOSTARTTIME = -9

Start time not found.

NOT_FORMATTED = -34

Disk not formatted.

NOT_IMPLEMENTED = -30

Not implemented.

NO_DISK = -13

No disk found.

OCCUPIED = -38

Device occupied.

OK = 0

OK!

PARAM = -3

Error in supplied parameters.

QUEUE_FULL = -20

Queue is full.

RESULT_TOO_BIG = -43

Result is too big for an out-parameter.

SECTOR_ERASED = -22

Sector unexpectadly erased.

SEQ_ERROR = -27

Unexpected sequence.

TIMEOUT = -36

Timeout.

UNSUPPORTED_VERSION = -29

Unsupported version.

USER_CANCEL = -39

User abort.

WRITE_PROT = -42

Disk is write protected.

WRONG_DISK_TYPE = -35

Wrong disk type.

FileType

class canlib.kvmlib.**FileType**(*value*)

kvmFILE_xxx

KME file type, a binary file format representing log data.

KME24 = 0

Deprecated format, use KME40

KME25 = 1

Deprecated format, use KME40

KME40 = 2

Kvaser binary format (KME 4.0)

KME50 = 3

Kvaser binary format (KME 5.0)

KME60 = 4

Kvaser binary format (KME 6.0) (Experimental)

LoggerDataFormat

class canlib.kvmlib.**LoggerDataFormat**(*value*)

kvmLDF_MAJOR_xxx

Logged data format (LDF) version.

MAJOR_CAN = 3

Used in Kvaser Memorator (2nd generation)

MAJOR_CAN64 = 5

Used in Kvaser Memorator (2nd generation) with extended data capabilities.

LogFileType

class canlib.kvmlib.enums.**LogFileType**(*value*)

kvmLogFileType_xxx

New in version 1.11.

ALL = 1

A log file containing all frames.

ERR = 0

A log file containing only error frames with frames before and after.

SoftwareInfoItem

class canlib.kvmlib.enums.**SoftwareInfoItem**(*value*)

kvm_SWINFO_xxx

Different types of version information that can be extracted using kvmDeviceGetSoftwareInfo(), used internally by *Memorator*.

CONFIG_VERSION_NEEDED = 5

Returns the version of the binary format the device requires (param.lif).

CPLD_VERSION = 6

Obsolete.

DRIVER = 2

Returns the used driver version information.

DRIVER_PRODUCT = 4

Obsolete. Returns the product version information.

FIRMWARE = 3

Returns the device firmware version information.

KVMLIB = 1

Returns the version of kvmlib.

Events

MessageEvent

class canlib.kvmlib.**MessageEvent**(*id=None*, *channel=None*, *dlc=None*, *flags=None*, *data=None*, *timestamp=None*)

Bases: *LogEvent*

A CAN message recorded by a Memorator

asframe()

Convert this event to a *canlib.Frame*

Creates a new *canlib.Frame* object with the same contents as this event.

LogEvent

class canlib.kvmlib.**LogEvent**(*timestamp=None*)

The base class for events recorded by a Memorator.

RTCEvent

class canlib.kvmlib.**RTCEvent**(*calendartime=None*, *timestamp=None*)

Bases: *LogEvent*

An real-time clock message recorded by a Memorator

TriggerEvent

class canlib.kvmlib.**TriggerEvent**(*type=None*, *timestamp=None*, *pretrigger=None*, *posttrigger=None*, *trigno=None*)

Bases: *LogEvent*

A trigger message recorded by a Memorator

VersionEvent

class canlib.kvmlib.**VersionEvent**(*lioMajor*, *lioMinor*, *fwMajor*, *fwMinor*, *fwBuild*, *serialNumber*, *eanHi*, *eanLo*)

Bases: *LogEvent*

A version message recorded by a Memorator

memoLogEventEx

class canlib.kvmlib.**memoLogEventEx**

Low level c type class holding a log event.

MEMOLOG_TYPE_CLOCK = 1

The event type used in kvmLogRtcClockEx

MEMOLOG_TYPE_INVALID = 0

Invalid MEMOLOG event type

MEMOLOG_TYPE_MSG = 2

The event type used in kvmLogMsgEx

MEMOLOG_TYPE_TRIGGER = 3

The event type used in kvmLogTriggerEx

MEMOLOG_TYPE_VERSION = 4

The event type used in kvmLogVersionEx

createMemoEvent()

Convert event to *LogEvent*.

event

Structure/Union member

Log files

UnmountedLog

class canlib.kvmlib.**UnmountedLog**(*memorator*)

The log area of a Memorator before mounting

Before the log area of a *Memorator* object has been mounted, its `log` attribute is set to an instance of this class.

This class has all the functionality available even when the log area has not been mounted – this functionality is still present after the log area has been mounted (or if the area is always mounted – see *Kmf*).

The number of log files can be read as the `len()` of this object (`container` is either a *Memorator* or *Kmf* object):

```
num_log_files = len(container.log)
```

All log files can also be deleted:

```
container.log.delete_all()
```

New in version 1.6.

delete_all()

Delete all log files

MountedLog

class canlib.kvmlib.**MountedLog**(*memorator*, *ldf_version*)

Bases: *UnmountedLog*

The log area of a Memorator once mounted

Once a *Memorator* object has been mounted, its `log` attribute is set to an instance of this class. This is the preferred way of using this class.

For *Kmf* objects, the `log` attribute is always an instance of this class as they are by definition mounted.

In the following examples `container` can be either a *Memorator* object that has been mounted, or a *Kmf* object.

The files within the log can be accessed via indexing:

```
container.log[index]
```

or all files can be iterated over:

```
for log_file in container.log:
    ...
```

The log area can also be validated:

```
container.log.validate()
```

Also see the super class *UnmountedLog* for functionality this class has inherited.

New in version 1.6.

validate()

Raises the corresponding exception if any errors are detected

LogFile

class canlib.kvmlib.**LogFile**(*container*, *index*)

A log file read from a *MountedLog* object

This class is normally not directly instantiated but retrieved from a *MountedLog* object.

The most common use of this class is iterating through it to get the individual events as *LogEvent* subclasses:

```
for event in logfile:
    ...
```

Note: While iterating over a *LogFile*, accessing any other *LogFile* is will result in a *LockedLogError*. Make sure to finish the loop (or when using iteration objects directly call the `close` method) before interacting with any other log files.

A fast approximation of the number of events is given by *event_count_estimation*, the exact number of events can be calculated using:

```
num_events = len(list(logfile))
```

Finally this class has several read-only properties for getting information about the log file itself.

Note: Before any data is fetched from the dll, this class will make sure that the correct file has been mounted on the underlying `kvmHandle`.

Manually mounting or unmounting log files by calling the dll directly is not supported.

New in version 1.6.

property creator_serial

The serial number of the interface that created the log file

Type
`int`

property end_time

The time of the last event in the log file

Type
`datetime.datetime`

event_count_estimation()

Returns an approximation of the number of events

The number is a rough estimation because it is calculated from the number of blocks allocated by the log file on the disk as an optimization.

New in version 1.7.

property log_type

The type of the log file

New in version 1.11.

Type
LogFileType

property start_time

The time of the first event in the log file

Type
`datetime.datetime`

kme files

createKme()

canlib.kvmlib.**createKme**(*path*, *filetype=FileType.KME50*)

Create a KME file on the host computer

Parameters

- **path** (`str`) – The full filepath for the .KME file (e.g. `"data/mylog.kme50"`).

- **filetype** (*FileType*) – The KME file type

Returns
Kme

New in version 1.7.

openKme()

canlib.kvmlib.**openKme**(*path*, *filetype=FileType.KME50*)

Open a KME file on the host computer

Parameters

- **path** (str) – The full filepath for the .KME file (e.g. "data/mylog.kme50").
- **filetype** (*FileType*) – The KME file type

Returns
Kme

New in version 1.7.

Kme

class canlib.kvmlib.**Kme**(*handle*)

A kme file

A class representing a KME file. The main use is twofold:

Either we create a KME file using *createKme* and would like to write events using *Kme.write_event*:

```
with kvmlib.createKme('out/data.kme50') as kme:
    ...
    kme.write_event(my_logevent)
```

Or we read events from an existing KME file:

```
with kvmlib.openKme('in/data.kme50') as kme:
    event = kme.read_event()
    ...
    print(event)
```

Note that only KME files of type KME50 and KME60 may currently be written to.

New in version 1.7.

Changed in version 1.20: Added experimental support for KME60.

close()

estimate_events()

Estimate how many events the KME file contains

Returns:
int: Approximate number of events in KME file.

New in version 1.7.

Deprecated since version 1.11.

event_count_estimation()

Returns an approximation of the number of events contained in the KME file.

Returns:
int: Approximate number of events in KME file.

New in version 1.11.

events()

read_event()

Read logevent from KME file

Returns:
memoLogEventEx

New in version 1.7.

write_event(*logevent*)

kme_file_type()

canlib.kvmlib.**kme_file_type**(*path*)

Scan KME file and report version

Open and read the file `path` and try to decode what version of KME it contains.

Parameters
path (`str`) – The full filepath for the .KME file (e.g. `"data/mylog.kme"`).

Returns
FileType – The KME file type

New in version 1.7.

kmf files

openKmf()

canlib.kvmlib.**openKmf**(*path*, *device_type=Device.MHYDRA_EXT*)

Open a kmf file from disk

Parameters

- **path** (`str`) – The filepath to the .KMF file (e.g. `"data/kmf/LOG00000.KMF"`).
- **device_type** (*Device*) – The type of the memorator that created the .KMF file(s) (defaults to *Device.MHYDRA_EXT*)

Returns
Kmf

New in version 1.6.

Kmf

class canlib.kvmlib.**Kmf**(*handle*, *ldf_version*)

A kmf file opened with *openKmf*

The main use of this class is using its `log` attribute, which is a *MountedLog* object (see its documentation for how to use it).

Also see the base class *KmfSystem* for inherited functionality.

Variables
log (*MountedLog*) – Object representing the log of log files within the kmf container-file.

New in version 1.6.

KmfSystem

class canlib.kvmlib.**KmfSystem**(*handle*)

The base class of *Kmf* and *Memorator*

The *Kmf* and *Memorator* classes are very similar, they are different ways of reading log files (*LogFile*) created by a memorator. This class represents the common ground between all ways of accessing log files.

All subclasses should have a `log` attribute which is an *UnmountedLog* or subclass thereof.

This class automatically closes its internal handle when garbage collected.

New in version 1.6.

class DiskUsage(*used*, *total*)

property total
Alias for field number 1

property used
Alias for field number 0

close()
Close the internal handle

> **Warning:** Closing invalidates the object.

property disk_usage
The disk usage

Returns

`namedtuple` – containing

- `used` (`int`): Used disk space in megabytes.
- `total` (`int`): Total disk space in megabytes.

Type
`namedtuple`

Memorator

openDevice()

canlib.kvmlib.**openDevice**(*channel_number*, *mount=False*, *device_type=Device.MHYDRA_EXT*)

Open a Memorator device

Parameters

- **channel_number** (int) – A channel number of the Memorator to be opened.
- **mount** (bool) – Whether the memorator log area should be mounted before returned.
- **device_type** (*Device*) – The type of the memorator to be opened (defaults to *Device.MHYDRA_EXT*)

Returns

Memorator

New in version 1.6.

Memorator

class canlib.kvmlib.**Memorator**(*handle*, *channel_number*, *device_type*)

A Memorator device opened with *openDevice*

This class should not be instantiated directly, instead call *openDevice*.

A device opened as *Memorator* can be configured from XML using *kvamemolibxml.load_xml_file* and *write_config*:

```
# Read the original XML file (config.xml)
config = kvamemolibxml.load_xml_file("config.xml")

# Validate the XML
errors, warnings = config.validate()
if errors or warnings:
    print(errors)
    print(warnings)
    raise Exception("One or more errors/warnings in xml configuration")

# Write the configuration in binary
memorator.write_config(config.lif)
```

The configuration can then be read back (in binary):

```
dev.read_config()
```

The log files on the device can be accessed via the log attribute. By default, the log area is not mounted so only a few operations are allowed, such as getting the number of log files:

```
num_log_files = len(memorator.log)
```

For a full list of allowed operations, see *UnmountedLog* (the type of .log before a mount).

The log area can be mounted either with *openDevice*'s *mount* argument set to True, or later with the *Memorator.mount* function. Once this is done the log attribute is a *MountedLog* which supports getting log files as *LogFile* objects:

```
# We can index the Memorator object if we know what file we want
log_file_number_two = memorator.log[2]

# Although usually we want to loop through all log files
for log_file in memorator.log:
    ...
```

See the documentation of *MountedLog* for all available operations.

Parameters

- **channel_number** (int) – The channel number that was used to connect to this memorator.
- **device_type** (*Device*) – The device type that was used to connect to this memorator.
- **mounted** (bool) – Whether the device's memory card has been mounted.

New in version 1.6.

property config_version_needed

The version of param.lif that the connected device expects

Type
canlib.VersionNumber

property disk_size

The disk size in megabytes

Warning: This is not necessarily the amount of space available for allocation; `Memorator.format_disk(reserved_space=Memorator.disk_size)` is not guaranteed to succeed.

The most reliable way of calculating reserved space is to first format the disk with `reserved_space` set to `0`, and then use `Memorator.disk_usage.total`.

Type
int

property driver_version

The used driver version information

Type
canlib.VersionNumber

property firmware_version

The device firmware version information

Type
canlib.VersionNumber

flash_leds()

Flash all LEDs on the Memorator

format_disk(*reserved_space=10*, *database_space=2*, *fat32=True*)

Format the SD memory card in the Memorator

Parameters

- **reserved_space** (int) – Space to reserve for user files, in MB.

- **database_space** (int) – Space to reserve for database files, in MB.
- **fat32** (bool) – Whether the filesystem should be formatted as fat32 (defaults to True)

Changed in version 1.9: Will now reopen the internal handle if the log is mounted in order to refresh Memorator.log.ldf_version

property kvmlib_version

Returns the version of kvmlib

Type
canlib.VersionNumber

mount()

Mount the Memorator's log area

This replaces the object log attribute with a *MountedLog*, which allows access to log files.

If the log has already been mounted (self.mounted == True), this is a no-op.

mounted = None

read_config()

Read the configuration of the Memorator

The configuration is returned as a bytes object with the binary configuration data (param.lif).

If a *kvamemolibxml.Configuration* is desired, the returned bytes can be parsed using *kvamemolibxml.load_lif*:

```
config_object = kvamemolibxml.load_lif(memorator.read_config())
```

reopen(*device_type=None*, *mount=False*)

Closes and then reopens the internal handle

property rtc

The value of the real-time clock

Type
datetime.datetime

property serial_number

The serial number of the Memorator

Type
int

write_config(*config_lif*)

Writes configuration to the Memorator

The configuration should be given as a bytes object with the binary configuration data (param.lif).

Given a *kvamemolibxml.Configuration* object, pass its lif attribute to this function:

```
memorator.write_config(config_object.lif)
```

Miscellaneous

dllversion()

canlib.kvmlib.**dllversion**()

Get the kvmlib version number as a *canlib.VersionNumber*

1.6.11 kvrlib

Exceptions

KvrError

exception canlib.kvrlib.**KvrError**

Bases: *DllException*

KvrGeneralError

exception canlib.kvrlib.exceptions.**KvrGeneralError**(*status*)

Bases: *KvrError*

A kvrlib error that does not (yet) have its own Exception

> **Warning:** Do not explicitly catch this error, instead catch *KvrError*. Your error may at any point in the future get its own exception class, and so will no longer be of this type. The actual status code that raised any *KvrError* can always be accessed through a `status` attribute.

KvrBlank

exception canlib.kvrlib.exceptions.**KvrBlank**

Bases: *KvrError*

List was not set or no more results.

status = -6

KvrNoAnswer

exception canlib.kvrlib.exceptions.**KvrNoAnswer**

Bases: *KvrError*

No answer arrived within given timeout.

status = -8

KvrParameterError

exception canlib.kvrlib.exceptions.**KvrParameterError**

Bases: *KvrError*

Error in supplied in parameters.

status = -4

KvrPasswordError

exception canlib.kvrlib.exceptions.**KvrPasswordError**

Bases: *KvrError*

Supplied password was wrong.

status = -5

KvrUnreachable

exception canlib.kvrlib.exceptions.**KvrUnreachable**

Bases: *KvrError*

Remote device is unreachable.

status = -7

DeviceNotInSetError

exception canlib.kvrlib.**DeviceNotInSetError**

Bases: KeyError, *CanlibException*

Discovery

Address

class canlib.kvrlib.**Address**(*type*, *address*)

An IP or MAC address

Parameters

- **type** (*AddressType*) – Address type.
- **address** (list[int]) – A list of integers, the numbers in the address.

classmethod from_c(*c_addr*)

Create an *Address* object from a *kvrAddress* object

to_c()

Create a *kvrAddress* from this object

DeviceInfo

class canlib.kvrlib.**DeviceInfo**(*device_info=None*)

Information about a device that can be written to the registry

See *DeviceInfoSet* for information about how to get *DeviceInfo* objects, process them, and then write them to the registry.

property accessibility

The accessibility of this device

Type
Accessibility

property availability

The availability of this device

Type
Availability

property base_station_id

Address of the base station

Type
Address

property client_address

Address of connected client

Type
Address

property connect

Whether the service should connect to this device

Type
bool

property device_address

Address of remote device

Type
Address

property ean

EAN of device

Type
EAN

property encryption_key

The encryption key to use when encrypting communication

Note: This attribute is write-only.

Type
bytes

property firmware_version

Firmware version

Type
VersionNumber

property hostname

DNS hostname if available, otherwise an empty string

Type
str

info()

Create a string with information about the *DeviceInfo*

To be used when the `str()` representation is not detailed enough.

property name

User-defined name of device

Type
str

property password

The accessibility password to use when connecting to a device

Note: This attribute is write-only.

Type
str

property serial

The serial number of the device

Type
int

property service_status

A tuple with the service status of the device

The returned tuple has the format `(state, start_info, text)`, where `state` is a *ServiceState*, `start_info` is a *StartInfo*, and `text` is a `str` with local connection status.

Type
ServiceStatus

property stored

Whether this *DeviceInfo* was read from the registry

Type
bool

update(*other*)

Update the attributes of this instance

This function takes a `dict`, and will set the attributes given by the keys to the corresponding value (on this object).

property usage

Usage status (Free/Remote/USB/Config)

Type
DeviceUsage

Discovery

class canlib.kvrlib.**Discovery**(*handle*)

A kvrlib "Discovery" process

Most of the time the discovery process can be handled by *discover_info_set*, which returns the results of the discovery as a *DeviceInfoSet*.

Even when interacted with directly, instnances of this class are normally not instantiated directly, but created using *start_discovery*, or sometimes using *openDiscovery*.

Instances of this class can be used as context handlers, in which case the discovery will be closed when the context is exited. The discovery will also be closed when the instance is garbage collected, if it hasn't already.

property addresses

The list of addresses to use for discovery

Note: This attribute is write-only.

This attribute should be a list of *Address* objects.

If the *Discovery* object was created by *start_discovery*, the addresses are automatically set. Otherwise, they must be assigned before *Discovery.start* can be called.

close()

Close the discovery process

This function is called when the *Discovery* object is garbage-collected.

If the *Discovery* object is used as a context handler, this function will be called when the context exits.

results()

Return an iterator of the result from the discovery

The results are yielded as *DeviceInfo* objects.

start(*delay*, *timeout*, *report_stored=True*)

Run the discovery

If the *Discovery* object was created by *start_discovery*, the discovery has already been run, and this function does not need to be called.

After the discovery has been run, its results can be retrieved using *Discovery.results*.

Parameters

- **delay** (int) – The millisecond delay between sending discovery messages to addresses in the address list.
- **timeout** (int) – The millisecond timeout after all discovery messages have been sent, before returning.
- **report_stored** (bool) – if True (the default), stored devices will be discovered.

get_default_discovery_addresses()

canlib.kvrlib.**get_default_discovery_addresses**(*broadcast=True*, *stored=False*)

Retrieve the default discovery addresses

Parameters

- **broadcast** (bool) – If True (the default), then the returned list will contain broadcast addresses.
- **stored** (bool) – If True (defaults to False), then the returned list will contain earlier stored addresses.

If both arguments are False, a ValueError will be raised.

Retruns a list of *Address* objects.

openDiscovery()

canlib.kvrlib.**openDiscovery**()

Creates and returns a *Discovery* object

Device discovery is normally done using *discover_info_set*.

ServiceStatus

class canlib.kvrlib.**ServiceStatus**(*state*, *start_info*, *text*)

property start_info

Alias for field number 1

property state

Alias for field number 0

property text

Alias for field number 2

set_clear_stored_devices_on_exit()

canlib.kvrlib.**set_clear_stored_devices_on_exit**(*val*)

Sets whether kvrlib should clear stored devices when the application exist

start_discovery()

canlib.kvrlib.**start_discovery**(*delay*, *timeout*, *addresses=None*, *report_stored=True*)

Start and return a *Discovery*

Device discovery is normally done using *discover_info_set*.

The returned object should usually be used as a context handler:

```
with kvrlib.start_discovery(delay=100, timeout=1000) as disc:
    for result in disc.results():
        # process the discovery result
        print(result)
```

store_devices()

canlib.kvrlib.**store_devices**(*devices*)

Store a collection of *DeviceInfo* objects in the registry

See *DeviceInfoSet* for a simpler way of dealing with device infos and the registry.

Note: Previously stored devices are cleared.

stored_devices()

canlib.kvrlib.**stored_devices**()

Read the devices stored in the registry as a tuple of *DeviceInfo* objects

Enumerations

Accessibility

class canlib.kvrlib.**Accessibility**(*value*)

kvrAccessibility_xxx

Remote device accessability status.

PRIVATE = 3

Private (invisible, password needed to connect).

PROTECTED = 2

Protected (visible for all, password needed to connect).

PUBLIC = 1

Public (visible for all, no password required to connect).

UNKNOWN = 0

Unknown (e.g., no reply from device).

AddressType

class canlib.kvrlib.**AddressType**(*value*)

kvrAddressType_xxx

Type of device address.

Note: Ports are currently not used.

IPV4 = 1

IP v.4 address.

IPV4_PORT = 3

IP v.4 address with tcp-port.

IPV6 = 2

IP v.6 address.

MAC = 4

Ethernet MAC address.

UNKNOWN = 0

Unknown (e.g., no reply from device).

AddressTypeFlag

class canlib.kvrlib.**AddressTypeFlag**(*value*)

kvrAddressTypeFlag_xxx

Flags for setting what addresses that should be returned by *get_default_discovery_addresses*.

ALL = 255

All defined below

BROADCAST = 1

Broadcast addresses

STORED = 2

Previously stored addresses

Availability

class canlib.kvrlib.**Availability**(*value*)

kvrAvailability_xxx

Device avalability flags.

FOUND_BY_SCAN = 1

Device was found by scan.

NONE = 0

Manually added.

STORED = 2

Device was stored.

BasicServiceSet

class canlib.kvrlib.**BasicServiceSet**(*value*)

kvrBss_xxx

Basic Service Set.

ANY = 2

Any.

INDEPENDENT = 1

Ad-hoc network.

INFRASTRUCTURE = 0

Network with AP.

ConfigMode

class canlib.kvrlib.**ConfigMode**(*value*)

kvrConfigMode_xxx

Configuration mode.

ERASE = 2

Erase and write.

R = 0

Read only.

RW = 1

Read/write.

DeviceUsage

class canlib.kvrlib.**DeviceUsage**(*value*)

kvrDeviceUsage_xxx

Remote device usage status.

CONFIG = 4

Device is being configured via USB.

FREE = 1

Not in use.

REMOTE = 2

Connected to a PC (as a remote device).

UNKNOWN = 0

Unknown (e.g., no reply from device).

USB = 3

Connected via USB cable.

Error

class canlib.kvrlib.**Error**(*value*)

An enumeration.

BLANK = -6

List was not set or no more results.

BUFFER_TOO_SMALL = -15

The buffer provided was not large enough to contain the requested data.

CHECKSUM = -3

Checksum problem.

DUPLICATED_DEVICE = -13

There are duplicates in the device list.

GENERIC = -2

Generic error.

NOT_IMPLEMENTED = -9

Function is not yet implemented.

NOT_INITIALIZED = -1

kvrlib has not been initialized.

NO_ANSWER = -8

No answer arrived within given timeout.

NO_DEVICE = -7

Remote device is unreachable.

NO_SERVICE = -12

The helper service is not running.

OK = 0

OK!

OUT_OF_SPACE = -11

Out of space, eg. to many open handles, to small buffer.

PARAMETER = -4

Error in supplied in parameters.

PASSWORD = -5

Supplied password was wrong.

PERMISSION_DENIED = -10

Permission denied.

XML_VALIDATION = -14

XML-file validation failed.

NetworkState

class canlib.kvrlib.**NetworkState**(*value*)

kvrNetworkState_xxx

States for network connection.

AUTHENTICATING = 8

EAPOL handshake ongoing.

AUTHENTICATION_FAILED = 9

Authentication have failed.

CONNECTED = 7

Network is reached.

CONNECTING = 6

Waiting for connections (ad-hoc).

CONNECTION_DELAY = 5

Delay during connection (ad-hoc).

FAILED_MIC = 11

MIC verification (EAPOL-key) failed.

INITIALIZING = 3

Started, waiting for initialization.

INVALID = 1

Network hardware has been disabled.

NOT_CONNECTED = 4

No connection (may auto-connect).

ONLINE = 10

Authentication completed.

STARTUP = 2

Configuring network hardware.

UNKNOWN = 0

Bad state, should never be reported.

RegulatoryDomain

class canlib.kvrlib.**RegulatoryDomain**(*value*)

kvrRegulatoryDomain_xxx

Regulatory domain.

CHINA_MII = 4

MII

EUROPE_ETSI = 1

ETSI

JAPAN_TELEC = 0
TELEC

NORTH_AMERICA_FCC = 2
FCC

WORLD = 3
WORLD

RemoteState

class canlib.kvrlib.**RemoteState**(*value*)
kvrRemoteState_xxx

State of connection to device.

AVAILABLE = 1
Tries to ping known device.

CLOSING = 10
Will stop communication.

CONFIG_CHANGED = 13
Same as UNWILLING.

CONNECTION_DOWN = 5
Will try and restore connection.

CONNECTION_UP = 6
Device connected, heartbeat up.

DISCOVERED = 2
Currently not used.

INSTALLING = 15
Driver installation is in progress.

REDISCOVER = 7
Trying to talk to device.

REDISCOVER_PENDING = 9
Will do rediscover in a moment.

REMOVE_ME = 11
Device removed, it will be stopped.

STANDBY = 12
Known device, but unused.

STARTED = 4
Currently not used.

STARTING = 3
Initializes for new device.

STOPPING = 14
Tries to stop device.

UNWILLING = 8

Device turned down connection req.

VOID = 0

Marked as not in list.

ServiceState

class canlib.kvrlib.**ServiceState**(*value*)

kvrServiceState_xxx

Current service state.

AVAILABLE = 1

Device available

CLOSING = 10

Closing device

CONFIG_CHANGED = 13

Configuration has changed

CONNECTION_DOWN = 5

Connection is currently down

CONNECTION_UP = 6

Connection is corrently up. The device should be showing up in Kvaser Hardware and be ready to be used from CANlib.

DISCOVERED = 2

Device discovered

INSTALLING = 15

Device is currently being installed

REDISCOVER = 7

We've lost the device - rediscover it

REDISCOVER_PENDING = 9

Rediscover is pending

REMOVE_ME = 11

Removing the device

STANDBY = 12

Standby, the service is not taking any actions against this device

STARTED = 4

Device is started

STARTING = 3

Device is starting, other devices may inhibit this device from being started at the moment (e.g. by installing).

STOPPING = 14

Stopping devic

UNWILLING = 8

Unwilling, see sub state for reason

VOID = 0

Void

StartInfo

class canlib.kvrlib.**StartInfo**(*value*)

kvrStartInfo_xxx

Current start information.

ERR_CONFIGURING = 5

I'm being configured so won't start

ERR_ENCRYPTION_PWD = 7

Wrong encryption password.

ERR_IN_USE = 2

Already connected to someone else

ERR_NOTME = 4

This start is not for me

ERR_PARAM = 6

Invalid parameters in QRV (non matching versions)

ERR_PWD = 3

Wrong connection pwd

NONE = 0

No information available

START_OK = 1

Started OK

Device Info Set

discover_info_set()

canlib.kvrlib.**discover_info_set**(*delay=100*, *timeout=1000*, *addresses=None*, *report_stored=True*)

empty_info_set()

canlib.kvrlib.**empty_info_set**()

stored_info_set()

canlib.kvrlib.**stored_info_set**()

DeviceInfoSet

class canlib.kvrlib.**DeviceInfoSet**(*iterable=None*)

A mutable set of *DeviceInfo* objects that can be written to the registry

There are three different functions for creating *DeviceInfoSet* objects:

- *empty_info_set*: Creates an empty set.
- *stored_info_set*: Creates a set from the device information stored in the registry.
- *discover_info_set*: Create a set from the results of a *Discovery*.

Once a *DeviceInfoSet* has been created it can be modified as a normal set, and the *DeviceInfo* elements can also be modified. Once all modification is done, the set can be written to the registry with *DeviceInfoSet.store*.

The main difference between *DeviceInfoSet* and normal sets is that it can only contain one *DeviceInfo* with a specific combination of EAN and serial number, even if they otherwise are not equal. This means that even if `info in infoset` evaluates to true, that exact object may not be in the set, and modifying it may not change the set.

To retrieve a specific *DeviceInfo* from the set use *DeviceInfoSet.find*:

```
info = infoset.find(ean='01234-5', serial=42)
```

Modifying the resulting *DeviceInfo* will then change the contents of the set.

Instances of this class can also be used as context managers, in which case they will write their content to the registry when the context exists.

add(*info*)

Add a *DeviceInfo* to this *DeviceInfoSet*

Parameters
info (*DeviceInfo*) – The element to add to this set

If the set already contains a *DeviceInfo* with the same EAN and serial number as `info`, the previous *DeviceInfo* will be discarded and replaced by `info`.

discard(*info*)

Remove an element. Do not raise an exception if absent.

find(*ean*, *serial*)

Find and return a specific *DeviceInfo* in this set

Parameters

- **ean** (*EAN*) – The EAN to search for
- **serial** (`int`) – The serial number to search for

If no *DeviceInfo* with the EAN and serial number is found in this set, *DeviceNotInSetError* is raised.

Note that there can never be more than one *DeviceInfo* with the same EAN and serial number in a *DeviceInfoSet*.

find_remove(*ean*, *serial*)

Find a specific *DeviceInfo* and remove it from this set

Like *DeviceInfoSet.find* but immediately removes the *DeviceInfo* found from the set.

Parameters

- **ean** (*EAN*) – The EAN to search for
- **serial** (int) – The serial number to search for

has(*ean*, *serial*)

Check whether the set contains a specific *DeviceInfo*

Similar to *DeviceInfoSet.find* but instead of returning a *DeviceInfo* or raising an exception, this function returns True or False.

Parameters

- **ean** (*EAN*) – The EAN to search for
- **serial** (int) – The serial number to search for

new_info(*ean*, *serial*, ***attrs*)

Create and return new *DeviceInfo* in this set

Any attribute of the *DeviceInfo* that should have a specific value can be passed as keyword arguments to this function.

The EAN and serial number must be provided.

Parameters

- **ean** (*EAN*) – The EAN of the info (*DeviceInfo.ean*)
- **serial** (int) – The serial number of the info (*DeviceInfo.serial*)
- **attrs** – Any other attributes to be set on the *DeviceInfo*

If the set already contains a *DeviceInfo* with the EAN ean and serial number serial, the previous *DeviceInfo* will be discarded and replaced by the new *DeviceInfo* created by this function.

store()

Store this set's *DeviceInfo* objects in the registry

update(**others*)

Update the set, adding elements from all others

All others must contain nothing but *DeviceInfo* objects, or this function will raise TypeError without modifying this *DeviceInfoSet*.

Remote Device

AddressInfo

class canlib.kvrlib.**AddressInfo**(*address1*, *address2*, *netmask*, *gateway*, *is_dhcp*)

property address1

Alias for field number 0

property address2

Alias for field number 1

property gateway

Alias for field number 3

property is_dhcp

Alias for field number 4

property netmask

Alias for field number 2

ConfigProfile

class canlib.kvrlib.**ConfigProfile**(*device*, *profile_number*)

A configuration profile of a remote-capable device

The active profile for a *RemoteDevice*, `rdev`, is accessed with:

```
profile = rdev.active_profile
```

Other profiles are accessed with `profiles`:

```
first = rdev.profiles[0]
```

See the documentation for *RemoteDevice* for more information on how to get *ConfigProfile* objects.

The specific configuration of a profile can be read:

```
xml_string = profile.read()
```

And it can also be written back:

```
profile.write(xml_string)
```

The password used by this object is taken from its parent *RemoteDevice* object. See that documentation for how to set the password.

XML_BUFFER_SIZE = 2046

property channel_number

The CANlib channel number used to communicate with the device

Type
`int`

clear()

Clear the configuration

This will also clear any previously set device password.

Note: This method requires the parent *RemoteDevice* to have the correct password.

property info

A simplified version of the configuration

It is not necessary to know the configuration password to access this information. Note that the partial XML data returned is not enough to reconfigure a device.

Type
str

property password

The password used to communicate with the device

Type
str

read()

Read the configuration

Returns either an XML string, or `None` if there is no configuration.

Note that *ConfigProfile.write* can handle `None`; in other words:

```
xml = profile.read()

# Do anything, including writing new configurations
...

profile.write(xml)
```

Will always work even if `xml` is `None`. This would mean that the profile originally had an empty configuration, and it will once again have an empty configuration at the end.

Note: This method requires the parent *RemoteDevice* to have the correct password.

write(*xml*)

Write the configuration area

This function takes as its single argument either an xml string that will be written to this profile, or `None` in which case the configuration will be cleared.

Note: This method requires the parent *RemoteDevice* to have the correct password.

ConnectionStatus

class canlib.kvrlib.**ConnectionStatus**(*state*, *tx_rate*, *rx_rate*, *channel_number*, *rssi*, *tx_power*)

property channel_number

Alias for field number 3

property rssi

Alias for field number 4

property rx_rate

Alias for field number 2

property state

Alias for field number 0

property tx_power

Alias for field number 5

property tx_rate
Alias for field number 1

ConnectionTestResult

class canlib.kvrlib.**ConnectionTestResult**(*rssi*, *rtt*)

property rssi
Alias for field number 0

property rtt
Alias for field number 1

ConnectionTest

class canlib.kvrlib.**ConnectionTest**(*config_handle*)
A connection test for a specific device

A connection test for a *RemoteDevice*, `rdev`, is created by:

```
test = rdev.connection_test()
```

While a connection test is running, the device will connect and start pinging itself to measure round-trip time (RTT) and Receive Signal Strength Indicator (RSSI).

A *ConnectionTest* is started with *ConnectionTest.start* and stopped with *ConnectionTest.stop*, after which its results are retrieved by *ConnectionTest.results*. If it is acceptable to block while the test is running, these three calls can be combined into *ConnectionTest.run*:

```
results = test.run(duration=10)
print(results)
```

Connection tests are automatically closed when garbage-collected, but they can also be closed manually with *ConnectionTest.close*.

close()
Close the internal handle

results(*maxresults=20*)
Get the results from a connection test

The test must have been running for any results to be available.

This function returns a *ConnectionTestResult*, which is a namedtuple of `(rssi, rtt)`. Both `rssi` and `rtt` are tuples containing a number of individual results for RSSI and RTT.

Parameters
maxresults (`int`) – The maximum number of rssi and rtt numbers to return. The returned `rssi` and `rtt` will be tuples with this long or the number of results available long, whichever is shortest.

run(*duration*, *maxresults=20*)
Run a connection test and return its results.

This function calls *ConnectionTest.start*, then blocks for `duration` seconds, then calls *ConnectionTest.stop* before finally returning the *ConnectionTestResult* from *ConnectionTest.results*.

Parameters

- **duration** (int) – Seconds to run the test for.
- **maxresults** (int) – Passed to *ConnectionTest.results* as maxlen.

start()

Start the connection test

stop()

Stop the connection test

openDevice()

canlib.kvrlib.**openDevice**(*channel_number*, *password=''*, *validate_password=False*)

Create a *RemoteDevice* object bound to a channel number

Parameters

- **channel_number** (int) – CANlib channel number of the device to open
- **password** (str) – Password of the device, if any
- **validate_password** (bool) – Whether the password should be validated (defaults to False).

This function checks that a remote-capable device is currently connection on the channel channel_number. If validate_password is True, it also checks that the password supplied is correct. If any of these checks fail, a ValueError will be raised.

RemoteDevice

class canlib.kvrlib.**RemoteDevice**(*channel_number*, *password*)

A remote-capable Kvaser device

This class is normally instantiated with *openDevice*:

```
rdev = kvrlib.openDevice(CHANNEL_NUMBER)
```

Once this is done, the currently active profile can be accessed:

```
active = rdev.active_profile
```

All profiles, including the active one, are *ConfigProfile* objects, see that documentation for all the operations available for profile objects.

The full list of profiles a device has can be inspected using rdev.profiles. This is a *RemoteDevice.ProfileList* object, which works much like a list:

```
# The profile list can be indexed
first = rdev.profiles[0]

# We can check if a configuration belongs to this device with 'in'
assert first in rdev.profiles

# The length of the profile list is the number of profiles
num_profiles = len(rdev.profiles)
```

(continues on next page)

(continued from previous page)

```
# Using the number of profiles, we can get the last one
last = rdev.profiles[num_profiles - 1]

# But negative indexes also work and are a better way of getting
# the last profile
last = rdev.profiles[-1]

# You can also iterate over all profiles
for profile in rdev.profiles:
    ...
```

RemoteDevice also lets you access a variety of information about the specific device, as well as the ability to do a WLAN scan with *RemoteDevice.wlan_scan*.

If the device is password protected, that password can be passed to *openDevice*:

```
protected_device = kvrlib.openDevice(0, password="Once upon a playground rainy")
```

After the object is created, the password is available as:

```
password = protected_device.password
```

The password can be changed with:

```
protected_device.password = "Wolves without teeth"
```

The reason the password is stored as clear-text is because it must be supplied to the underlying library each time an operation is done using this and related classes. This also means that the password is only validated, and required, when one of the functions requiring a password is called.

If the device is not password-protected, the password should be an empty string:

```
unprotected_device = kvrlib.openDevice(1)
assert unprotected_device.password == ''
```

class ProfileList(*channel_number*)

The available profiles in a remote-capable device

This is the type of `RemoteDevice.profiles`. It implements the following:

- `len(profiles)`
- `profile in profiles`
- `profile[index]`

property active_profile

The currently active profile

Activating another profile is done by assigning this attribute to another profile:

```
new_profile = remote_device.profiles[index]
remote_device.active_profile = new_profile
```

The value assigned to this property must be a *ConfigProfile* that belongs to this device, i.e. the following must be `True`:

```
new_profile in remote_device.profiles
```

Type
ConfigProfile

property address_info

Information about network address settings

Note: Accessing this attribute requires the correct password be set on the object.

Type
AddressInfo

property connection_status

Connection status information

The returned tuple is a `(state, tx_rate, rx_rate, channel, rssi, tx_power)` namedtuple of:

1. `state` (*NetworkState*): Network connection state
2. `tx_rate` (`int`): Transmit rate in kbit/s
3. `rx_rate` (`int`): Receive rate in kbit/s
4. `channel` (`int`): Channel
5. `rssi` (`int`): Receive Signal Strength Indicator
6. `tx_power` (`int`): Transmit power level in dB

Note: Accessing this attribute requires the correct password be set on the object.

Type
ConnectionStatus

connection_test()

Creates a connection test for this device

Returns
ConnectionTest

See the documentation of *ConnectionTest* for more information.

Note: Accessing this attribute requires the correct password be set on the object.

property hostname

Device's hostname

Note: Accessing this attribute requires the correct password be set on the object.

Type
str

password_valid()

Checks whether the password set is valid

Returns
bool – True if the password is valid, False otherwise.

wlan_scan(*active=False*, *bss_type=BasicServiceSet.ANY*, *domain=RegulatoryDomain.WORLD*)

Creates and starts a wlan scan for this device

Returns
WlanScan

See the documentation of *WlanScan* for more information.

Note: Accessing this attribute requires the correct password be set on the object.

WlanScan

class canlib.kvrlib.**WlanScan**(*config_handle*)

A wlan scan for this device

The device starts scanning as soon as this object is created by *RemoteDevice.wlan_scan*:

```
scan = rdev.wlan_scan()
```

When calling *RemoteDevice.wlan_scan*, you can also specify whether the scan should be an active one, the Basic Service Set (bss) to use, and the regulatory domain:

```
scan = rdev.wlan_scan(
    active=True,
    bss=kvrlib.BasicServiceSet.INFRASTRUCTURE,
    domain=kvrlib.RegulatoryDomain.EUROPE_ETSI,
)
```

Results from the scan are retrieved by iterating over the *WlanScan* object:

for result in scan:
print(result)

When getting the next result, the code will block until a new result is available or until no more results are available, in which case the iteration stops.

Wlan scans are automatically closed when garbage-collected, but they can also be closed manually with *WlanScan.close*.

close()

Closes the internal handle

WlanScanResult

class canlib.kvrlib.**WlanScanResult**(*rssi*, *channel_number*, *mac*, *bss_type*, *ssid*, *security_text*)

property bss_type
Alias for field number 3

property channel_number
Alias for field number 1

property mac
Alias for field number 2

property rssi
Alias for field number 0

property security_text
Alias for field number 5

property ssid
Alias for field number 4

Structures

kvrAddress

class canlib.kvrlib.**kvrAddress**

TypeText = {0: 'UNKNOWN', 1: 'IPV4', 2: 'IPV6', 3: 'IPV4_PORT', 4: 'MAC'}

Type_IPV4 = 1
IP v.4 address.

Type_IPV4_PORT = 3
IP v.4 address with tcp-port.

Type_IPV6 = 2
IP v.6 address.

Type_MAC = 4
Ethernet MAC address.

Type_UNKNOWN = 0
Unknown (e.g., no reply from device).

address
Structure/Union member

type
Structure/Union member

kvrAddressList

class canlib.kvrlib.**kvrAddressList**(*num_of_structs=20*)

STRUCT_ARRAY
Structure/Union member

elements
Structure/Union member

kvrDeviceInfo

class canlib.kvrlib.**kvrDeviceInfo**

accessibility
Structure/Union member

accessibility_pwd
Structure/Union member

availability
Structure/Union member

base_station_id
Structure/Union member

client_address
Structure/Union member

connect()

device_address
Structure/Union member

disconnect()

ean_hi
Structure/Union member

ean_lo
Structure/Union member

encryption_key
Structure/Union member

fw_build_ver
Structure/Union member

fw_major_ver
Structure/Union member

fw_minor_ver
Structure/Union member

host_name
Structure/Union member

name
Structure/Union member

request_connection
Structure/Union member

reserved1
Structure/Union member

reserved2
Structure/Union member

ser_no
Structure/Union member

struct_size
Structure/Union member

usage
Structure/Union member

kvrDeviceInfoList

class canlib.kvrlib.**kvrDeviceInfoList**(*deviceInfos*)

STRUCT_ARRAY
Structure/Union member

elements
Structure/Union member

kvrVersion

class canlib.kvrlib.**kvrVersion**

major
Structure/Union member

minor
Structure/Union member

Service

canlib.kvrlib.service.**query**()
Queries the status of the helper service.

The helper service is installed as a part of the Windows CANlib driver package and is normally set to automatic start.

canlib.kvrlib.service.**start**()
Start the helper service.

The helper service is installed as a part of the Windows CANlib driver package and is normally set to automatic start.

canlib.kvrlib.service.**stop**()

Stop the helper service.

The helper service is installed as a part of the Windows CANlib driver package and is normally set to automatic start.

Miscellaneous

configActiveProfileSet()

canlib.kvrlib.**configActiveProfileSet**(*channel*, *profile_number*)

Set active profile.

configActiveProfileGet()

canlib.kvrlib.**configActiveProfileGet**(*channel*)

Get active profile.

configNoProfilesGet()

canlib.kvrlib.**configNoProfilesGet**(*channel*)

Get the maximum number of profile(s) the device can store.

When a remote device is connected to the host it can be configured. The remote device can hold a number of different profiles.

deviceGetServiceStatus()

canlib.kvrlib.**deviceGetServiceStatus**(*device_info*)

Returns local connection status of the selected device.

deviceGetServiceStatusText()

canlib.kvrlib.**deviceGetServiceStatusText**(*device_info*)

Returns local connection status of the selected device as ASCII text.

dllversion()

canlib.kvrlib.**dllversion**()

Get the kvrlib version number as a *VersionNumber*

generate_wep_keys()

canlib.kvrlib.**generate_wep_keys**(*pass_phrase*)

Generates four 64-bit and one 128-bit WEP keys

Parameters
pass_phrase (str) – The pass phrase to use to generate the keys.

Returns a (key64, key128) (*WEPKeys*) namedtuple, where key64 is a list of four bytes object with the four 64-bit WEP keys, and where `key128 is a bytes object with the single 128-bit WEP key.

generate_wpa_keys()

canlib.kvrlib.**generate_wpa_keys**(*pass_phrase*, *ssid*)

Generate a WPA key from a pass phrase and ssid

Parameters

- **pass_phrase** (str) – The pass phrase to use to generate the key.
- **ssid** (str) – The SSID to use to generate the key.

Returns
bytes – The generated WPA key.

hostname()

canlib.kvrlib.**hostname**(*ean*, *serial*)

Generate a hostname from ean and serial number

Parameters

- **ean** (*canlib.EAN*) – European Article Number
- **serial** (int) – Serial number

kvrDiscovery

class canlib.kvrlib.**kvrDiscovery**(*kvrlib=None*)

clearDevicesAtExit(*onoff*)

close()

classmethod getDefaultAddresses(*addressTypeFlag=AddressTypeFlag.BROADCAST*, *listSize=20*)

getResults()

setAddresses(*addressList*)

setEncryptionKey(*device_info*, *key*)

setPassword(*device_info*, *password*)

setScanTime(*delay_ms*, *timeout_ms*)

start(*delay_ms=None*, *timeout_ms=None*)

storeDevices(*deviceInfos*)

kvrConfig

class canlib.kvrlib.**kvrConfig**(*kvrlib=None*, *channel=0*, *mode=0*, *password=''*, *profile_no=0*)

MODE_ERASE = 2

MODE_R = 0

MODE_RW = 1

XML_BUFFER_SIZE = 2046

clear()

close()

getXml()

openEx(*channel=None*, *mode=None*, *password=None*, *profile_no=None*)

setXml()

unload()

canlib.kvrlib.**unload**()
Unloads library stuff.

verify_xml()

canlib.kvrlib.**verify_xml**(*xml_string*)
Verify that the xml string complies with both the DTD and internal restrictions

WEPKeys

class canlib.kvrlib.**WEPKeys**(*key64*, *key128*)

property key128
Alias for field number 1

property key64
Alias for field number 0

1.6.12 linlib

Exceptions

LinError

exception canlib.linlib.**LinError**

Base class for exceptions raised by the linlib class

Looks up the error text in the linlib dll and presents it together with the error code and the wrapper function that triggered the exception.

LinGeneralError

exception canlib.linlib.**LinGeneralError**(*status*)

A linlib error that does not (yet) have its own Exception

> **Warning:** Do not explicitly catch this error, instead catch *LinError*. Your error may at any point in the future get its own exception class, and so will no longer be of this type. The actual status code that raised any *LinError* can always be accessed through a `status` attribute.

LinNoMessageError

exception canlib.linlib.**LinNoMessageError**

Bases: *LinError*

No messages where availible

status = -1

LinNotImplementedError

exception canlib.linlib.**LinNotImplementedError**

Bases: *LinError*

Feature/function not implemented in the device

status = -26

Channel

openChannel()

canlib.linlib.**openChannel**(*channel_number*, *channel_type*, *bps=None*)

Open a channel to a LIN interface.

Parameters

- **channel_number** (`int`) – The number of the channel. This is the same channel number as used by *canlib.openChannel*.

- **channel_type** (*ChannelType*) – Whether the LIN interface will be a master or slave.
- **bps** (`int` or `None`) – If not `None`, *Channel.setBitrate* will be called with this value before the channel is returned.

Returns
(*Channel*) – The opened channel

Note: For DRV Lin: The cable must be powered and connected to a LAPcan channel. For Kvaser LIN Leaf: The Leaf must be powered from the LIN side.

---

openMaster()

`canlib.linlib.`**`openMaster`**(*channel_number*, *bps=None*)

Open a channel as a master

This function simply calls *openChannel* with `channel_type` set to *ChannelType.MASTER*.

openSlave()

`canlib.linlib.`**`openSlave`**(*channel_number*, *bps=None*)

Open a channel as a slave

This function simply calls *openChannel* with `channel_type` set to *ChannelType.SLAVE*.

Channel

`class` `canlib.linlib.`**`Channel`**(*handle*)

A LINlib channel

This class is normally instantiated with *openMaster* or *openSlave*.

Channels are automatically closed on garbage collection, and can also be used as context managers in which case they close as soon as the context exits.

`busOff`()

Go bus off

This function deactivates the LIN interface. It will not participate further in the LIN bus traffic.

`busOn`()

Go bus on

This function activates the LIN interface.

`clearMessage`(*msg_id*)

Clear a message buffer for a LIN slave

The message buffer will not answer next time it is polled.

close()

Close this LINlib channel

Closes an open handle to a LIN channel.

Note: It is normally not necessary to call this function directly, as the internal handle is automatically closed when the *Channel* object is garbage collected.

New in version 1.6.

getCanHandle()

Return the CAN handle given an open LIN handle

Deprecated since version 1.20: Use *get_can_channel* instead.

getFirmwareVersion()

Retrieve the firmware version from the LIN interface

Returns a *FirmwareVersion* namedtuple containing `boot_version` and `app_version` that are *canlib.VersionNumber* namedtuples. If only one of these is needed, the return value can be unpacked as such:

```
boot_ver, app_ver = channel.getFirmwareVersion()
```

Note: For newer interfaces use *getChannelData* with *ChannelData.CARD_FIRMWARE_REV* instead.

The version numbers aren't valid until *Channel.busOn* has been called.

The firmware in the LIN interface is divided into two parts, the boot code and the application. The boot code is used only when reprogramming (reflashing) the LIN interface. The application handles all LIN communication.

Version numbers are, since the precambric era, divided into a major version number, a minor version number and a build number. These are usually written like, for example, 3.2.12. Here the major number is 3, the minor number 2 and the build number 12.

get_can_channel()

Return the CAN Channel used by this LIN Channel

Note: Since the returned *canlib.Channel* is owned and controlled by linlib, this function should be used with great care.

New in version 1.20.

read(*timeout=0*)

Read a message from the LIN interface

If a message is available for reception, linOK is returned. This is a non-blocking call. It waits until a message is received in the LIN interface, or the specified timeout period elapses.

This may return a frame sent by *writeMessage* or *writeWakeup*.

Note: This call will also return echoes of what the LIN interface is transmitting with *writeMessage*. In other words, the LIN interface can hear itself.

Parameters
timeout (int) – Timeout in milliseconds.

Returns
(*canlib.LINFrame*)

requestMessage(*msgid*)

Request a message from a slave

This function writes a LIN message header to the LIN bus. A slave in the system is then expected to fill in the header with data.

Note: This call is only available in master mode.

setBitrate(*bps*)

Set the bitrate in bits per second

This function sets the bit rate for a master, or the initial bit rate for a slave. The LIN interface should not be on-bus when this function is called.

Note: The LIN Interface should not be on bus. Supported bit rates are 1000 - 20000 bits per second.

setupIllegalMessage(*msgid*, *disturb_flags*, *delay*)

Create a corrupted LIN message

Using this function, it is possible to use the LIN interface to create corrupted LIN messages. You call the function once for each LIN identifier that should be affected.

To return to normal mode, either restart the LIN interface (by going off bus and on the bus again) or call the function with delay and disturb_flags set to zero.

Parameters

- **msgid** (int) – The identifier of the LIN message
- **disturb_flags** (*MessageDisturb*) – One or more of the *MessageDisturb* flags.
- **delay** (int) – The delay parameter will result in a delay of this many bittimes after the header and before the first data byte.

Note: The LIN Interface must be on bus for this command to work. It is supported in firmware version 2.4.1 and later.

setupLIN(*flags=Setup.VARIABLE_DLC*, *bps=0*)

Setup the LIN interface

This function changes various settings on a LIN Interface that is on bus. When going on bus, the bit rate and the flag values listed below are set to the default value (either as hard-coded in the firmware, or as stored in the non-volatile memory of the LIN Interface).

With this function, you can do one or more of the following things:

- Select checksum according to LIN 2.0
- Turn variable message length off. The message length then will depend on the message ID.

In master mode it is also possible to change the bit rate without going off bus first.

Note: The LIN Interface must be on bus for this command to work. It is supported in firmware version 2.5.1 and later. For LIN 2.0 compliance, you must specify both LIN_ENHANCED_CHECKSUM and LIN_VARIABLE_DLC.

Parameters

- **flags** (*Setup*) – One or more of the *Setup* flags
- **bps** (int) – The bit rate in bits per second. This parameter can be used only in master mode. The bit rate is set without going off bus.

updateMessage(*frame*)

Update a message buffer in a slave

This function updates a message buffer in a slave. The contents of the message buffer will be used the next time the slave is polled for the specified LIN message id.

Note: The LIN Interface must be on bus.

Parameters

frame (*canlib.Frame*) – The information to be updated. Only the *Frame.id*, *Frame.data*, and *Frame.dlc* attributes are used. Note that the frame can, but not need not, be a *LINFrame*.

writeMessage(*frame*)

Write a LIN message

Write a LIN message. It is advisable to wait until the message is echoed by *read* before transmitting a new message, or in case of a schedule table being used, transmit the next message when the previous one is known to be complete.

Note: Only available in master mode

Parameters

frame (*canlib.Frame*) – *Frame.data*, and *Frame.dlc* attributes are used. Note that the frame can, but not need not, be a *LINFrame*.

writeSync(*timeout*)

Make sure all message transmitted to the interface have been received

timeout is in milliseconds.

When messages are transmitted to the LIN Interface, they are queued by the driver before appearing on the CAN bus.

If the LIN Interface is in master mode and a LIN message has been transmitted with *writeMessage*, this function will return when the LIN Interface has received the message. If another LIN message is being received or transmitted, the message will not be transmitted on the LIN bus at once. And even if the LIN Interface is idle, the header of the new message will just have been started when *writeSync* returns.

After calling *updateMessage* and *clearMessage* for a slave, this function is enough to know that the LIN Interface is updated.

After *writeMessage*, it is advisable to wait until the message is echoed by *read* before transmitting a new message, or in case of a schedule table being used, transmit the next message when the previous one is known to be complete.

When, in master mode, a message should be transmitted after a poll (reception) is done, it might be necessary to call *writeMessage* before the result is received via *read* as the LIN Interface waits up to the maximum frame length before knowing a received message is complete. A new message to transmit will force completion if the currently received one.

writeWakeup(*count=0*, *interval=1*)

Write one or more wakeup frames

If *count* is zero (the default), one single wakeup frame is transmitted. If *count* > 1, several wakeup frames are transmitted spaced with *interval* bit times. The LIN interface will interrupt the sequence when a LIN message or another command is received. The stream of wakeups will be recived as incoming messages with the *MessageFlag.RX* flag.

Parameters

- **count** (int) – The number of wakeup frames to send.
- **interval** (int) – The time, in bit times, between the wakeup frames.

FirmwareVersion

class canlib.linlib.**FirmwareVersion**(*boot_version*, *app_version*)

property app_version

Alias for field number 1

property boot_version

Alias for field number 0

Enumerations

ChannelData

class canlib.linlib.**ChannelData**(*value*)

linCHANNELDATA_xxx

These defines are used in *getChannelData*.

CARD_FIRMWARE_REV = 9

Firmware version from the LIN interface.

ChannelType

class canlib.linlib.**ChannelType**(*value*)

Flags for *openChannel*

MASTER = 1

The LIN interface will be a LIN master

SLAVE = 2

The LIN interface will be a LIN slave

Error

class canlib.linlib.**Error**(*value*)

An enumeration.

CANERROR = -15

Internal error in the driver

DRIVER = -18

CAN driver type not supported

DRIVERFAILED = -19

DeviceIOControl failed

ERRRESP = -16

There was an error response from the LIN interface

INTERNAL = -22

Internal error in the driver

INVHANDLE = -14

Handle is invalid

LICENSE = -21

The license is not valid

MASTERONLY = -5

The function is only for a master.

NOCARD = -20

The card was removed or not inserted

NOCHANNELS = -10

No channels avaliable

NOHANDLES = -13

Can't get handle

NOMEM = -9

Out of memory

NOMSG = -1

No messages available

NOTFOUND = -8

Specified hardware not found. This error is reported when the LIN transceiver isn't powered up

NOTINITIALIZED = -12

Library not initialized

NOTRUNNING = -3

Handle not on-bus. Some functions requires that the handle is on-bus before being called.

NOT_IMPLEMENTED = -26

The requested feature or function is not implemented in the device you are trying to use it on

NO_ACCESS = -23

Access denied

NO_REF_POWER = -25

Function not supported in this version

PARAM = -7

Error in parameter

RUNNING = -4

Handle not off-bus. Some functions requires that the handle is off-bus before being called.

SLAVEONLY = -6

The function is only for a slave.

TIMEOUT = -11

Timeout occurred

VERSION = -24

Function not supported in this version

WRONGRESP = -17

The LIN interface response wasn't the expected one

MessageDisturb

class canlib.linlib.**MessageDisturb**(*value*)

LIN illegal message flags

CSUM = 1

The checksum of transmitted messages will be inverted

PARITY = 2

The two parity bits will be inverted, used only in master mode.

MessageFlag

class canlib.linlib.**MessageFlag**(*value*)

LIN message flags

The following flags is used in *LINFrame.flags*.

BIT_ERROR = 128

Bit error when transmitting.

CSUM_ERROR = 16

Checksum error

NODATA = 8

No data, only a header.

PARITY_ERROR = 32

ID parity error

RX = 2

The message was something we received from the bus.

SYNCH_ERROR = 64

A synch error

SYNC_ERROR = 64

A synch error

TX = 1

The message was something we transmitted on the bus.

WAKEUP_FRAME = 4

MessageParity

class canlib.linlib.**MessageParity**(*value*)

LIN message parity

- *MessageParity.STANDARD* == LIN_MSG_USE_STANDARD_PARITY
- *MessageParity.ENHANCED* == LIN_MSG_USE_ENHANCED_PARITY

ENHANCED = 8

Use enhanced (2.x) parity for the specified msg

STANDARD = 4

Use standard (1.x) parity for the specified msg

Setup

class canlib.linlib.**Setup**(*value*)

Used in *Channel.setupLIN*

ENHANCED_CHECKSUM = 1

When specified, the LIN interface will use the "enhanced" checksum according to LIN 2.0. Note that as per the LIN 2.0 spec) the enhanced checksum is not used on the diagnostic frames even if the *Setup.ENHANCED_CHECKSUM* setting is in effect.

VARIABLE_DLC = 2

When specified, turns variable message length on, so the the message length will depend on the message ID.

MessageInfo

class canlib.linlib.**MessageInfo**

Provides more information about the LIN message.

The precision of the timing data given in us (microseconds) can be less than one microsecond; for low bitrates the lowest bits might always be zero.

The min and max values listed inside [] of the message timing values can be calculated from the LIN specification by using the shortest (0 bytes) or longest (8 bytes) messages at the lowest or highest allowed bitrate.

Note: The LIN interface will accept messages that are a bit out-of-bounds as well.

timestamp

Kvaser DRV Lin timestamp - Timestamp in milliseconds of the falling edge of the synch break of the message. Uses the canlib CAN timer.

Kvaser LIN Leaf timestamp - Timestamp in milliseconds of the falling edge of the synch break of the message. Uses the canlib CAN timer.

Note: All Kvaser Leaf with Kvaser MagiSync&tm; are synchronized (also with CAN channels).

bitrate

The bitrate of the message in bits per seconds. Range [1000 .. 20000] (plus some margin)

byteTime

Start time in microseconds of each data byte. In case of 8-byte messages, the crc time isn't included (but can be deduced from frameLength).

Note: Not supported by all devices.

checkSum

The checksum as read from the LIN bus. Might not match the data in case of *MessageFlag.CSUM_ERROR*.

frameLength

The total frame length in microseconds; from the synch break to the end of the crc. [2200 .. 173600]

idPar

The id with parity of the message as read from the LIN bus. Might be invalid in case of *MessageFlag.PARITY_ERROR*.

synchBreakLength

Length of the synch break in microseconds. [650 .. 13000], [400 .. 8000] for a wakeup signal.

synchEdgeTime

Time in microseconds of the falling edges in the synch byte relative the falling edge of the start bit.

Note: Not supported by all devices.

Miscellaneous

dllversion()

canlib.linlib.**dllversion**()

Retrieve the LIN library version as a *VersionNumber*

Note: Requires CANlib v5.3

getChannelData()

canlib.linlib.**getChannelData**(*channel_number*, *item=ChannelData.CARD_FIRMWARE_REV*)

This function can be used to retrieve certain pieces of information about a channel.

Note: You must pass a channel number and not a channel handle.

Parameters

- **channel** (*int*) – The number of the channel you are interested in. Channel numbers are integers in the interval beginning at 0.
- **item** (*ChannelData*) – This parameter specifies what data to obtain for the specified channel. Currently the only item available is *CARD_FIRMWARE_REV*, which is the default.

getTransceiverData

canlib.linlib.**getTransceiverData**(*channel_number*)

Get the transceiver information for a CAN channel

The application typically uses this call to find out whether a particular CAN channel has a LIN interface connected to it. For a Kvaser LIN Leaf it retrieves the transceiver type and device information.

This function call will open the CAN channel, but no CAN messages are transmitted on it. In other words, it's risk-free to use even if no LIN interface is connected, or if the channel is connected to a CAN system.

Note: Attempts to use the channel for LIN communication will be meaningful only if the returned *TransceiverData*'s ~type~ attribute is one of *TransceiverType.LIN* or *TransceiverType.CANFD_LIN*

A LIN interface need not be powered for this call to succeed.

The information may not always be accurate. Especially after changing transceiver on a running LAPcan card, you should go on bus and off bus again to be sure the transceiver information is updated.

---

initializeLibrary

canlib.linlib.**initializeLibrary**()

Initialize LINlib

Note: LINlib is automatically initialized when `canlib.linlib` is imported. This function is only necessary when LINlib has been manually unloaded with *unloadLibrary*.

TransceiverData

class canlib.linlib.**TransceiverData**(*ean*, *serial*, *type*)

property ean

Alias for field number 0

property serial

Alias for field number 1

property type

Alias for field number 2

unloadLibrary

canlib.linlib.**unloadLibrary**()

Deinitialize LINlib

This function de-initializes the LIN library. After this function is called `linInitializeLibrary` must be called before any other LIN function is called.

1.6.13 j1939

This module holds definitions of J1939 Protocol Data Unit (PDU) frames.

Example of usage:

```
>>> from canlib import Frame, j1939
>>> from canlib.canlib import MessageFlag
>>> def frame_from_pdu(pdu):
...     can_id = j1939.can_id_from_pdu(pdu)
...     frame = Frame(
...         id_=can_id,
...         data=pdu.data,
...         flags=MessageFlag.EXT,
...     )
...     return frame
>>> pdu = j1939.Pdu1(
...     p=3, edp=0, dp=0, pf=0x99, ps=0xfe, sa=0xfe,
...     data=[1]
... )
>>> frame_from_pdu(pdu)
Frame(id=211418878, data=bytearray(b'\x01'), dlc=1, flags=<MessageFlag.EXT: 4>,␣
↪timestamp=None)
```

The particular characteristics of J1939 are:

- Extended CAN identifier (29 bit)
- Bit rate 250 kbit/s
- Peer-to-peer and broadcast communication
- Transport protocols for up to 1785 data bytes
- Network management

- Definition of parameter groups for commercial vehicles and others
- Manufacturer specific parameter groups are supported
- Diagnostics features

(Extended) Data Page Bit

Extended Data page	Data page	Description
0	0	SAE J1939 Page 0 Parameter Groups
0	1	SAE J1939 Page 1 Parameter Groups (NMEA2000)
1	1	SAE J1939 reserved
1	1	ISO 15765-3 defined

New in version 1.18.

Protocol Data Units

class canlib.j1939.**Pdu**(*, *p: int*, *edp: int*, *dp: int*, *pf: int*, *ps: int*, *sa: int*, *data: Optional[List[int]] = None*)

Protocol Data Unit in j1939.

Base class with attributes common to *Pdu1* and *Pdu2*

data: Optional[List[int]]

data field

dp: int

data page

edp: int

extended data page

p: int

priority

pf: int

PDU format

ps: int

PDU specific

sa: int

source address

class canlib.j1939.**Pdu1**(*, *p: int*, *edp: int*, *dp: int*, *pf: int*, *ps: int*, *sa: int*, *data: Optional[List[int]] = None*, *da: Optional[int] = None*, *pgn: Optional[int] = None*)

Protocol Data Unit, Format 1

When *Pdu.pf* < 240, the PDU Specific field is a Destination Address and

pgn = Extended Data Page + Data Page + PDU Format + “00”

da: Optional[int]

destination address, *Pdu.ps*

pgn: Optional[int]

parameter group number

class canlib.j1939.**Pdu2**(*, *p: int*, *edp: int*, *dp: int*, *pf: int*, *ps: int*, *sa: int*, *data: Optional[List[int]] = None*, *ge: Optional[int] = None*, *pgn: Optional[int] = None*)

Protocol Data Unit, Format 2

When *Pdu.pf* >= 240, the PDU Specific field is the Group Extension

pgn = Extended Data Page + Data Page + PDU Format + Group Extension

ge: Optional[int]

group extension, equal to *Pdu.ps*

pgn: Optional[int]

parameter group number

Converting CAN Id

For a j1939 message, the CAN identifier is divided into the following fields:

Priority	Extended Data Page	Data Page	PDU Format	PDU Specific	Source Address
3 bit	1 bit	1 bit	8 bit	8 bit	8 bit

Use *pdu_from_can_id* and *can_id_from_pdu* to convert.

canlib.j1939.**pdu_from_can_id**(*can_id*)

Create j1939 Protocol Data Unit object based on CAN Id.

Parameters
can_id (int) – CAN Identifier

Returns
Pdu1 or *Pdu2* depending on value of can_id

canlib.j1939.**can_id_from_pdu**(*pdu*)

Extract CAN Id based on j1939 Protocol Data Unit object.

Parameters
pdu (*Pdu1* or *Pdu2*) – Protocol Data Unit

Returns
can_id (int) – CAN Identifier

CHAPTER

TWO

RELEASE NOTES

2.1 Release Notes

This is the release notes for the pycanlib module.

Contents

- *Release Notes*
  - *New Features and Fixed Problems in V1.22.565 (08-SEP-2022)*
  - *New Features and Fixed Problems in V1.21.463 (29-MAY-2022)*
  - *New Features and Fixed Problems in V1.20.360 (15-FEB-2022)*
  - *New Features and Fixed Problems in V1.19.205 (13-SEP-2021)*
  - *New Features and Fixed Problems in V1.18.846 (25-MAY-2021)*
  - *New Features and Fixed Problems in V1.17.748 (16-FEB-2021)*
  - *New Features and Fixed Problems in V1.16.588 (09-SEP-2020)*
  - *New Features and Fixed Problems in V1.15.483 (27-MAY-2020)*
  - *New Features and Fixed Problems in V1.14.428 (02-APR-2020)*
  - *New Features and Fixed Problems in V1.13.390 (24-FEB-2020)*
  - *New Features and Fixed Problems in V1.12.251 (08-OCT-2019)*
  - *New Features and Fixed Problems in V1.11.226 (13-SEP-2019)*
  - *New Features and Fixed Problems in V1.10.102 (12-MAY-2019)*
  - *New Features and Fixed Problems in V1.9.909 (03-MAR-2019)*
  - *New Features and Fixed Problems in V1.8.812 (26-NOV-2018)*
  - *New Features and Fixed Problems in V1.7.741 (16-SEP-2018)*
  - *New Features and Fixed Problems in V1.6.615 (13-MAY-2018)*
  - *New Features and Fixed Problems in V1.5.525 (12-FEB-2018)*
  - *New Features and Fixed Problems in V1.4.373 (13-SEP-2017)*
  - *New Features and Fixed Problems in V1.3.242 (05-MAY-2017)*
  - *New Features and Fixed Problems in V1.2.163 (15-FEB-2017)*

- *New Features and Fixed Problems in V1.1.23 (28-SEP-2016)*
- *New Features and Fixed Problems in V1.0.10 (15-SEP-2016)*

2.1.1 New Features and Fixed Problems in V1.22.565 (08-SEP-2022)

- `canlib`:
  - Added support for object buffers *canlib.objbuf*.

2.1.2 New Features and Fixed Problems in V1.21.463 (29-MAY-2022)

- General:
  - Added CAN FD versions of examples:
    - *Send Random Messages on CAN Channel* (*CAN FD version*), and
    - *Monitor Channel Using CAN Database* (*CAN FD version*).
- `canlib`:
  - Added exceptions *canlib.CanTimeout* and `canlib.CanOverflowError`.
  - Added new bitrate constant *BITRATE_2M_60P*.
  - Added support for *canlib.MessageFlag.LOCAL_TXACK*. This requires CANlib SDK v5.39 or newer.
  - Checking for *canlib.MessageFlag.OVERRUN* using `in` will now return `True` if either *SW_OVERRUN* or *HW_OVERRUN* is set.
- `kvadblib`:
  - Added read only property *kvadblib.Message.canflags* which returns relevant `VFrameFormat` attributes converted into *canlib.MessageFlag* (e.g. the J1939 `VFrameFormat` attribute does not have a corresponding flag and is excluded).
  - *kvadblib.Message.asframe()* now include CAN FD flags on the returned *Frame*. Note that this requires CANlib SDK v5.39 or newer.

2.1.3 New Features and Fixed Problems in V1.20.360 (15-FEB-2022)

- General:
  - Python 3.10 is now officially supported.
  - Fixed *canlib.connected_devices()* to ignore removed devices, instead of raising an exception.
  - Added *canlib.exceptions.CanGeneralError* to documentation, noting that this should *not* be caught explicitly.
  - Extracted tutorial sample code into standalone files, updated bus parameters in CAN FD code to work with U100.
- `canlib`:
  - Removed internal attribute `Channel.flags`, use `canlib.ChannelData.channel_flags` instead.
  - Corrected return value of *is_can_fd* when channel was opened explicitly using *NO_INIT_ACCESS*. Now also always returns a `bool`.

  - Added *ChannelData.bus_param_limits* (wraps canCHANNELDATA_BUS_PARAM_LIMITS)
  - Added t Programming chapter to documentation.
  - Corrected name of bitrate constant inside table in "Set CAN Bitrate" chapter.
- `linlib`:
  - Deprecated *getCanHandle*, use *get_can_channel* instead.
- `kvlclib`:
  - *kvlclib.WriterFormat.getPropertyDefault* and *kvlclib.ReaderFormat.getPropertyDefault* now returns `None` if property do not support get/set, as e.g. *SIGNAL_BASED*.
  - Added support for experimental format KME60
  - Clarified usage of *kvlclib.Converter.addDatabaseFile()*.
- `kvadblib`:
  - Added support for Attribute Definition of type HEX, *kvadblib.AttributeType.HEX*.
  - Comment and Unit on a signal now converts cp1252 coding to utf-8.
  - Added support for experimental format KME60

2.1.4 New Features and Fixed Problems in V1.19.205 (13-SEP-2021)

- General:
  - Updated docstrings, mainly of lower level classes.
  - Modernized code, mainly conversions to f-strings.
- `canlib.dllLoader`:
  - Setting the environment variable `READTHEDOCS == True` inhibits loading of shared libraries. Used e.g. when building documentation on ReadTheDocs.
- `canlib.kvamemolibxml`:
  - The functions *xmlGetLastError*, *xmlGetValidationWarning* and *xmlGetValidationError* now returns enum classes when possible.
- `canlib.kvrlib`:
  - Minor readability updates for `kvrDeviceInfo.__str__`

2.1.5 New Features and Fixed Problems in V1.18.846 (25-MAY-2021)

- `canlib.canlib`:
  - Added LEDs 4 through 11 to *canlib.canlib.LEDAction* (needs CANlib v5.19+).
- `canlib.kvadblib`:
  - Default value of EnumAttribute is now returned as `int`
  - Added wrapper for kvaDbGetMsgByPGNEx
- `canlib.kvlclib`:
  - Added wrapper for kvlcFeedLogEvent
- Added *canlib.j1939* module for some j1939 helpers.

2.1.6 New Features and Fixed Problems in V1.17.748 (16-FEB-2021)

- canlib.canlib:
  - Corrected *set_bus_params_tq* regarding type of flag attribute.
  - Added support for using *setBusParams* and *getBusParams* for channels that were opened using *BusParamsTq*.
  - Added *Bitrate* and *BitrateFD* enums for use with *setBusParams* and *openChannel*. canlib.canBITRATE_xxx and canlib.canFD_BITRATE_xxx constants are still supported but deprecated.
  - Added enum member BITRATE_8M_80P to *BitrateFD* and constant canlib.canFD_BITRATE_8M_80P.
- canlib.kvlclib
  - Added exception *KvlcFileExists*.

2.1.7 New Features and Fixed Problems in V1.16.588 (09-SEP-2020)

- canlib.canlib:
  - Added support for new bus parameter API in CANlib v.5.34. See section *Set CAN Bitrate* for more information.
  - Added attributes to canlib.IOControl.__dir__ and canlib.ChannelData.__dir__ in order to better support auto completion in IDEs.
  - Deprecated *canlib.Device.channel*, use *canlib.Device.open_channel* instead, which correctly handles keyword arguments
  - Added new Open flag *canlib.canlib.Open.NOFLAG* for parameter flags.
- canlib.kvadblib:
  - Corrected *interpret* when looking for CAN messages with extended id.
  - Updated *get_message* so that it requires EXT (bit 31) to be set on id if using extended id:s.
  - Added a new argument flags to *get_message_by_id*. If using messages with extended id:s, EXT should be set on flags.
- canlib.kvlclib:
  - The file_format parameter in *canlib.kvlclib.Converter.setInputFile* now accepts *ReaderFormat* as well.
  - Added a newer version of the BLF format, now also with CAN FD support 'canlib.kvlclib.FileFormat.VECTOR_BLF_FD'. The format has both read and write capabilities.

2.1.8 New Features and Fixed Problems in V1.15.483 (27-MAY-2020)

- Dropped support for v2.7, v3.4 and v3.5, added v3.7 and v3.8.

2.1.9 New Features and Fixed Problems in V1.14.428 (02-APR-2020)

- Minor changes.

2.1.10 New Features and Fixed Problems in V1.13.390 (24-FEB-2020)

- `canlib.canlib`:
  - Added *HandleData* to wrap canGetHandleData. Also added *channel_data* as a helper function.
  - IOControl now returns utf-8 decoded strings instead of "bytes in string".
  - Fixed a bug where `isconnected` would return `False` if the `channel_number` attribute was larger than the total number of available CANlib channels, regardles of if the device was connected or not.
- `canlib`:
  - Corrected *Frame* comparison (!=) with other types, e.g. None

2.1.11 New Features and Fixed Problems in V1.12.251 (08-OCT-2019)

- Minor changes.

2.1.12 New Features and Fixed Problems in V1.11.226 (13-SEP-2019)

- `canlib.canlib`:
  - Added a slight delay in get_bus_statistics because the underlying functions in CANlib are asynchronous.
  - Added *read_error_counters* and `iocontrol` `clear_error_counters`.
  - Added *getBusOutputControl*.
  - Added *fileDiskFormat* that formats the disk in a remote device, i.e Kvaser DIN Rail.
- `canlib.BoundSignal.value`:
  - If the signal is an enum-signal, and the signal's value is not found in the enum definition, the raw value is now returned.
- `canlib.kvmlib`:
  - Marked using kvmlib class as deprecated (was deprecated in v1.6)
  - Replaced *estimate_events* with *Kme.event_count_estimation* in order to have same name as *LogFile.event_count_estimation*. Old function name is now deprecated.
  - When found, new 64 bit version of the dll call, `kvmLogFileMountEx`, `kvlcEventCountEx`, and `kvmKmeCountEventsEx` (added in CANlib v5.29), is now used.
  - Added *log_type* for supporting the different log types generated by Kvaser Memorator Light HS v2.
- `canlib.kvadblib`:
  - *Dbc* raises `KvdDbFileParse` if the dbc file loaded contains syntax errors.

2.1.13 New Features and Fixed Problems in V1.10.102 (12-MAY-2019)

- Reference documentation has been restructured.
- *Channel*:
  - Added support for slicing environment variables declared as char. Replaced low level function `scriptEnvvarSetData` with *script_envvar_set_data* and added *DataEnvVar* which is now returned when a char envvar is returned.
- `canlib.kvadblib`:
  - Error messages from the CAN database parser in *Dbc* can be retrieved using *get_last_parse_error()*.

2.1.14 New Features and Fixed Problems in V1.9.909 (03-MAR-2019)

- `canlib.kvadblib`:
  - Error texts are now fetched from the dll using kvaDbGetErrorText().
- `canlib.kvlclib`:
  - Added support for DLC mismatch handling included in CANlib v5.27
- `canlib.kvDevice`:
  - The `canlib.kvDevice.kvDevice` class is now deprecated, use *canlib.Device* instead
- *canlib.Device*:
  - Added method `Device.issubset` as a helper to find loosely specified devices.
- *canlib.canlib.iopin*:
  - Added attributes `fw_version` and `serial` to `IoPin`. To read these attributes, CANlib v5.27 is needed.
  - `AddonModule` is a new class, holding attributes of one add-on module.
  - `Config.modules` is now an attribute, calculated at creation time and containing an ordered list of `AddonModule` objects. The old functionality has been moved to `Config._modules`
  - `Config.issubset` is a new method to identify if a configuration contains the expected add-on modules.

2.1.15 New Features and Fixed Problems in V1.8.812 (26-NOV-2018)

- `canlib.canlib`:
  - Fixed issue were Channel.handle attribute would not be initialized when opening of the channel failed.
  - Added experimental support for accessing IO-pins on sub modules of the Kvaser DIN Rail SE 400S that was added to CANlib v5.26. This includes a new module *canlib.canlib.iopin*.
- `canlib.kvadblib`:
  - Fixed issue with signals were multiplexing mode, and scaling (factor and offset) returned wrong values from a loaded .dbc file.
  - Added show_all argument to Dbc.messages. Dbc.__iter__ now set show_all to False in order to skip VECTOR__INDEPENDENT_SIG_MSG messages.

2.1.16 New Features and Fixed Problems in V1.7.741 (16-SEP-2018)

- canlib.kvmlib:
  - Added canlib.kvmlib.event_count_estimation
  - Added canlib.kvmlib.kme Previous kvmlib.kmeXXX functions are now deprecated.
- canlib.canlib:
  - Added *canlib.canlib.ScriptStatus*
  - Added enums to *canlib.canlib.ChannelCap*
  - Fixed canlib.canlib.canWriteSync
- canlib.kvlclib:
  - Added API to access information about reader formats.
  - Added kvlclib.Property to replace old PROPERTY_XXX constants which are now deprecated.
  - Added kvlclib.reader_formats and kvlclib.writer_formats to replace now deprecated kvlclib.WriterFormat.getFirstWriterFormat and kvlclib.WriterFormat.getNextWriterFormat.

2.1.17 New Features and Fixed Problems in V1.6.615 (13-MAY-2018)

- Updated for CANlib SDK v5.23.
- Getting version numbers should now be done with dllversion(), which will return *canlib.BetaVersionNumber* if the dll is marked as Beta. Also added canlib.prodversion() to return the CANlib product version number.
- canlib.device:
  - New *canlib.device.Device* class (available as *canlib.Device*) that is a simpler version of kvDevice. *canlib.device.Device* objects can be defined using an EAN and serial number, or a connected device can be searched for using canlib.device.Device.find. These objects do not require the device to stay connected, and can be used to later create most other canlib objects, e.g. *canlib.canlib.Channel*, *canlib.kvmlib.Memorator*, etc.
  - New canlib.device.connected_devices which returns an iterator of *canlib.device.Device* objects, one for each device currently connected.
- canlib.ean:
  - *canlib.ean.EAN* objects can be tested for equality, both with other *canlib.ean.EAN* objects and with strings.
  - Added CanNotFound exception.
  - *canlib.ean.EAN* objects can now be directly instantiated from string, i.e. ean = canlib.EAN(ean_string) instead of ean = canlib.EAN.from_string(ean_string).
  - *canlib.ean.EAN* objects can be converted back into any of the representations that can be used to create them. See the documentation of *canlib.ean.EAN* for more info.
  - *canlib.ean.EAN* objects can be indexed and iterated upon, yielding the digits as ints.
- canlib.canlib:
  - *canlib.canlib.EnvVar* object raises EnvvarNameError when given an illegal name, instead of AssertionError.

  - *canlib.canlib.openChannel* can now set the bitrate of the channel opened.
  - *canlib.canlib.Channel* objects automatically close their handles when garbage collected
  - *canlib.canlib.Channel* has new methods *canlib.canlib.Channel.scriptRequestText* and *canlib.canlib.Channel.scriptGetText* to get text printed with `printf()` by a script. This text is returned as a *canlib.canlib.ScriptText* object.
- `canlib.kvamemolibxml`:
  - A new, object oriented way of dealing with kvamemolibxml using *canlib.kvamemolibxml.Configuration* objects.
- `canlib.kvmlib`:
  - Improved object model
    - New *canlib.kvmlib.openDevice* function that returns a *canlib.kvmlib.Memorator* object representing a connected Memorator device. See the documentation of *canlib.kvmlib.Memorator* for instructions on how to use this new class to more easily interface with your Memorators.
    - New *canlib.kvmlib.openKmf* function for opening .KMF files that returns a *canlib.kvmlib.Kmf* object that is similar to *canlib.kvmlib.Memorator*. See the docstring of *canlib.kvmlib.Kmf* for more information.
- `canlib.linlib`:
  - Getting version number with *canlib.linlib.dllversion* (requires CANlib SDK v5.23 or newer).
  - Explicit *canlib.linlib.Channel.close* function for forcing a linlib channel's internal handle to be closed.
- `canlib.canlib`:
  - Added support for accessing information within compiled t program (.txe) files.
    - Added wrapper function for `kvScriptTxeGetData`.
    - Added compiled t program (.txe) interface class *canlib.canlib.Txe*.
- `canlib.kvadblib`:
  - enums now returns non-empty dictionary in attribute definition returned from `EnumDefinition.definition`

2.1.18 New Features and Fixed Problems in V1.5.525 (12-FEB-2018)

- Updated for CANlib SDK v5.22.
- Added support for LIN bus API (LINlib)
- Added support for Database API (kvaDbLib) Needs version v5.22 of CANlib SDK to get supported dll.

Restructuring of code in order to make the API simpler and the code base more maintainable have resulted in the following changes (old style is deprecated, shown in details while running Python with the -Wd argument):

- `canlib.kvMessage` has been renamed *canlib.Frame*
  - *canlib.Frame* objects are now accepted and returned when writing and reading channels.
  - The new `canlib.kvadblib` module uses these *canlib.Frame* objects heavily.
- `canlib.canlib`:
  - Added wrapper functions for `canReadStatus` and `canRequestChipStatus`

  - Deprecated use of `canlib.canlib.canlib()` objects; all methods have been moved to the module.
    * See the docstring of `canlib.canlib.CANLib` for more information
  - Simplified the names of the channel-classes (old names are deprecated):
    * The channel class is now *canlib.canlib.Channel*, instead of `canlib.canChannel`.
    * `canlib.canlib.ChannelData_Channel_Flags` is now *canlib.canlib.ChannelFlags*
    * `canlib.canlib.ChannelData_Channel_Flags_bits` is now `canlib.canlib.ChannelFlagBits`
  - *canlib.canlib.Channel* now uses *canlib.Frame* objects for reading and writing.
    * `canlib.Channel.read` now returns a *canlib.Frame* object instead of a tuple. However, *canlib.Frame* objects are largely compatible with tuples.
    * `canlib.Channel.write` takes a single argument, a *canlib.Frame* object. The previous call signature has been taken over by `canlib.Channel.write_raw`.
    * Likewise for `canlib.Channel.writeWait` and its new friend `canlib.Channel.writeWait_raw`.
  - The class `canlib.canlib.canVersion` has been removed, and *canlib.canlib.getVersion* now returns a *canlib.VersionNumber*. The new class still supports conversion to string and accessing `.major` and `.minor` attributes.

- `canlib.kvmlib`:
  - Added wrapper functions for kvmKmeReadEvent.
  - Corrected encoding for Python 3 in kmeOpenFile().
  - Deprecated names for several classes to make them more logical and more pythonic:
    * `canlib.kvmlib.memoMsg` is now *canlib.kvmlib.LogEvent*
    * `canlib.kvmlib.logMsg` is now *canlib.kvmlib.MessageEvent*
    * `canlib.kvmlib.rtcMsg` is now *canlib.kvmlib.RTCEvent*
    * `canlib.kvmlib.trigMsg` is now *canlib.kvmlib.TriggerEvent*
    * `canlib.kvmlib.verMsg` is now *canlib.kvmlib.VersionEvent*
  - The class `canlib.kvmlib.kvmVersion` has been removed, and `canlib.kvmlib.KvmLib.getVersion` now returns a *canlib.VersionNumber*. The new class still supports conversion to string and accessing `.major`, `.minor`, and `.build` attributes.

- `canlib.kvlclib`:
  - Added method `canlib.kvlclib.addDatabaseFile` and helper object *canlib.kvlclib.ChannelMask*.
  - The `canlib.kvlclib.KvlcLib` object has been deprecated.
    * All functions that relate to converters have been moved to the more appropriately named *canlib.kvlclib.Converter*.
      · Some of these functions have been renamed:
      · `IsOutputFilenameNew`, `IsOverrunActive`, and `IsDataTruncated` have all had their initial "i" lower-cased, as the upper case "I" was an error.
      · `getPropertyDefault` and `isPropertySupported` are no longer available on the `Converter` object, they must be accessed via the `format` attribute:

```
converter.format.getPropertyDefault(...)
```

    - *canlib.kvlclib.WriterFormat.getFirstWriterFormat* and *canlib.kvlclib.WriterFormat.getNextWriterFormat* now returns a `kvlclib.FileFormat` object (which is based on the `IntEnum` class).
    - Other functions have been moved to the `canlib.kvlclib` module.
    - `deleteConverter` is no longer supported. Instead, converters are automatically deleted when garbage collected. If their contents must be flushed to file, see the new *canlib.kvlclib.Converter.flush* method.
  - The class `canlib.kvlclib.KvlcVersion` has been removed, and `canlib.kvmlib.kvlclib.getVersion` now returns a *canlib.VersionNumber*. The new class still supports conversion to string and accessing `.major`, `.minor`, and `.build` attributes.
- `canlib.kvrlib`:
  - The `canlib.kvrlib.KvrLib` object has been deprecated; all methods have been moved to the module.
  - `canlib.kvrlib.getVersion` no longer returns a *canlib.kvrlib.kvrVersion* but a *canlib.VersionNumber*. The return value still supports conversion to string and accessing `.major` and `.minor` attributes.
- `canlib.kvamemolibxml`:
  - Renamed from `canlib.KvaMemoLibXml`, however trying to import the old name will just import the new one instead.
  - Deprecated the use of `canlib.kvamemolibxml.KvaMemoLib` objects, all methods have been moved to the `canlib.kvamemolibxml` module itself.
  - Breaking change: Moved values that were incorrectly defined as constants into enums. In most cases this should not have any impact, as all the values are internal error codes and are turned into Python exceptions. But if you nonetheless use the kvamemolibxml status values directly, you'll need to change your code as follows:
    - `KvaXmlStatusERR_XXX_XXX` is now `Error.XXX_XXX`.
    - `KvaXmlValidationStatusERR_XXX_XXX` is now `ValidationError.XXX_XXX`
    - `KvaXmlValidationStatusWARN` is now `ValidationWarning.XXX_XXX`.
    - `KvaXmlStatusFail` is now `Error.FAIL` (Changed to be consistent with other KvaXmlStatus errors). The same is true for `ValidationError.FAIL`.
    - `KvaXmlStatusOK` and `KvaXmlValidationStatusOK` are still treated as if they are constants, as they are not error statuses.
  - `canlib.kvamemolibxml.getVersion` no longer returns a string but a *canlib.VersionNumber*. The return value still supports conversion to string.
- Exceptions:
  - Exceptions throughout the package have been standardised, and now all inherit from *canlib.exceptions.CanlibException*.
  - The `canERR` attribute that some exceptions had has been deprecated in favour of a `status` attribute. Furthermore, all `canlib` exceptions now have this attribute; the status code that was returned from a C call that triggered the specific exception.

2.1.19 New Features and Fixed Problems in V1.4.373 (13-SEP-2017)

- Minor changes.

2.1.20 New Features and Fixed Problems in V1.3.242 (05-MAY-2017)

- Added missing unicode conversions for Python3.
- Linux
  - Added support for new libraries (kvadblib, kvmlib, kvamemolibxml, kvlclib).
  - Added wrappers KvFileGetCount, kvFileGetName, kvFileCopyXxxx, kvDeviceSetMode, kvDeviceGetMode
- canlib:
  - Added wrapper for kvFileDelete
  - Enhanced printout from canScriptFail errors.
  - Second file argument in fileCopyFromDevice and fileCopyToDevice is now optional.
  - OS now loads all dependency dll (also adding KVDLLPATH to PATH in Windows).

2.1.21 New Features and Fixed Problems in V1.2.163 (15-FEB-2017)

- Added wrapper function canlib.getChannelData_Cust_Name()
- Added module canlib.kvlclib which is a wrapper for the Converter Library kvlclib in CANlib SDK.
- Added wrapper function canChannel.flashLeds().
- Added missing unicode conversions for Python3.
- Fixed bug where CANlib SDK install directory was not always correctly detected in Windows 10.

2.1.22 New Features and Fixed Problems in V1.1.23 (28-SEP-2016)

- canSetAcceptanceFilter and kvReadTimer was not implemented in Linux

2.1.23 New Features and Fixed Problems in V1.0.10 (15-SEP-2016)

- Initial module release.
- Added kvmlib.kmeSCanFileType()
- Added canChannel.canAccept() and canChannel.canSetAcceptanceFilter()

PYTHON MODULE INDEX

C

canlib.canlib.envvar, 102
canlib.canlib.iopin, 129
canlib.canlib.objbuf, 138
canlib.j1939, 237
canlib.kvrlib.service, 222

INDEX

A

ABL (*canlib.canlib.MessageFlag attribute*), 119
ABORT (*canlib.kvamemolibxml.ValidationError attribute*), 167
ABORT (*canlib.kvamemolibxml.ValidationWarning attribute*), 168
ACCEPT_LARGE_DLC (*canlib.canlib.Open attribute*), 121
ACCEPT_VIRTUAL (*canlib.canlib.Open attribute*), 121
AcceptFilterFlag (*class in canlib.canlib*), 103
accessibility (*canlib.kvrlib.DeviceInfo property*), 199
accessibility (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
Accessibility (*class in canlib.kvrlib*), 203
accessibility_pwd (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
ACQUISITOR (*canlib.canlib.HardwareType attribute*), 113
active_profile (*canlib.kvrlib.RemoteDevice property*), 217
add() (*canlib.kvrlib.DeviceInfoSet method*), 211
add_enum_definition() (*canlib.kvadblib.EnumDefinition method*), 146
add_enum_definition() (*canlib.kvadblib.EnumSignal method*), 160
add_message() (*canlib.kvadblib.FrameBox method*), 156
add_node() (*canlib.kvadblib.Signal method*), 161
addDatabaseFile() (*canlib.kvlclib.Converter method*), 172
AddonModule (*class in canlib.canlib.iopin*), 129
address (*canlib.kvrlib.kvrAddress attribute*), 220
Address (*class in canlib.kvrlib*), 198
address1 (*canlib.kvrlib.AddressInfo property*), 212
address2 (*canlib.kvrlib.AddressInfo property*), 212
address_info (*canlib.kvrlib.RemoteDevice property*), 218
addresses (*canlib.kvrlib.Discovery property*), 201
AddressInfo (*class in canlib.kvrlib*), 212
AddressType (*class in canlib.kvrlib*), 204
AddressTypeFlag (*class in canlib.kvrlib*), 204
ADH (*canlib.canlib.OperationalMode attribute*), 123
AFDX (*canlib.kvadblib.ProtocolType attribute*), 154
AI_HYSTERESIS (*canlib.canlib.iopin.Info attribute*), 133
AI_LP_FILTER_ORDER (*canlib.canlib.iopin.Info attribute*), 133
ALL (*canlib.kvlclib.ChannelMask attribute*), 175
ALL (*canlib.kvmlib.enums.LogFileType attribute*), 186
ALL (*canlib.kvrlib.AddressTypeFlag attribute*), 204
ALL_LEDS_OFF (*canlib.canlib.LEDAction attribute*), 117
ALL_LEDS_ON (*canlib.canlib.LEDAction attribute*), 117
ALL_SLOTS (*canlib.canlib.ScriptRequest attribute*), 123
allocate_periodic_objbuf() (*canlib.canlib.Channel method*), 82
allocate_response_objbuf() (*canlib.canlib.Channel method*), 83
ANALOG (*canlib.canlib.iopin.ModuleType attribute*), 135
ANALOG (*canlib.canlib.iopin.PinType attribute*), 136
AnalogIn (*class in canlib.canlib.iopin*), 130
AnalogOut (*class in canlib.canlib.iopin*), 130
announceIdentityEx() (*canlib.canlib.Channel method*), 83
ANY (*canlib.kvrlib.BasicServiceSet attribute*), 205
app_version (*canlib.linlib.FirmwareVersion property*), 231
asframe() (*canlib.kvadblib.Message method*), 158
asframe() (*canlib.kvmlib.MessageEvent method*), 187
attachFile() (*canlib.kvlclib.Converter method*), 172
ATTACHMENTS (*canlib.kvlclib.Property attribute*), 178
ATTR_NOT_FOUND (*canlib.kvamemolibxml.Error attribute*), 166
ATTR_VALUE (*canlib.kvamemolibxml.Error attribute*), 166
Attribute (*class in canlib.kvadblib*), 145
attribute_definitions() (*canlib.kvadblib.Dbc method*), 148
AttributeDefinition (*class in canlib.kvadblib*), 145
AttributeOwner (*class in canlib.kvadblib*), 152
attributes() (*canlib.kvadblib.Dbc method*), 148
attributes() (*canlib.kvadblib.Message method*), 158
attributes() (*canlib.kvadblib.Node method*), 160
attributes() (*canlib.kvadblib.Signal method*), 161
AttributeType (*class in canlib.kvadblib*), 153
AUTHENTICATING (*canlib.kvrlib.NetworkState attribute*), 207

AUTHENTICATION_FAILED (*canlib.kvrlib.NetworkState attribute*), 207
availability (*canlib.kvrlib.DeviceInfo property*), 199
availability (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
Availability (*class in canlib.kvrlib*), 204
AVAILABLE (*canlib.kvrlib.RemoteState attribute*), 208
AVAILABLE (*canlib.kvrlib.ServiceState attribute*), 209

B

BAGEL (*canlib.canlib.HardwareType attribute*), 113
base_station_id (*canlib.kvrlib.DeviceInfo property*), 199
base_station_id (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
BasicServiceSet (*class in canlib.kvrlib*), 205
bcd() (*canlib.EAN method*), 67
BEAN (*canlib.kvadblib.ProtocolType attribute*), 154
beta (*canlib.BetaVersionNumber property*), 72
beta (*canlib.VersionNumber property*), 71
BetaVersionNumber (*class in canlib*), 72
bind() (*canlib.kvadblib.Message method*), 158
bind() (*canlib.kvadblib.Signal method*), 161
BIT (*canlib.canlib.MessageFlag attribute*), 119
BIT0 (*canlib.canlib.MessageFlag attribute*), 119
BIT1 (*canlib.canlib.MessageFlag attribute*), 119
BIT_ERROR (*canlib.linlib.MessageFlag attribute*), 233
bitrate (*canlib.linlib.MessageInfo attribute*), 235
Bitrate (*class in canlib.canlib*), 103
bitrate() (*canlib.canlib.busparams.BusParamsTq method*), 80
BITRATE_100K (*canlib.canlib.Bitrate attribute*), 103
BITRATE_10K (*canlib.canlib.Bitrate attribute*), 103
BITRATE_125K (*canlib.canlib.Bitrate attribute*), 103
BITRATE_1M (*canlib.canlib.Bitrate attribute*), 103
BITRATE_1M_80P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_250K (*canlib.canlib.Bitrate attribute*), 103
BITRATE_2M_60P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_2M_80P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_4M_80P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_500K (*canlib.canlib.Bitrate attribute*), 103
BITRATE_500K_80P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_50K (*canlib.canlib.Bitrate attribute*), 104
BITRATE_62K (*canlib.canlib.Bitrate attribute*), 104
BITRATE_83K (*canlib.canlib.Bitrate attribute*), 104
BITRATE_8M_60P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_8M_70P (*canlib.canlib.BitrateFD attribute*), 104
BITRATE_8M_80P (*canlib.canlib.BitrateFD attribute*), 104
bitrate_to_BusParamsTq() (*canlib.canlib.Channel method*), 83
BitrateFD (*class in canlib.canlib*), 104
BitrateSetting (*class in canlib.canlib.busparams*), 81
BLACKBIRD (*canlib.canlib.HardwareType attribute*), 113
BLACKBIRD_V2 (*canlib.canlib.HardwareType attribute*), 113
BLANK (*canlib.kvrlib.Error attribute*), 206
boot_version (*canlib.linlib.FirmwareVersion property*), 231
BoundMessage (*class in canlib.kvadblib*), 157
BoundSignal (*class in canlib.kvadblib*), 157
BROADCAST (*canlib.kvrlib.AddressTypeFlag attribute*), 204
BRS (*canlib.canlib.MessageFlag attribute*), 119
bss_type (*canlib.kvrlib.WlanScanResult property*), 220
BUFFER_SIZE (*canlib.kvlclib.Error attribute*), 175
BUFFER_TOO_SMALL (*canlib.canlib.Error attribute*), 110
BUFFER_TOO_SMALL (*canlib.kvadblib.Error attribute*), 153
BUFFER_TOO_SMALL (*canlib.kvrlib.Error attribute*), 206
build (*canlib.VersionNumber property*), 71
BUS_OFF (*canlib.canlib.Stat attribute*), 124
bus_param_limits (*canlib.canlib.ChannelData property*), 99
BUS_PARAM_LIMITS (*canlib.canlib.ChannelDataItem attribute*), 106
BUS_STATISTICS (*canlib.canlib.ChannelCap attribute*), 105
BUS_TYPE (*canlib.canlib.ChannelDataItem attribute*), 106
BUSERR (*canlib.canlib.MessageFlag attribute*), 119
busOff() (*canlib.canlib.Channel method*), 83
busOff() (*canlib.linlib.Channel method*), 227
busOn() (*canlib.canlib.Channel method*), 83
busOn() (*canlib.linlib.Channel method*), 227
BUSONOFF (*canlib.canlib.Notify attribute*), 121
BusParamsTq (*class in canlib.canlib.busparams*), 79
BusParamTqLimits (*class in canlib.canlib.busparams*), 80
BusTypeGroup (*class in canlib.canlib*), 104
byte_order (*canlib.kvadblib.Signal property*), 161
bytes_to_dlc() (*in module canlib.kvadblib*), 163
byteTime (*canlib.linlib.MessageInfo attribute*), 235

C

calc_bitrate() (*in module canlib.canlib.busparams*), 76
calc_busparamstq() (*in module canlib.canlib.busparams*), 76
calc_sjw() (*in module canlib.canlib.busparams*), 77

CALENDAR_TIME_STAMPS (*canlib.kvlclib.Property attribute*), 178
CAN (*canlib.kvadblib.ProtocolType attribute*), 154
CAN_FD (*canlib.canlib.ChannelCap attribute*), 105
CAN_FD (*canlib.canlib.Open attribute*), 122
CAN_FD_NONISO (*canlib.canlib.ChannelCap attribute*), 105
CAN_FD_NONISO (*canlib.canlib.Open attribute*), 122
can_id_from_pdu() (*in module canlib.j1939*), 239
canAccept() (*canlib.canlib.Channel method*), 84
CanBusStatistics (*class in canlib.canlib.structures*), 97
canERR (*canlib.DllException property*), 65
CanError, 72
CANERROR (*canlib.linlib.Error attribute*), 232
CANFD (*canlib.canlib.TransceiverType attribute*), 125
CANFD (*canlib.kvadblib.ProtocolType attribute*), 154
CANFD_LIN (*canlib.canlib.TransceiverType attribute*), 125
canflags (*canlib.kvadblib.Message property*), 158
CanGeneralError, 72
CanInvalidHandle, 73
canlib.canlib.envvar
    module, 102
canlib.canlib.iopin
    module, 129
canlib.canlib.objbuf
    module, 138
canlib.j1939
    module, 237
canlib.kvrlib.service
    module, 222
CanlibException, 65
CANLINHYBRID (*canlib.canlib.HardwareType attribute*), 113
CanNoMsg, 73
CanNotFound, 73
CanOutOfMemory, 73
CANPARI (*canlib.canlib.HardwareType attribute*), 113
CanScriptFail, 74
canSetAcceptanceFilter() (*canlib.canlib.Channel method*), 84
CanTimeout, 74
CARD_FIRMWARE_REV (*canlib.canlib.ChannelDataItem attribute*), 106
CARD_FIRMWARE_REV (*canlib.linlib.ChannelData attribute*), 231
CARD_HARDWARE_REV (*canlib.canlib.ChannelDataItem attribute*), 106
CARD_NUMBER (*canlib.canlib.ChannelDataItem attribute*), 106
CARD_SERIAL_NO (*canlib.canlib.ChannelDataItem attribute*), 106
CARD_TYPE (*canlib.canlib.ChannelDataItem attribute*), 106
CARD_UPC_NO (*canlib.canlib.ChannelDataItem attribute*), 106
CHAN_NO_ON_CARD (*canlib.canlib.ChannelDataItem attribute*), 107
Channel (*class in canlib.canlib*), 82
Channel (*class in canlib.linlib*), 227
channel() (*canlib.Device method*), 68
CHANNEL_CAP (*canlib.canlib.ChannelDataItem attribute*), 106
CHANNEL_CAP_EX (*canlib.canlib.ChannelDataItem attribute*), 106
CHANNEL_CAP_MASK (*canlib.canlib.ChannelDataItem attribute*), 106
channel_data (*canlib.canlib.Channel property*), 84
channel_data() (*canlib.Device method*), 68
CHANNEL_FLAGS (*canlib.canlib.ChannelDataItem attribute*), 107
CHANNEL_MASK (*canlib.kvlclib.Property attribute*), 178
channel_name (*canlib.canlib.ChannelData property*), 99
CHANNEL_NAME (*canlib.canlib.ChannelDataItem attribute*), 107
channel_number (*canlib.kvrlib.ConfigProfile property*), 213
channel_number (*canlib.kvrlib.ConnectionStatus property*), 214
channel_number (*canlib.kvrlib.WlanScanResult property*), 220
channel_number() (*canlib.Device method*), 68
CHANNEL_QUALITY (*canlib.canlib.ChannelDataItem attribute*), 107
ChannelCap (*class in canlib.canlib*), 105
ChannelData (*class in canlib.canlib*), 97
ChannelData (*class in canlib.linlib*), 231
ChannelDataItem (*class in canlib.canlib*), 106
ChannelFlags (*class in canlib.canlib*), 109
ChannelMask (*class in canlib.kvlclib*), 175
ChannelType (*class in canlib.linlib*), 231
CHECKSUM (*canlib.kvrlib.Error attribute*), 206
checkSum (*canlib.linlib.MessageInfo attribute*), 235
CHINA_MII (*canlib.kvrlib.RegulatoryDomain attribute*), 207
clear() (*canlib.kvrlib.ConfigProfile method*), 213
clear() (*canlib.kvrlib.kvrConfig method*), 225
CLEAR_ERROR_COUNTERS (*canlib.canlib.IOControlItem attribute*), 115
clearDevicesAtExit() (*canlib.kvrlib.kvrDiscovery method*), 224
clearMessage() (*canlib.linlib.Channel method*), 227
client_address (*canlib.kvrlib.DeviceInfo property*), 199
client_address (*canlib.kvrlib.kvrDeviceInfo attribute*),

221
CLOCK_INFO (*canlib.canlib.ChannelDataItem attribute*), 107
ClockInfo (*class in canlib.canlib.busparams*), 79
close() (*canlib.canlib.Channel method*), 84
close() (*canlib.kvadblib.Dbc method*), 148
close() (*canlib.kvmlib.Kme method*), 191
close() (*canlib.kvmlib.KmfSystem method*), 193
close() (*canlib.kvrlib.ConnectionTest method*), 215
close() (*canlib.kvrlib.Discovery method*), 201
close() (*canlib.kvrlib.kvrConfig method*), 225
close() (*canlib.kvrlib.kvrDiscovery method*), 224
close() (*canlib.kvrlib.WlanScan method*), 219
close() (*canlib.linlib.Channel method*), 227
CLOSING (*canlib.kvrlib.RemoteState attribute*), 208
CLOSING (*canlib.kvrlib.ServiceState attribute*), 209
code (*canlib.kvamemolibxml.ValidationMessage property*), 165
comment (*canlib.kvadblib.Message property*), 158
comment (*canlib.kvadblib.Node property*), 160
comment (*canlib.kvadblib.Signal property*), 161
compiler_version (*canlib.canlib.Txe property*), 136
COMPILER_VERSION (*canlib.canlib.TxeDataItem attribute*), 126
COMPRESSION_LEVEL (*canlib.kvlclib.Property attribute*), 178
CONFIG (*canlib.canlib.Error attribute*), 110
CONFIG (*canlib.kvrlib.DeviceUsage attribute*), 205
CONFIG_CHANGED (*canlib.kvrlib.RemoteState attribute*), 208
CONFIG_CHANGED (*canlib.kvrlib.ServiceState attribute*), 209
CONFIG_ERROR (*canlib.kvmlib.Error attribute*), 184
CONFIG_VERSION_NEEDED (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 186
config_version_needed (*canlib.kvmlib.Memorator property*), 195
configActiveProfileGet() (*in module canlib.kvrlib*), 223
configActiveProfileSet() (*in module canlib.kvrlib*), 223
ConfigMode (*class in canlib.kvrlib*), 205
configNoProfilesGet() (*in module canlib.kvrlib*), 223
ConfigProfile (*class in canlib.kvrlib*), 213
Configuration (*class in canlib.canlib.iopin*), 130
Configuration (*class in canlib.kvamemolibxml*), 164
confirm() (*canlib.canlib.iopin.Configuration method*), 131
connect (*canlib.kvrlib.DeviceInfo property*), 199
connect() (*canlib.kvrlib.kvrDeviceInfo method*), 221
CONNECT_TO_VIRTUAL_BUS (*canlib.canlib.IOControlItem attribute*), 115
CONNECTED (*canlib.kvrlib.NetworkState attribute*), 207
connected_devices() (*in module canlib*), 70
CONNECTING (*canlib.kvrlib.NetworkState attribute*), 207
CONNECTION_DELAY (*canlib.kvrlib.NetworkState attribute*), 207
CONNECTION_DOWN (*canlib.kvrlib.RemoteState attribute*), 208
CONNECTION_DOWN (*canlib.kvrlib.ServiceState attribute*), 209
connection_status (*canlib.kvrlib.RemoteDevice property*), 218
connection_test() (*canlib.kvrlib.RemoteDevice method*), 218
CONNECTION_UP (*canlib.kvrlib.RemoteState attribute*), 208
CONNECTION_UP (*canlib.kvrlib.ServiceState attribute*), 209
ConnectionStatus (*class in canlib.kvrlib*), 214
ConnectionTest (*class in canlib.kvrlib*), 215
ConnectionTestResult (*class in canlib.kvrlib*), 215
contents (*canlib.canlib.SourceElement property*), 136
Converter (*class in canlib.kvlclib*), 172
convertEvent() (*canlib.kvlclib.Converter method*), 172
CONVERTING (*canlib.kvlclib.Error attribute*), 175
CPLD_VERSION (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 186
CRC (*canlib.canlib.Error attribute*), 110
CRC (*canlib.canlib.MessageFlag attribute*), 119
CRC_ERROR (*canlib.kvmlib.Error attribute*), 184
createKme() (*in module canlib.kvmlib*), 190
createMemoEvent() (*canlib.kvmlib.memoLogEventEx method*), 188
CREATION_DATE (*canlib.kvlclib.Property attribute*), 178
creator_serial (*canlib.kvmlib.LogFile property*), 190
CROP_PRETRIGGER (*canlib.kvlclib.Property attribute*), 178
CSUM (*canlib.linlib.MessageDisturb attribute*), 233
CSUM_ERROR (*canlib.linlib.MessageFlag attribute*), 233
CSV (*canlib.kvlclib.FileFormat attribute*), 176
CSV_SIGNAL (*canlib.kvlclib.FileFormat attribute*), 176
CUST_CHANNEL_NAME (*canlib.canlib.ChannelDataItem attribute*), 107
custom_name (*canlib.canlib.ChannelData property*), 99

D

da (*canlib.j1939.Pdu1 attribute*), 238
data (*canlib.Frame attribute*), 70
data (*canlib.j1939.Pdu attribute*), 238
data (*canlib.LINFrame attribute*), 71
data_from() (*canlib.kvadblib.Signal method*), 161
DATA_IN_HEX (*canlib.kvlclib.Property attribute*), 178
DATABASE_FLAG_J1939 (*canlib.kvadblib.dbc attribute*), 148
DATABASE_INTERNAL (*canlib.kvadblib.Error attribute*), 153

DataEnvVar (*class in canlib.canlib.envvar*), 102
date (*canlib.canlib.Txe property*), 136
DATE (*canlib.canlib.TxeDataItem attribute*), 126
DB (*canlib.kvadblib.AttributeOwner attribute*), 152
DB_FILE_OPEN (*canlib.kvadblib.Error attribute*), 153
DB_FILE_PARSE (*canlib.kvadblib.Error attribute*), 153
Dbc (*class in canlib.kvadblib*), 148
DEBUG (*canlib.kvlclib.FileFormat attribute*), 176
DECIMAL_CHAR (*canlib.kvlclib.Property attribute*), 178
default (*canlib.kvadblib.DefaultDefinition property*), 145
default (*canlib.kvadblib.EnumDefaultDefinition property*), 146
default (*canlib.kvadblib.MinMaxDefinition property*), 147
DefaultDefinition (*class in canlib.kvadblib*), 145
definition (*canlib.kvadblib.EnumDefinition property*), 146
definition (*canlib.kvadblib.FloatDefinition property*), 146
definition (*canlib.kvadblib.IntegerDefinition property*), 147
definition (*canlib.kvadblib.StringDefinition property*), 147
delete_all() (*canlib.kvmlib.UnmountedLog method*), 189
delete_attribute() (*canlib.kvadblib.Dbc method*), 148
delete_attribute() (*canlib.kvadblib.Message method*), 158
delete_attribute() (*canlib.kvadblib.Node method*), 160
delete_attribute() (*canlib.kvadblib.Signal method*), 161
delete_attribute_definition() (*canlib.kvadblib.Dbc method*), 148
delete_message() (*canlib.kvadblib.Dbc method*), 149
delete_node() (*canlib.kvadblib.Dbc method*), 149
delete_signal() (*canlib.kvadblib.Message method*), 158
description (*canlib.canlib.Txe property*), 136
DESCRIPTION (*canlib.canlib.TxeDataItem attribute*), 126
DEVDESCR_ASCII (*canlib.canlib.ChannelDataItem attribute*), 107
DEVDESCR_UNICODE (*canlib.canlib.ChannelDataItem attribute*), 107
Device (*class in canlib*), 67
Device (*class in canlib.kvmlib*), 183
device() (*canlib.canlib.Channel method*), 85
device_address (*canlib.kvrlib.DeviceInfo property*), 199
device_address (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
DEVICE_COMM_ERROR (*canlib.kvmlib.Error attribute*), 184
DEVICE_FILE (*canlib.canlib.Error attribute*), 110
device_name (*canlib.canlib.ChannelData property*), 99
DEVICE_PHYSICAL_POSITION (*canlib.canlib.ChannelDataItem attribute*), 107
deviceGetServiceStatus() (*in module canlib.kvrlib*), 223
deviceGetServiceStatusText() (*in module canlib.kvrlib*), 223
DeviceInfo (*class in canlib.kvrlib*), 199
DeviceInfoSet (*class in canlib.kvrlib*), 211
DeviceMode (*class in canlib.canlib*), 109
DeviceNotInSetError, 198
DeviceUsage (*class in canlib.kvrlib*), 205
DEVNAME_ASCII (*canlib.canlib.ChannelDataItem attribute*), 107
DI_HIGH_LOW_FILTER (*canlib.canlib.iopin.Info attribute*), 133
DI_LOW_HIGH_FILTER (*canlib.canlib.iopin.Info attribute*), 133
DIADEM (*canlib.kvlclib.FileFormat attribute*), 177
DIAGNOSTICS (*canlib.canlib.ChannelCap attribute*), 105
DIGITAL (*canlib.canlib.iopin.ModuleType attribute*), 135
DIGITAL (*canlib.canlib.iopin.PinType attribute*), 136
DigitalIn (*class in canlib.canlib.iopin*), 132
DigitalOut (*class in canlib.canlib.iopin*), 133
DigitalValue (*class in canlib.canlib.iopin*), 133
DINRAIL (*canlib.canlib.HardwareType attribute*), 113
DIRECTION (*canlib.canlib.iopin.Info attribute*), 133
direction (*canlib.canlib.iopin.IoPin property*), 134
Direction (*class in canlib.canlib.iopin*), 133
disable() (*canlib.canlib.objbuf.Periodic method*), 138
disable() (*canlib.canlib.objbuf.Response method*), 139
discard() (*canlib.kvrlib.DeviceInfoSet method*), 211
disconnect() (*canlib.kvrlib.kvrDeviceInfo method*), 221
DISCONNECT_FROM_VIRTUAL_BUS (*canlib.canlib.IOControlItem attribute*), 115
discover_info_set() (*in module canlib.kvrlib*), 210
DISCOVERED (*canlib.kvrlib.RemoteState attribute*), 208
DISCOVERED (*canlib.kvrlib.ServiceState attribute*), 209
Discovery (*class in canlib.kvrlib*), 201
DISK (*canlib.canlib.Error attribute*), 110
DISK_ERROR (*canlib.kvmlib.Error attribute*), 184
DISK_FULL_AND_FIFO (*canlib.kvamemolibxml.ValidationWarning attribute*), 168
DISK_FULL_STARTS_LOG (*canlib.kvamemolibxml.ValidationError attribute*), 167
disk_size (*canlib.kvmlib.Memorator property*), 195
disk_usage (*canlib.kvmlib.KmfSystem property*), 193
DISKFULL_DATA (*canlib.kvmlib.Error attribute*), 184
DISKFULL_DIR (*canlib.kvmlib.Error attribute*), 184

dlc (*canlib.Frame attribute*), 70
dlc (*canlib.kvadblib.Message property*), 158
dlc (*canlib.LINFrame attribute*), 71
dlc_to_bytes() (*in module canlib.kvadblib*), 163
DLL_FILE_VERSION (*canlib.canlib.ChannelDataItem attribute*), 107
DLL_FILETYPE (*canlib.canlib.ChannelDataItem attribute*), 107
DLL_PRODUCT_VERSION (*canlib.canlib.ChannelDataItem attribute*), 107
DllException, 65
dllversion() (*in module canlib.canlib*), 140
dllversion() (*in module canlib.kvadblib*), 163
dllversion() (*in module canlib.kvamemolibxml*), 168
dllversion() (*in module canlib.kvlclib*), 182
dllversion() (*in module canlib.kvmlib*), 197
dllversion() (*in module canlib.kvrlib*), 223
dllversion() (*in module canlib.linlib*), 235
DNOPTO (*canlib.canlib.TransceiverType attribute*), 125
DOUBLE (*canlib.kvadblib.SignalType attribute*), 155
dp (*canlib.j1939.Pdu attribute*), 238
DRIVER (*canlib.canlib.Error attribute*), 110
DRIVER (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 186
DRIVER (*canlib.linlib.Error attribute*), 232
Driver (*class in canlib.canlib*), 109
DRIVER_FILE_VERSION (*canlib.canlib.ChannelDataItem attribute*), 107
DRIVER_NAME (*canlib.canlib.ChannelDataItem attribute*), 107
DRIVER_PRODUCT (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 186
DRIVER_PRODUCT_VERSION (*canlib.canlib.ChannelDataItem attribute*), 107
driver_version (*canlib.kvmlib.Memorator property*), 195
DriverCap (*class in canlib.canlib*), 110
DRIVERFAILED (*canlib.canlib.Error attribute*), 110
DRIVERFAILED (*canlib.linlib.Error attribute*), 232
DRIVERLOAD (*canlib.canlib.Error attribute*), 110
DTD_VALIDATION (*canlib.kvamemolibxml.Error attribute*), 166
DUPLICATED_DEVICE (*canlib.kvrlib.Error attribute*), 206
DYNAINIT (*canlib.canlib.Error attribute*), 110
DYNALIB (*canlib.canlib.Error attribute*), 110
DYNALOAD (*canlib.canlib.Error attribute*), 111

E

EAGLE (*canlib.canlib.HardwareType attribute*), 113
ean (*canlib.Device attribute*), 68
ean (*canlib.kvrlib.DeviceInfo property*), 199
ean (*canlib.linlib.TransceiverData property*), 237
EAN (*class in canlib*), 66
ean_hi (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
ean_lo (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
EDL (*canlib.canlib.MessageFlag attribute*), 120
edp (*canlib.j1939.Pdu attribute*), 238
ELEM_NOT_FOUND (*canlib.kvamemolibxml.Error attribute*), 166
ELEMENT_COUNT (*canlib.kvamemolibxml.ValidationError attribute*), 167
elements (*canlib.kvrlib.kvrAddressList attribute*), 221
elements (*canlib.kvrlib.kvrDeviceInfoList attribute*), 222
empty_info_set() (*in module canlib.kvrlib*), 210
enable() (*canlib.canlib.objbuf.Periodic method*), 138
enable() (*canlib.canlib.objbuf.Response method*), 139
encryption_key (*canlib.kvrlib.DeviceInfo property*), 199
encryption_key (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
end_time (*canlib.kvmlib.LogFile property*), 190
ENHANCED (*canlib.linlib.MessageParity attribute*), 234
ENHANCED_CHECKSUM (*canlib.linlib.Setup attribute*), 234
ENUM (*canlib.kvadblib.AttributeType attribute*), 153
ENUM_SIGNED (*canlib.kvadblib.SignalType attribute*), 155
ENUM_UNSIGNED (*canlib.kvadblib.SignalType attribute*), 155
ENUM_VALUES (*canlib.kvlclib.Property attribute*), 178
EnumDefaultDefinition (*class in canlib.kvadblib*), 146
EnumDefinition (*class in canlib.kvadblib*), 146
enumerate_hardware() (*in module canlib.canlib*), 142
enums (*canlib.kvadblib.EnumDefaultDefinition property*), 146
enums (*canlib.kvadblib.EnumSignal property*), 160
EnumSignal (*class in canlib.kvadblib*), 160
ENV (*canlib.kvadblib.AttributeOwner attribute*), 152
ENVVAR (*canlib.canlib.Notify attribute*), 121
EnvVar (*class in canlib.canlib*), 102
EnvvarException, 74
EnvvarNameError, 74
EnvVarType (*class in canlib.canlib*), 110
EnvvarValueError, 75
EOF (*canlib.kvlclib.Error attribute*), 175
EOF (*canlib.kvmlib.Error attribute*), 184
ERASE (*canlib.kvrlib.ConfigMode attribute*), 205
ERR (*canlib.kvmlib.enums.LogFileType attribute*), 186
ERR_CONFIGURING (*canlib.kvrlib.StartInfo attribute*), 210
ERR_ENCRYPTION_PWD (*canlib.kvrlib.StartInfo attribute*), 210
ERR_IN_USE (*canlib.kvrlib.StartInfo attribute*), 210
ERR_NOTME (*canlib.kvrlib.StartInfo attribute*), 210
ERR_PARAM (*canlib.kvrlib.StartInfo attribute*), 210
ERR_PWD (*canlib.kvrlib.StartInfo attribute*), 210
ERROR (*canlib.canlib.Notify attribute*), 121

Error (*class in canlib.canlib*), 110
Error (*class in canlib.kvadblib*), 153
Error (*class in canlib.kvamemolibxml*), 166
Error (*class in canlib.kvlclib*), 175
Error (*class in canlib.kvmlib*), 184
Error (*class in canlib.kvrlib*), 206
Error (*class in canlib.linlib*), 232
ERROR_ACTIVE (*canlib.canlib.Stat attribute*), 124
ERROR_COUNTERS (*canlib.canlib.ChannelCap attribute*), 105
ERROR_FRAME (*canlib.canlib.MessageFlag attribute*), 120
ERROR_PASSIVE (*canlib.canlib.Stat attribute*), 124
ERROR_WARNING (*canlib.canlib.Stat attribute*), 124
ErrorCounters (*class in canlib.canlib.channel*), 82
ERRRESP (*canlib.linlib.Error attribute*), 232
ESI (*canlib.canlib.MessageFlag attribute*), 120
estimate_events() (*canlib.kvmlib.Kme method*), 191
ETHERCAN (*canlib.canlib.HardwareType attribute*), 113
ETHERNET (*canlib.kvadblib.ProtocolType attribute*), 154
EUROPE_ETSI (*canlib.kvrlib.RegulatoryDomain attribute*), 207
EVA (*canlib.canlib.TransceiverType attribute*), 125
event (*canlib.kvmlib.memoLogEventEx attribute*), 188
event_count_estimation() (*canlib.kvmlib.Kme method*), 192
event_count_estimation() (*canlib.kvmlib.LogFile method*), 190
eventCount() (*canlib.kvlclib.Converter method*), 173
events() (*canlib.kvmlib.Kme method*), 192
EXCLUSIVE (*canlib.canlib.Open attribute*), 122
EXPRESSION (*canlib.kvamemolibxml.ValidationError attribute*), 167
EXT (*canlib.canlib.MessageFlag attribute*), 120
EXT (*canlib.kvadblib.MessageFlag attribute*), 154
EXTENDED_CAN (*canlib.canlib.ChannelCap attribute*), 105

F

factor (*canlib.kvadblib.ValueScaling property*), 163
FAIL (*canlib.kvadblib.Error attribute*), 153
FAIL (*canlib.kvamemolibxml.Error attribute*), 166
FAIL (*canlib.kvamemolibxml.ValidationError attribute*), 167
FAIL (*canlib.kvlclib.Error attribute*), 175
FAIL (*canlib.kvmlib.Error attribute*), 184
FAILED_MIC (*canlib.kvrlib.NetworkState attribute*), 207
FAMOS (*canlib.kvlclib.FileFormat attribute*), 177
FAMOS_XCP (*canlib.kvlclib.FileFormat attribute*), 177
FATAL_ERROR (*canlib.kvmlib.Error attribute*), 184
FDF (*canlib.canlib.MessageFlag attribute*), 120
FDMSG_MASK (*canlib.canlib.MessageFlag attribute*), 120
FEATURE_EAN (*canlib.canlib.ChannelDataItem attribute*), 107
feedLogEvent() (*canlib.kvlclib.Converter method*), 173
feedNextFile() (*canlib.kvlclib.Converter method*), 173
FIBER (*canlib.canlib.TransceiverType attribute*), 125
FILE_ERROR (*canlib.kvlclib.Error attribute*), 175
FILE_ERROR (*canlib.kvmlib.Error attribute*), 184
FILE_EXISTS (*canlib.kvlclib.Error attribute*), 175
FILE_NOT_FOUND (*canlib.kvmlib.Error attribute*), 184
FILE_SYSTEM_CORRUPT (*canlib.kvmlib.Error attribute*), 184
FILE_TOO_LARGE (*canlib.kvlclib.Error attribute*), 175
file_version (*canlib.canlib.Txe property*), 137
FILE_VERSION (*canlib.canlib.TxeDataItem attribute*), 126
fileCopyFromDevice() (*canlib.canlib.Channel method*), 85
fileCopyToDevice() (*canlib.canlib.Channel method*), 85
fileDelete() (*canlib.canlib.Channel method*), 85
fileDiskFormat() (*canlib.canlib.Channel method*), 85
FileFormat (*class in canlib.kvlclib*), 176
fileGetCount() (*canlib.canlib.Channel method*), 85
fileGetName() (*canlib.canlib.Channel method*), 85
FileType (*class in canlib.kvmlib*), 185
FILL_BLANKS (*canlib.kvlclib.Property attribute*), 178
find() (*canlib.Device class method*), 68
find() (*canlib.kvrlib.DeviceInfoSet method*), 211
find_remove() (*canlib.kvrlib.DeviceInfoSet method*), 211
FIRMWARE (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 187
FIRMWARE (*canlib.kvmlib.Error attribute*), 184
firmware_version (*canlib.kvmlib.Memorator property*), 195
firmware_version (*canlib.kvrlib.DeviceInfo property*), 199
FirmwareVersion (*class in canlib.linlib*), 231
FIRST_TRIGGER (*canlib.kvlclib.Property attribute*), 178
FIVE (*canlib.kvlclib.ChannelMask attribute*), 175
flags (*canlib.Frame attribute*), 70
flags (*canlib.kvadblib.Dbc property*), 149
flags (*canlib.kvadblib.Message property*), 158
flags (*canlib.LINFrame attribute*), 71
flash_leds() (*canlib.kvmlib.Memorator method*), 195
flashLeds() (*canlib.canlib.Channel method*), 85
FLEXRAY (*canlib.kvadblib.ProtocolType attribute*), 154
FLOAT (*canlib.canlib.EnvVarType attribute*), 110
FLOAT (*canlib.kvadblib.AttributeType attribute*), 153
FLOAT (*canlib.kvadblib.SignalType attribute*), 155
FloatDefinition (*class in canlib.kvadblib*), 146
flush() (*canlib.kvlclib.Converter method*), 173
FLUSH_RX_BUFFER (*canlib.canlib.IOControlItem attribute*), 115
FLUSH_TX_BUFFER (*canlib.canlib.IOControlItem attribute*), 115
fmt (*canlib.EAN attribute*), 67

FORCED (*canlib.canlib.ScriptStop attribute*), 124
FORM (*canlib.canlib.MessageFlag attribute*), 120
format_disk() (*canlib.kvmlib.Memorator method*), 195
FOUND_BY_SCAN (*canlib.kvrlib.Availability attribute*), 204
FOUR (*canlib.kvlclib.ChannelMask attribute*), 175
Frame (*class in canlib*), 70
FrameBox (*class in canlib.kvadblib*), 156
frameLength (*canlib.linlib.MessageInfo attribute*), 235
frames() (*canlib.kvadblib.FrameBox method*), 156
FREE (*canlib.kvrlib.DeviceUsage attribute*), 205
free() (*canlib.canlib.objbuf.Periodic method*), 138
free() (*canlib.canlib.objbuf.Response method*), 139
free_objbuf() (*canlib.canlib.Channel method*), 86
frequency() (*canlib.canlib.busparams.ClockInfo method*), 79
from_bcd() (*canlib.EAN class method*), 67
from_c() (*canlib.kvrlib.Address class method*), 198
from_hilo() (*canlib.EAN class method*), 67
from_number() (*canlib.kvamemolibxml.Error class method*), 167
from_number() (*canlib.kvamemolibxml.ValidationError class method*), 168
from_number() (*canlib.kvamemolibxml.ValidationWarning class method*), 168
from_predefined() (*canlib.canlib.busparams.BitrateSetting class method*), 81
from_string() (*canlib.EAN class method*), 67
FULLY_QUALIFIED_NAMES (*canlib.kvlclib.Property attribute*), 178
fw_build_ver (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
fw_major_ver (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
fw_minor_ver (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
FW_VERSION (*canlib.canlib.iopin.Info attribute*), 134
fw_version (*canlib.canlib.iopin.IoPin property*), 134

G

gateway (*canlib.kvrlib.AddressInfo property*), 212
ge (*canlib.j1939.Pdu2 attribute*), 239
GENERATE_ERROR (*canlib.canlib.ChannelCap attribute*), 105
GENERATE_OVERLOAD (*canlib.canlib.ChannelCap attribute*), 105
generate_wep_keys() (*in module canlib.kvrlib*), 224
generate_wpa_keys() (*in module canlib.kvrlib*), 224
GENERIC (*canlib.kvrlib.Error attribute*), 206
get_attribute_definition_by_name() (*canlib.kvadblib.Dbc method*), 149
get_attribute_value() (*canlib.kvadblib.Dbc method*), 149
get_attribute_value() (*canlib.kvadblib.Message method*), 158
get_attribute_value() (*canlib.kvadblib.Node method*), 160
get_attribute_value() (*canlib.kvadblib.Signal method*), 161
get_bus_params_tq() (*canlib.canlib.Channel method*), 87
get_bus_statistics() (*canlib.canlib.Channel method*), 87
GET_BUS_TYPE (*canlib.canlib.IOControlItem attribute*), 115
get_can_channel() (*canlib.linlib.Channel method*), 228
GET_CHANNEL_QUALITY (*canlib.canlib.IOControlItem attribute*), 115
get_default_discovery_addresses() (*in module canlib.kvrlib*), 202
GET_DEVNAME_ASCII (*canlib.canlib.IOControlItem attribute*), 115
GET_DRIVERHANDLE (*canlib.canlib.IOControlItem attribute*), 115
GET_EVENTHANDLE (*canlib.canlib.IOControlItem attribute*), 115
get_io_pin() (*canlib.canlib.Channel method*), 87
get_last_parse_error() (*in module canlib.kvadblib*), 152
get_message() (*canlib.kvadblib.Dbc method*), 149
get_message_by_id() (*canlib.kvadblib.Dbc method*), 149
get_message_by_name() (*canlib.kvadblib.Dbc method*), 150
get_message_by_pgn() (*canlib.kvadblib.Dbc method*), 150
get_node_by_name() (*canlib.kvadblib.Dbc method*), 150
get_protocol_properties() (*in module canlib.kvadblib*), 163
GET_REPORT_ACCESS_ERRORS (*canlib.canlib.IOControlItem attribute*), 115
GET_ROUNDTRIP_TIME (*canlib.canlib.IOControlItem attribute*), 115
GET_RX_BUFFER_LEVEL (*canlib.canlib.IOControlItem attribute*), 115
get_signal() (*canlib.kvadblib.Message method*), 159
get_signal_by_name() (*canlib.kvadblib.Message method*), 159
GET_THROTTLE_SCALED (*canlib.canlib.IOControlItem attribute*), 115
GET_TIME_SINCE_LAST_SEEN (*canlib.canlib.IOControlItem attribute*), 116
GET_TIMER_SCALE (*canlib.canlib.IOControlItem attribute*), 116
GET_TREF_LIST (*canlib.canlib.IOControlItem attribute*),

116
GET_TX_BUFFER_LEVEL (*canlib.canlib.IOControlItem attribute*), 116
GET_TXACK (*canlib.canlib.IOControlItem attribute*), 116
GET_USB_THROTTLE (*canlib.canlib.IOControlItem attribute*), 116
GET_USER_IOPORT (*canlib.canlib.IOControlItem attribute*), 116
GET_WAKEUP (*canlib.canlib.IOControlItem attribute*), 116
getBusOutputControl() (*canlib.canlib.Channel method*), 86
getBusParams() (*canlib.canlib.Channel method*), 86
getBusParamsFd() (*canlib.canlib.Channel method*), 86
getCanHandle() (*canlib.linlib.Channel method*), 228
getChannelData() (*in module canlib.linlib*), 236
getChannelData_CardNumber() (*canlib.canlib.Channel method*), 86
getChannelData_Chan_No_On_Card() (*canlib.canlib.Channel method*), 87
getChannelData_Cust_Name() (*canlib.canlib.Channel method*), 87
getChannelData_DriverName() (*canlib.canlib.Channel method*), 87
getChannelData_EAN() (*canlib.canlib.Channel method*), 87
getChannelData_EAN_short() (*canlib.canlib.Channel method*), 87
getChannelData_Firmware() (*canlib.canlib.Channel method*), 87
getChannelData_Name() (*canlib.canlib.Channel method*), 87
getChannelData_Serial() (*canlib.canlib.Channel method*), 87
getDefaultAddresses() (*canlib.kvrlib.kvrDiscovery class method*), 224
getDlcMismatch() (*canlib.kvlclib.Converter method*), 173
getErrorText() (*in module canlib.canlib*), 140
getFirmwareVersion() (*canlib.linlib.Channel method*), 228
getFirstWriterFormat() (*canlib.kvlclib.WriterFormat class method*), 182
getNextWriterFormat() (*canlib.kvlclib.WriterFormat class method*), 182
getNumberOfChannels() (*in module canlib.canlib*), 140
getOutputFilename() (*canlib.kvlclib.Converter method*), 173
getProperty() (*canlib.kvlclib.Converter method*), 173
getPropertyDefault() (*canlib.kvlclib.Converter method*), 173
getPropertyDefault() (*canlib.kvlclib.ReaderFormat method*), 181
getPropertyDefault() (*canlib.kvlclib.WriterFormat method*), 182
getResults() (*canlib.kvrlib.kvrDiscovery method*), 224
getTransceiverData() (*in module canlib.linlib*), 236
getVersion() (*in module canlib.canlib*), 141
getXml() (*canlib.kvrlib.kvrConfig method*), 225

H

HandleData (*class in canlib.canlib*), 100
HARDWARE (*canlib.canlib.Error attribute*), 111
HardwareType (*class in canlib.canlib*), 113
has() (*canlib.kvrlib.DeviceInfoSet method*), 212
HEX (*canlib.kvadblib.AttributeType attribute*), 153
HexDefinition (*class in canlib.kvadblib*), 147
HIGH (*canlib.canlib.iopin.DigitalValue attribute*), 133
high_low_filter (*canlib.canlib.iopin.DigitalIn property*), 132
HIGHSPEED (*canlib.canlib.DriverCap attribute*), 110
hilo() (*canlib.EAN method*), 67
HLP_J1939 (*canlib.kvlclib.Property attribute*), 178
HOST_FILE (*canlib.canlib.Error attribute*), 111
host_name (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
hostname (*canlib.kvrlib.DeviceInfo property*), 200
hostname (*canlib.kvrlib.RemoteDevice property*), 218
hostname() (*in module canlib.kvrlib*), 224
HW_OVERRUN (*canlib.canlib.MessageFlag attribute*), 120
HW_OVERRUN (*canlib.canlib.Stat attribute*), 124
HW_STATUS (*canlib.canlib.ChannelDataItem attribute*), 107
hysteresis (*canlib.canlib.iopin.AnalogIn property*), 130
hysteresis (*canlib.canlib.iopin.IoPin property*), 134

I

id (*canlib.Frame attribute*), 70
id (*canlib.kvadblib.Message property*), 159
id (*canlib.LINFrame attribute*), 71
ID_IN_HEX (*canlib.kvlclib.Property attribute*), 179
IDLE (*canlib.canlib.ScriptStatus attribute*), 123
idPar (*canlib.linlib.MessageInfo attribute*), 235
IGNORED_ELEMENT (*canlib.kvamemolibxml.ValidationWarning attribute*), 168
ILLEGAL_REQUEST (*canlib.kvmlib.Error attribute*), 184
IN (*canlib.canlib.iopin.Direction attribute*), 133
IN_USE (*canlib.kvadblib.Error attribute*), 153
INDEPENDENT (*canlib.kvrlib.BasicServiceSet attribute*), 205
index() (*canlib.canlib.iopin.Configuration method*), 131
info (*canlib.kvrlib.ConfigProfile property*), 213
info (*canlib.LINFrame attribute*), 71
Info (*class in canlib.canlib.iopin*), 133
info() (*canlib.kvrlib.DeviceInfo method*), 200

INFRASTRUCTURE (*canlib.canlib.OperationalMode attribute*), 123
INFRASTRUCTURE (*canlib.kvrlib.BasicServiceSet attribute*), 205
INIFILE (*canlib.canlib.Error attribute*), 111
initializeLibrary() (*in module canlib.canlib*), 141
initializeLibrary() (*in module canlib.linlib*), 236
INITIALIZING (*canlib.kvrlib.NetworkState attribute*), 207
INSTALLING (*canlib.kvrlib.RemoteState attribute*), 208
INSTALLING (*canlib.kvrlib.ServiceState attribute*), 209
INT (*canlib.canlib.EnvVarType attribute*), 110
INTEGER (*canlib.kvadblib.AttributeType attribute*), 153
IntegerDefinition (*class in canlib.kvadblib*), 147
INTEL (*canlib.kvadblib.SignalByteOrder attribute*), 155
INTERFACE (*canlib.canlib.DeviceMode attribute*), 109
INTERNAL (*canlib.canlib.BusTypeGroup attribute*), 104
INTERNAL (*canlib.canlib.Error attribute*), 111
INTERNAL (*canlib.canlib.iopin.ModuleType attribute*), 135
INTERNAL (*canlib.kvadblib.Error attribute*), 153
INTERNAL (*canlib.kvamemolibxml.Error attribute*), 166
INTERNAL (*canlib.linlib.Error attribute*), 232
INTERNAL_ERROR (*canlib.kvlclib.Error attribute*), 176
interpret() (*canlib.kvadblib.Dbc method*), 150
INTERRUPTED (*canlib.canlib.Error attribute*), 111
INVALID (*canlib.kvadblib.AttributeOwner attribute*), 152
INVALID (*canlib.kvadblib.AttributeType attribute*), 153
INVALID (*canlib.kvadblib.SignalType attribute*), 155
INVALID (*canlib.kvlclib.FileFormat attribute*), 177
INVALID (*canlib.kvrlib.NetworkState attribute*), 207
INVALID_LOG_EVENT (*canlib.kvlclib.Error attribute*), 176
INVALID_PASSWORD (*canlib.canlib.Error attribute*), 111
INVALID_SESSION (*canlib.canlib.Error attribute*), 111
INVHANDLE (*canlib.canlib.Error attribute*), 111
INVHANDLE (*canlib.linlib.Error attribute*), 232
IO_API (*canlib.canlib.ChannelCap attribute*), 105
IO_CONFIG_CHANGED (*canlib.canlib.Error attribute*), 111
io_confirm_config() (*canlib.canlib.Channel method*), 88
IO_NO_VALID_CONFIG (*canlib.canlib.Error attribute*), 111
IO_NOT_CONFIRMED (*canlib.canlib.Error attribute*), 111
IO_PENDING (*canlib.canlib.Error attribute*), 111
io_pins() (*canlib.canlib.Channel method*), 88
IO_WRONG_PIN_TYPE (*canlib.canlib.Error attribute*), 111
iocontrol (*canlib.canlib.Channel property*), 88
IOControl (*class in canlib.canlib*), 127
iocontrol() (*canlib.Device method*), 69
IOControl.clear_error_counters() (*in module canlib.canlib*), 128
IOControl.connect_to_virtual_bus() (*in module canlib.canlib*), 128
IOControl.disconnect_from_virtual_bus() (*in module canlib.canlib*), 128
IOControl.flush_rx_buffer() (*in module canlib.canlib*), 128
IOControl.flush_tx_buffer() (*in module canlib.canlib*), 128
IOControl.prefer_ext() (*in module canlib.canlib*), 128
IOControl.prefer_std() (*in module canlib.canlib*), 128
IOControl.reset_overrun_count() (*in module canlib.canlib*), 128
IOControlItem (*class in canlib.canlib*), 115
ioCtl_flush_rx_buffer() (*canlib.canlib.Channel method*), 88
ioCtl_get_report_access_errors() (*canlib.canlib.Channel method*), 88
ioCtl_set_report_access_errors() (*canlib.canlib.Channel method*), 88
ioCtl_set_timer_scale() (*canlib.canlib.Channel method*), 88
IoNoValidConfiguration, 75
IoPin (*class in canlib.canlib.iopin*), 134
IoPinConfigurationNotConfirmed, 75
IPV4 (*canlib.kvrlib.AddressType attribute*), 204
IPV4_PORT (*canlib.kvrlib.AddressType attribute*), 204
IPV6 (*canlib.kvrlib.AddressType attribute*), 204
IRIS (*canlib.canlib.HardwareType attribute*), 113
is_can_fd() (*canlib.canlib.Channel method*), 88
IS_CANFD (*canlib.canlib.ChannelFlags attribute*), 109
is_dhcp (*canlib.kvrlib.AddressInfo property*), 213
is_encrypted (*canlib.canlib.Txe property*), 137
IS_ENCRYPTED (*canlib.canlib.TxeDataItem attribute*), 127
is_enum (*canlib.kvadblib.BoundSignal property*), 157
IS_EXCLUSIVE (*canlib.canlib.ChannelFlags attribute*), 109
IS_LIN (*canlib.canlib.ChannelFlags attribute*), 109
IS_LIN_MASTER (*canlib.canlib.ChannelFlags attribute*), 109
IS_LIN_SLAVE (*canlib.canlib.ChannelFlags attribute*), 109
IS_OPEN (*canlib.canlib.ChannelFlags attribute*), 109
IS_REMOTE (*canlib.canlib.ChannelDataItem attribute*), 107
isconnected() (*canlib.Device method*), 69
IsDataTruncated() (*canlib.kvlclib.Converter method*), 172
isDataTruncated() (*canlib.kvlclib.Converter method*), 173
isDlcMismatch() (*canlib.kvlclib.Converter method*), 174

ISO8601_DECIMALS (*canlib.kvlclib.Property attribute*), 179
IsOutputFilenameNew() (*canlib.kvlclib.Converter method*), 172
isOutputFilenameNew() (*canlib.kvlclib.Converter method*), 174
IsOverrunActive() (*canlib.kvlclib.Converter method*), 172
isOverrunActive() (*canlib.kvlclib.Converter method*), 174
isPropertySupported() (*canlib.kvlclib.Converter method*), 174
isPropertySupported() (*canlib.kvlclib.ReaderFormat method*), 181
isPropertySupported() (*canlib.kvlclib.WriterFormat method*), 182
issubset() (*canlib.canlib.iopin.AddonModule method*), 129
issubset() (*canlib.canlib.iopin.Configuration method*), 132
issubset() (*canlib.Device method*), 69

J

J1587 (*canlib.kvlclib.FileFormat attribute*), 177
J1587_ALT (*canlib.kvlclib.FileFormat attribute*), 177
J1708 (*canlib.kvadblib.ProtocolType attribute*), 154
J1939 (*canlib.kvadblib.MessageFlag attribute*), 154
JAPAN_TELEC (*canlib.kvrlib.RegulatoryDomain attribute*), 207

K

K (*canlib.canlib.TransceiverType attribute*), 125
K251 (*canlib.canlib.TransceiverType attribute*), 125
key128 (*canlib.kvrlib.WEPKeys property*), 225
key64 (*canlib.kvrlib.WEPKeys property*), 225
Kme (*class in canlib.kvmlib*), 191
KME24 (*canlib.kvlclib.FileFormat attribute*), 177
KME24 (*canlib.kvmlib.FileType attribute*), 185
KME25 (*canlib.kvlclib.FileFormat attribute*), 177
KME25 (*canlib.kvmlib.FileType attribute*), 185
KME40 (*canlib.kvlclib.FileFormat attribute*), 177
KME40 (*canlib.kvmlib.FileType attribute*), 186
KME50 (*canlib.kvlclib.FileFormat attribute*), 177
KME50 (*canlib.kvmlib.FileType attribute*), 186
KME60 (*canlib.kvlclib.FileFormat attribute*), 177
KME60 (*canlib.kvmlib.FileType attribute*), 186
kme_file_type() (*in module canlib.kvmlib*), 192
Kmf (*class in canlib.kvmlib*), 193
KmfSystem (*class in canlib.kvmlib*), 193
KmfSystem.DiskUsage (*class in canlib.kvmlib*), 193
KONE (*canlib.canlib.TransceiverType attribute*), 125
kvaBufferToXml() (*in module canlib.kvamemolibxml*), 169
KvaError, 164
kvaXmlToBuffer() (*in module canlib.kvamemolibxml*), 169
kvaXmlToFile() (*in module canlib.kvamemolibxml*), 169
kvaXmlValidate() (*in module canlib.kvamemolibxml*), 169
KvdBufferTooSmall, 143
KvdDbFileParse, 143
KvdErrInParameter, 143
KvdError, 143
kvDeviceGetMode() (*canlib.canlib.Channel method*), 89
kvDeviceSetMode() (*canlib.canlib.Channel method*), 89
KvdInUse, 143
KvdNoAttribute, 144
KvdNoMessage, 144
KvdNoNode, 144
KvdNoSignal, 144
KvdNotFound, 144
KvdOnlyOneAllowed, 144
KvlcEndOfFile, 171
KvlcError, 171
KvlcFileExists, 171
KvlcNotImplemented, 171
KvmDiskError, 183
KvmDiskNotFormated, 183
KvmError, 182
KVMLIB (*canlib.kvmlib.enums.SoftwareInfoItem attribute*), 187
kvmlib_version (*canlib.kvmlib.Memorator property*), 196
KvmNoDisk, 183
KvmNoLogMsg, 183
kvrAddress (*class in canlib.kvrlib*), 220
kvrAddressList (*class in canlib.kvrlib*), 221
KvrBlank, 197
kvrConfig (*class in canlib.kvrlib*), 225
kvrDeviceInfo (*class in canlib.kvrlib*), 221
kvrDeviceInfoList (*class in canlib.kvrlib*), 222
kvrDiscovery (*class in canlib.kvrlib*), 224
KvrError, 197
KvrGeneralError, 197
KvrNoAnswer, 197
KvrParameterError, 198
KvrPasswordError, 198
KvrUnreachable, 198
kvrVersion (*class in canlib.kvrlib*), 222

L

LAN (*canlib.canlib.RemoteType attribute*), 123
LAPCAN (*canlib.canlib.HardwareType attribute*), 113
last_channel_number (*canlib.Device attribute*), 69
LEAF (*canlib.canlib.HardwareType attribute*), 113

LEAF2 (*canlib.canlib.HardwareType attribute*), 113
LED_0_OFF (*canlib.canlib.LEDAction attribute*), 117
LED_0_ON (*canlib.canlib.LEDAction attribute*), 117
LED_10_OFF (*canlib.canlib.LEDAction attribute*), 117
LED_10_ON (*canlib.canlib.LEDAction attribute*), 118
LED_11_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_11_ON (*canlib.canlib.LEDAction attribute*), 118
LED_1_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_1_ON (*canlib.canlib.LEDAction attribute*), 118
LED_2_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_2_ON (*canlib.canlib.LEDAction attribute*), 118
LED_3_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_3_ON (*canlib.canlib.LEDAction attribute*), 118
LED_4_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_4_ON (*canlib.canlib.LEDAction attribute*), 118
LED_5_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_5_ON (*canlib.canlib.LEDAction attribute*), 118
LED_6_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_6_ON (*canlib.canlib.LEDAction attribute*), 118
LED_7_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_7_ON (*canlib.canlib.LEDAction attribute*), 118
LED_8_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_8_ON (*canlib.canlib.LEDAction attribute*), 118
LED_9_OFF (*canlib.canlib.LEDAction attribute*), 118
LED_9_ON (*canlib.canlib.LEDAction attribute*), 119
LEDAction (*class in canlib.canlib*), 117
length (*canlib.kvadblib.ValueSize property*), 163
LICENSE (*canlib.canlib.Error attribute*), 111
LICENSE (*canlib.linlib.Error attribute*), 232
lif (*canlib.kvamemolibxml.Configuration property*), 164
LIMIT_DATA_BYTES (*canlib.kvlclib.Property attribute*), 179
limits (*canlib.kvadblib.Signal property*), 161
LIN (*canlib.canlib.TransceiverType attribute*), 125
LIN (*canlib.kvadblib.ProtocolType attribute*), 154
LIN_HYBRID (*canlib.canlib.ChannelCap attribute*), 105
lin_master() (*canlib.Device method*), 69
LIN_MODE (*canlib.canlib.IOControlItem attribute*), 116
lin_slave() (*canlib.Device method*), 69
LinError, 226
LINFrame (*class in canlib*), 71
LinGeneralError, 226
LinNoMessageError, 226
LinNotImplementedError, 226
LINX_J1708 (*canlib.canlib.TransceiverType attribute*), 125
LINX_K (*canlib.canlib.TransceiverType attribute*), 125
LINX_LIN (*canlib.canlib.TransceiverType attribute*), 125
LINX_LS (*canlib.canlib.TransceiverType attribute*), 125
LINX_SWC (*canlib.canlib.TransceiverType attribute*), 125
load_lif() (*in module canlib.kvamemolibxml*), 165
load_lif_file() (*in module canlib.kvamemolibxml*), 165
load_xml() (*in module canlib.kvamemolibxml*), 165
load_xml_file() (*in module canlib.kvamemolibxml*), 165
LOADED (*canlib.canlib.ScriptStatus attribute*), 124
LOCAL (*canlib.canlib.BusTypeGroup attribute*), 104
LOCAL_TXACK (*canlib.canlib.MessageFlag attribute*), 120
LockedLogError, 183
log_type (*canlib.kvmlib.LogFile property*), 190
LogEvent (*class in canlib.kvmlib*), 187
LogFile (*class in canlib.kvmlib*), 189
LOGFILEOPEN (*canlib.kvmlib.Error attribute*), 184
LOGFILEREAD (*canlib.kvmlib.Error attribute*), 184
LogFileType (*class in canlib.kvmlib.enums*), 186
LOGFILEWRITE (*canlib.kvmlib.Error attribute*), 184
LOGGER (*canlib.canlib.ChannelCap attribute*), 105
LOGGER (*canlib.canlib.DeviceMode attribute*), 109
LOGGER_TYPE (*canlib.canlib.ChannelDataItem attribute*), 107
LoggerDataFormat (*class in canlib.kvmlib*), 186
LoggerType (*class in canlib.canlib*), 119
LOW (*canlib.canlib.iopin.DigitalValue attribute*), 133
low_high_filter (*canlib.canlib.iopin.DigitalIn property*), 132
lp_filter_order (*canlib.canlib.iopin.AnalogIn property*), 130
lp_filter_order (*canlib.canlib.iopin.IoPin property*), 134

M

MAC (*canlib.kvrlib.AddressType attribute*), 204
mac (*canlib.kvrlib.WlanScanResult property*), 220
major (*canlib.kvrlib.kvrVersion attribute*), 222
major (*canlib.VersionNumber property*), 71
MAJOR_CAN (*canlib.kvmlib.LoggerDataFormat attribute*), 186
MAJOR_CAN64 (*canlib.kvmlib.LoggerDataFormat attribute*), 186
MAP_RXQUEUE (*canlib.canlib.IOControlItem attribute*), 116
MASTER (*canlib.linlib.ChannelType attribute*), 231
MASTERONLY (*canlib.linlib.Error attribute*), 232
MATLAB (*canlib.kvlclib.FileFormat attribute*), 177
max (*canlib.kvadblib.MinMaxDefinition property*), 147
max (*canlib.kvadblib.ValueLimits property*), 162
MAX_BITRATE (*canlib.canlib.ChannelDataItem attribute*), 108
MDF (*canlib.kvlclib.FileFormat attribute*), 177
MDF_4X (*canlib.kvlclib.FileFormat attribute*), 177
MDF_4X_SIGNAL (*canlib.kvlclib.FileFormat attribute*), 177
MDF_SIGNAL (*canlib.kvlclib.FileFormat attribute*), 177
MEMO_FAIL (*canlib.canlib.Error attribute*), 111
MEMO_LOG (*canlib.kvlclib.FileFormat attribute*), 177
MEMOLOG_TYPE_CLOCK (*canlib.kvmlib.memoLogEventEx attribute*), 188

MEMOLOG_TYPE_INVALID (*canlib.kvmlib.memoLogEventEx attribute*), 188
MEMOLOG_TYPE_MSG (*canlib.kvmlib.memoLogEventEx attribute*), 188
MEMOLOG_TYPE_TRIGGER (*canlib.kvmlib.memoLogEventEx attribute*), 188
MEMOLOG_TYPE_VERSION (*canlib.kvmlib.memoLogEventEx attribute*), 188
memoLogEventEx (*class in canlib.kvmlib*), 188
Memorator (*class in canlib.kvmlib*), 194
memorator() (*canlib.Device method*), 69
MEMORATOR_II (*canlib.canlib.HardwareType attribute*), 114
MEMORATOR_LIGHT (*canlib.canlib.HardwareType attribute*), 114
MEMORATOR_PRO (*canlib.canlib.HardwareType attribute*), 114
MEMORATOR_PRO2 (*canlib.canlib.HardwareType attribute*), 114
MEMORATOR_V2 (*canlib.canlib.HardwareType attribute*), 114
MERGE_LINES (*canlib.kvlclib.Property attribute*), 179
MESSAGE (*canlib.kvadblib.AttributeOwner attribute*), 152
Message (*class in canlib.kvadblib*), 158
message() (*canlib.kvadblib.FrameBox method*), 156
MessageDisturb (*class in canlib.linlib*), 233
MessageEvent (*class in canlib.kvmlib*), 187
MessageFilter (*class in canlib.canlib.objbuf*), 138
MessageFlag (*class in canlib.canlib*), 119
MessageFlag (*class in canlib.kvadblib*), 154
MessageFlag (*class in canlib.linlib*), 233
MessageInfo (*class in canlib.linlib*), 234
MessageParity (*class in canlib.linlib*), 234
messages() (*canlib.kvadblib.Dbc method*), 150
messages() (*canlib.kvadblib.FrameBox method*), 156
MFGNAME_ASCII (*canlib.canlib.ChannelDataItem attribute*), 108
MFGNAME_UNICODE (*canlib.canlib.ChannelDataItem attribute*), 108
MHYDRA (*canlib.kvmlib.Device attribute*), 183
MHYDRA_EXT (*canlib.kvmlib.Device attribute*), 183
min (*canlib.kvadblib.MinMaxDefinition property*), 147
min (*canlib.kvadblib.ValueLimits property*), 162
MINIHYDRA (*canlib.canlib.HardwareType attribute*), 114
MINIPCIE (*canlib.canlib.HardwareType attribute*), 114
MinMaxDefinition (*class in canlib.kvadblib*), 147
minor (*canlib.kvrlib.kvrVersion attribute*), 222
minor (*canlib.VersionNumber property*), 72
MIXED_ENDIANESS (*canlib.kvlclib.Error attribute*), 176
mode (*canlib.kvadblib.Signal property*), 161
MODE_ERASE (*canlib.kvrlib.kvrConfig attribute*), 225
MODE_R (*canlib.kvrlib.kvrConfig attribute*), 225
MODE_RW (*canlib.kvrlib.kvrConfig attribute*), 225
module
    canlib.canlib.envvar, 102
    canlib.canlib.iopin, 129
    canlib.canlib.objbuf, 138
    canlib.j1939, 237
    canlib.kvrlib.service, 222
MODULE_NUMBER (*canlib.canlib.iopin.Info attribute*), 134
MODULE_TYPE (*canlib.canlib.iopin.Info attribute*), 134
module_type (*canlib.canlib.iopin.IoPin property*), 134
ModuleType (*class in canlib.canlib.iopin*), 135
MOST (*canlib.kvadblib.ProtocolType attribute*), 154
MOTOROLA (*canlib.kvadblib.SignalByteOrder attribute*), 155
mount() (*canlib.kvmlib.Memorator method*), 196
mounted (*canlib.kvmlib.Memorator attribute*), 196
MountedLog (*class in canlib.kvmlib*), 189
MSG_MASK (*canlib.canlib.MessageFlag attribute*), 120
MSGERR_MASK (*canlib.canlib.MessageFlag attribute*), 120
MULTIPLE_EXT_TRIGGER (*canlib.kvamemolibxml.ValidationError attribute*), 167
MULTIPLE_EXT_TRIGGER (*canlib.kvamemolibxml.ValidationWarning attribute*), 168
MULTIPLE_START_TRIGGER (*canlib.kvamemolibxml.ValidationError attribute*), 167
MUX_INDEPENDENT (*canlib.kvadblib.SignalMultiplexMode attribute*), 155
MUX_SIGNAL (*canlib.kvadblib.SignalMultiplexMode attribute*), 155

N

name (*canlib.canlib.SourceElement property*), 136
name (*canlib.kvadblib.Attribute property*), 145
name (*canlib.kvadblib.AttributeDefinition property*), 145
name (*canlib.kvadblib.BoundSignal property*), 157
name (*canlib.kvadblib.Dbc property*), 151
name (*canlib.kvadblib.Message property*), 159
name (*canlib.kvadblib.Node property*), 160
name (*canlib.kvadblib.Signal property*), 161
name (*canlib.kvrlib.DeviceInfo property*), 200
name (*canlib.kvrlib.kvrDeviceInfo attribute*), 221
name() (*canlib.canlib.iopin.Configuration method*), 132
NAME_MANGLING (*canlib.kvlclib.Property attribute*), 179
NERR (*canlib.canlib.MessageFlag attribute*), 120
netmask (*canlib.kvrlib.AddressInfo property*), 213
NetworkState (*class in canlib.kvrlib*), 207
new_attribute_definition() (*canlib.kvadblib.Dbc method*), 151
new_info() (*canlib.kvrlib.DeviceInfoSet method*), 212
new_message() (*canlib.kvadblib.Dbc method*), 151
new_node() (*canlib.kvadblib.Dbc method*), 151
new_signal() (*canlib.kvadblib.Message method*), 159

nextInputFile() (*canlib.kvlclib.Converter method*), 174
NO_ACCESS (*canlib.canlib.Error attribute*), 112
NO_ACCESS (*canlib.linlib.Error attribute*), 232
NO_ACTIVE_LOG (*canlib.kvamemolibxml.ValidationWarning attribute*), 168
NO_ANSWER (*canlib.kvrlib.Error attribute*), 206
NO_ATTRIB (*canlib.kvadblib.Error attribute*), 153
NO_DATABASE (*canlib.kvadblib.Error attribute*), 153
NO_DEVICE (*canlib.kvrlib.Error attribute*), 206
NO_DISK (*canlib.kvmlib.Error attribute*), 185
NO_FREE_HANDLES (*canlib.kvlclib.Error attribute*), 176
NO_INIT_ACCESS (*canlib.canlib.Open attribute*), 122
NO_INPUT_SELECTED (*canlib.kvlclib.Error attribute*), 176
NO_MSG (*canlib.kvadblib.Error attribute*), 154
NO_NODE (*canlib.kvadblib.Error attribute*), 154
NO_REF_POWER (*canlib.linlib.Error attribute*), 232
NO_SERVICE (*canlib.kvrlib.Error attribute*), 206
NO_SIGNAL (*canlib.kvadblib.Error attribute*), 154
NO_SUCH_FUNCTION (*canlib.canlib.Error attribute*), 112
NO_TIME_REFERENCE (*canlib.kvlclib.Error attribute*), 176
NOCARD (*canlib.canlib.Error attribute*), 111
NOCARD (*canlib.linlib.Error attribute*), 232
NOCHANNELS (*canlib.canlib.Error attribute*), 111
NOCHANNELS (*canlib.linlib.Error attribute*), 232
NOCONFIGMGR (*canlib.canlib.Error attribute*), 111
NODATA (*canlib.linlib.MessageFlag attribute*), 233
NODE (*canlib.kvadblib.AttributeOwner attribute*), 153
Node (*class in canlib.kvadblib*), 160
node_in_signal() (*canlib.kvadblib.Dbc method*), 151
nodes() (*canlib.kvadblib.Dbc method*), 152
nodes() (*canlib.kvadblib.Signal method*), 161
NOFLAG (*canlib.canlib.Open attribute*), 122
NOHANDLES (*canlib.canlib.Error attribute*), 111
NOHANDLES (*canlib.linlib.Error attribute*), 232
NOLOGMSG (*canlib.kvmlib.Error attribute*), 184
NOMEM (*canlib.canlib.Error attribute*), 112
NOMEM (*canlib.linlib.Error attribute*), 232
NOMSG (*canlib.canlib.Error attribute*), 112
NOMSG (*canlib.linlib.Error attribute*), 232
NONE (*canlib.canlib.HardwareType attribute*), 114
NONE (*canlib.canlib.Notify attribute*), 121
NONE (*canlib.canlib.OperationalMode attribute*), 123
NONE (*canlib.kvrlib.Availability attribute*), 204
NONE (*canlib.kvrlib.StartInfo attribute*), 210
NORMAL (*canlib.canlib.Driver attribute*), 109
NORMAL (*canlib.canlib.ScriptStop attribute*), 124
NORTH_AMERICA_FCC (*canlib.kvrlib.RegulatoryDomain attribute*), 208
NOSTARTTIME (*canlib.kvmlib.Error attribute*), 184
NOT_A_LOGGER (*canlib.canlib.LoggerType attribute*), 119
NOT_AUTHORIZED (*canlib.canlib.Error attribute*), 112
NOT_CONNECTED (*canlib.kvrlib.NetworkState attribute*), 207
NOT_FORMATTED (*canlib.kvmlib.Error attribute*), 185
NOT_IMPLEMENTED (*canlib.canlib.Error attribute*), 112
NOT_IMPLEMENTED (*canlib.kvlclib.Error attribute*), 176
NOT_IMPLEMENTED (*canlib.kvmlib.Error attribute*), 185
NOT_IMPLEMENTED (*canlib.kvrlib.Error attribute*), 206
NOT_IMPLEMENTED (*canlib.linlib.Error attribute*), 232
NOT_INITIALIZED (*canlib.kvrlib.Error attribute*), 206
NOT_REMOTE (*canlib.canlib.RemoteType attribute*), 123
NOT_SUPPORTED (*canlib.canlib.Error attribute*), 112
NOTFOUND (*canlib.canlib.Error attribute*), 112
NOTFOUND (*canlib.linlib.Error attribute*), 232
Notify (*class in canlib.canlib*), 121
NOTINITIALIZED (*canlib.canlib.Error attribute*), 112
NOTINITIALIZED (*canlib.linlib.Error attribute*), 232
NOTRUNNING (*canlib.linlib.Error attribute*), 232
NULL_MASK (*canlib.canlib.AcceptFilterFlag attribute*), 103
NULL_POINTER (*canlib.kvlclib.Error attribute*), 176
num_digits (*canlib.EAN attribute*), 67
NUM_OUT_OF_RANGE (*canlib.kvamemolibxml.ValidationError attribute*), 167
NUMBER_OF_BITS (*canlib.canlib.iopin.Info attribute*), 134
number_of_bits (*canlib.canlib.iopin.IoPin property*), 134
NUMBER_OF_DATA_DECIMALS (*canlib.kvlclib.Property attribute*), 179
number_of_io_pins() (*canlib.canlib.Channel method*), 89
NUMBER_OF_TIME_DECIMALS (*canlib.kvlclib.Property attribute*), 179

O

OCCUPIED (*canlib.kvmlib.Error attribute*), 185
OFF (*canlib.canlib.Driver attribute*), 109
offset (*canlib.kvadblib.ValueScaling property*), 163
OFFSET (*canlib.kvlclib.Property attribute*), 179
OK (*canlib.kvamemolibxml.Error attribute*), 166
OK (*canlib.kvamemolibxml.ValidationError attribute*), 167
OK (*canlib.kvmlib.Error attribute*), 185
OK (*canlib.kvrlib.Error attribute*), 206
ONE (*canlib.kvlclib.ChannelMask attribute*), 175
ONLINE (*canlib.kvrlib.NetworkState attribute*), 207
ONLY_ONE_ALLOWED (*canlib.kvadblib.Error attribute*), 154
Open (*class in canlib.canlib*), 121
open_channel() (*canlib.Device method*), 69
openChannel() (*in module canlib.canlib*), 81
openChannel() (*in module canlib.linlib*), 226
openDevice() (*in module canlib.kvmlib*), 194

openDevice() (*in module canlib.kvrlib*), 216
openDiscovery() (*in module canlib.kvrlib*), 202
openEx() (*canlib.kvrlib.kvrConfig method*), 225
openKme() (*in module canlib.kvmlib*), 191
openKmf() (*in module canlib.kvmlib*), 192
openMaster() (*in module canlib.linlib*), 227
openSlave() (*in module canlib.linlib*), 227
OperationalMode (*class in canlib.canlib*), 123
OUT (*canlib.canlib.iopin.Direction attribute*), 133
OUT_OF_SPACE (*canlib.kvrlib.Error attribute*), 206
OVERRIDE_EXCLUSIVE (*canlib.canlib.Open attribute*), 122
overrun (*canlib.canlib.channel.ErrorCounters property*), 82
OVERRUN (*canlib.canlib.MessageFlag attribute*), 120
OVERRUN (*canlib.canlib.Stat attribute*), 124
OVERWRITE (*canlib.kvlclib.Property attribute*), 179
owner (*canlib.kvadblib.AttributeDefinition property*), 145

P

p (*canlib.j1939.Pdu attribute*), 238
PARAM (*canlib.canlib.Error attribute*), 112
PARAM (*canlib.kvadblib.Error attribute*), 154
PARAM (*canlib.kvlclib.Error attribute*), 176
PARAM (*canlib.kvmlib.Error attribute*), 185
PARAM (*canlib.linlib.Error attribute*), 233
PARAMETER (*canlib.kvrlib.Error attribute*), 206
PARITY (*canlib.linlib.MessageDisturb attribute*), 233
PARITY_ERROR (*canlib.linlib.MessageFlag attribute*), 233
PARSER (*canlib.kvamemolibxml.ValidationError attribute*), 167
password (*canlib.kvrlib.ConfigProfile property*), 214
password (*canlib.kvrlib.DeviceInfo property*), 200
PASSWORD (*canlib.kvrlib.Error attribute*), 206
password_valid() (*canlib.kvrlib.RemoteDevice method*), 219
path (*canlib.canlib.Txe property*), 137
PC104_PLUS (*canlib.canlib.HardwareType attribute*), 114
PCCAN (*canlib.canlib.HardwareType attribute*), 114
PCICAN (*canlib.canlib.HardwareType attribute*), 114
PCICAN_II (*canlib.canlib.HardwareType attribute*), 114
PCICANX_II (*canlib.canlib.HardwareType attribute*), 114
PCIE_V2 (*canlib.canlib.HardwareType attribute*), 114
Pdu (*class in canlib.j1939*), 238
Pdu1 (*class in canlib.j1939*), 238
Pdu2 (*class in canlib.j1939*), 239
pdu_from_can_id() (*in module canlib.j1939*), 239
Periodic (*class in canlib.canlib.objbuf*), 138
PERMISSION_DENIED (*canlib.kvrlib.Error attribute*), 206
pf (*canlib.j1939.Pdu attribute*), 238
pgn (*canlib.j1939.Pdu1 attribute*), 238
pgn (*canlib.j1939.Pdu2 attribute*), 239
phys (*canlib.kvadblib.BoundSignal property*), 157
phys_from() (*canlib.kvadblib.Signal method*), 161
pin() (*canlib.canlib.iopin.Configuration method*), 132
PIN_TYPE (*canlib.canlib.iopin.Info attribute*), 134
pin_type (*canlib.canlib.iopin.IoPin property*), 135
PinType (*class in canlib.canlib.iopin*), 136
PLAIN_ASC (*canlib.kvlclib.FileFormat attribute*), 177
POSTFIXEXPR (*canlib.kvamemolibxml.Error attribute*), 166
PREFER_EXT (*canlib.canlib.IOControlItem attribute*), 116
PREFER_STD (*canlib.canlib.IOControlItem attribute*), 116
PRIVATE (*canlib.kvrlib.Accessibility attribute*), 203
probe_info() (*canlib.Device method*), 69
product() (*canlib.EAN method*), 67
prodversion() (*in module canlib.canlib*), 141
Property (*class in canlib.kvlclib*), 178
PROTECTED (*canlib.kvrlib.Accessibility attribute*), 203
protocol (*canlib.kvadblib.Dbc property*), 152
ProtocolType (*class in canlib.kvadblib*), 154
ps (*canlib.j1939.Pdu attribute*), 238
PUBLIC (*canlib.kvrlib.Accessibility attribute*), 203

Q

qualified_name (*canlib.kvadblib.Message property*), 159
qualified_name (*canlib.kvadblib.Signal property*), 162
query() (*in module canlib.kvrlib.service*), 222
QUEUE_FULL (*canlib.kvmlib.Error attribute*), 185

R

R (*canlib.kvrlib.ConfigMode attribute*), 205
RANGE_MAX (*canlib.canlib.iopin.Info attribute*), 134
range_max (*canlib.canlib.iopin.IoPin property*), 135
RANGE_MIN (*canlib.canlib.iopin.Info attribute*), 134
range_min (*canlib.canlib.iopin.IoPin property*), 135
raw (*canlib.kvadblib.BoundSignal property*), 157
raw() (*canlib.canlib.ChannelData method*), 100
raw() (*canlib.canlib.HandleData method*), 102
raw() (*canlib.canlib.IOControl method*), 128
raw_from() (*canlib.kvadblib.Signal method*), 162
read() (*canlib.canlib.Channel method*), 89
read() (*canlib.kvrlib.ConfigProfile method*), 214
read() (*canlib.linlib.Channel method*), 228
read_config() (*canlib.kvmlib.Memorator method*), 196
read_error_counters() (*canlib.canlib.Channel method*), 90
read_event() (*canlib.kvmlib.Kme method*), 192
readDeviceCustomerData() (*canlib.canlib.Channel method*), 90
reader_formats() (*in module canlib.kvlclib*), 180
ReaderFormat (*class in canlib.kvlclib*), 181

readSpecificSkip() (*canlib.canlib.Channel method*), 90
readStatus() (*canlib.canlib.Channel method*), 90
readSyncSpecific() (*canlib.canlib.Channel method*), 90
readTimer() (*canlib.canlib.Channel method*), 90
REDISCOVER (*canlib.kvrlib.RemoteState attribute*), 208
REDISCOVER (*canlib.kvrlib.ServiceState attribute*), 209
REDISCOVER_PENDING (*canlib.kvrlib.RemoteState attribute*), 208
REDISCOVER_PENDING (*canlib.kvrlib.ServiceState attribute*), 209
REGISTRY (*canlib.canlib.Error attribute*), 112
RegulatoryDomain (*class in canlib.kvrlib*), 207
reinitializeLibrary() (*in module canlib.canlib*), 141
RELAY (*canlib.canlib.iopin.ModuleType attribute*), 135
RELAY (*canlib.canlib.iopin.PinType attribute*), 136
Relay (*class in canlib.canlib.iopin*), 136
release (*canlib.VersionNumber property*), 72
REMOTE (*canlib.canlib.BusTypeGroup attribute*), 105
REMOTE (*canlib.kvrlib.DeviceUsage attribute*), 205
remote() (*canlib.Device method*), 70
REMOTE_ACCESS (*canlib.canlib.ChannelCap attribute*), 105
REMOTE_HOST_NAME (*canlib.canlib.ChannelDataItem attribute*), 108
REMOTE_MAC (*canlib.canlib.ChannelDataItem attribute*), 108
REMOTE_OPERATIONAL_MODE (*canlib.canlib.ChannelDataItem attribute*), 108
REMOTE_PROFILE_NAME (*canlib.canlib.ChannelDataItem attribute*), 108
REMOTE_TYPE (*canlib.canlib.ChannelDataItem attribute*), 108
RemoteDevice (*class in canlib.kvrlib*), 216
RemoteDevice.ProfileList (*class in canlib.kvrlib*), 217
RemoteState (*class in canlib.kvrlib*), 208
RemoteType (*class in canlib.canlib*), 123
REMOVE_ME (*canlib.kvrlib.RemoteState attribute*), 208
REMOVE_ME (*canlib.kvrlib.ServiceState attribute*), 209
remove_node() (*canlib.kvadblib.Signal method*), 162
REMOVED (*canlib.canlib.Notify attribute*), 121
reopen() (*canlib.kvmlib.Memorator method*), 196
request_connection (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
requestChipStatus() (*canlib.canlib.Channel method*), 90
requestMessage() (*canlib.linlib.Channel method*), 229
REQUIRE_EXTENDED (*canlib.canlib.Open attribute*), 122
REQUIRE_INIT_ACCESS (*canlib.canlib.Open attribute*), 122
RESAMPLE_COLUMN (*canlib.kvlclib.Property attribute*), 179
RESERVED (*canlib.canlib.OperationalMode attribute*), 123
reserved1 (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
reserved2 (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
RESERVED_1 (*canlib.canlib.ChannelCap attribute*), 105
RESERVED_1 (*canlib.canlib.Stat attribute*), 124
RESERVED_2 (*canlib.canlib.ChannelCap attribute*), 105
RESERVED_2 (*canlib.canlib.Error attribute*), 112
RESERVED_5 (*canlib.canlib.Error attribute*), 112
RESERVED_6 (*canlib.canlib.Error attribute*), 112
RESERVED_7 (*canlib.canlib.Error attribute*), 112
RESET_OVERRUN_COUNT (*canlib.canlib.IOControlItem attribute*), 116
resetDlcMismatch() (*canlib.kvlclib.Converter method*), 174
resetOverrunActive() (*canlib.kvlclib.Converter method*), 174
resetStatusTruncated() (*canlib.kvlclib.Converter method*), 174
Response (*class in canlib.canlib.objbuf*), 139
RESULT_TOO_BIG (*canlib.kvlclib.Error attribute*), 176
RESULT_TOO_BIG (*canlib.kvmlib.Error attribute*), 185
results() (*canlib.kvrlib.ConnectionTest method*), 215
results() (*canlib.kvrlib.Discovery method*), 201
ROUNDTRIP_TIME (*canlib.canlib.ChannelDataItem attribute*), 108
RPCIII (*canlib.kvlclib.FileFormat attribute*), 177
RS485 (*canlib.canlib.TransceiverType attribute*), 125
rssi (*canlib.kvrlib.ConnectionStatus property*), 214
rssi (*canlib.kvrlib.ConnectionTestResult property*), 215
rssi (*canlib.kvrlib.WlanScanResult property*), 220
rtc (*canlib.kvmlib.Memorator property*), 196
RTCEvent (*class in canlib.kvmlib*), 187
RTR (*canlib.canlib.MessageFlag attribute*), 120
rtt (*canlib.kvrlib.ConnectionTestResult property*), 215
run() (*canlib.kvrlib.ConnectionTest method*), 215
RUNNING (*canlib.canlib.ScriptStatus attribute*), 124
RUNNING (*canlib.linlib.Error attribute*), 233
RW (*canlib.kvrlib.ConfigMode attribute*), 205
rx (*canlib.canlib.channel.ErrorCounters property*), 82
RX (*canlib.canlib.Notify attribute*), 121
RX (*canlib.linlib.MessageFlag attribute*), 233
RX_PENDING (*canlib.canlib.Stat attribute*), 124
rx_rate (*canlib.kvrlib.ConnectionStatus property*), 214
RXERR (*canlib.canlib.Stat attribute*), 124

S

sa (*canlib.j1939.Pdu attribute*), 238
SAMPLE_AND_HOLD_TIMESTEP (*canlib.kvlclib.Property attribute*), 179
sample_point() (*canlib.canlib.busparams.BusParamsTq method*), 80

sample_point_ns() (*canlib.canlib.busparams.BusParamsTq method*), 80
scaling (*canlib.kvadblib.Signal property*), 162
SCRIPT (*canlib.canlib.ChannelCap attribute*), 105
SCRIPT (*canlib.kvamemolibxml.ValidationError attribute*), 167
SCRIPT_CONFLICT (*canlib.kvamemolibxml.ValidationError attribute*), 167
script_envvar_get_data() (*canlib.canlib.Channel method*), 93
script_envvar_set_data() (*canlib.canlib.Channel method*), 93
SCRIPT_ERROR (*canlib.kvamemolibxml.Error attribute*), 166
SCRIPT_FAIL (*canlib.canlib.Error attribute*), 112
SCRIPT_NOT_FOUND (*canlib.kvamemolibxml.ValidationError attribute*), 167
SCRIPT_TOO_LARGE (*canlib.kvamemolibxml.ValidationError attribute*), 167
SCRIPT_TOO_MANY (*canlib.kvamemolibxml.ValidationError attribute*), 167
SCRIPT_WRONG_VERSION (*canlib.canlib.Error attribute*), 112
scriptEnvvarClose() (*canlib.canlib.Channel method*), 91
scriptEnvvarGetData() (*canlib.canlib.Channel method*), 91
scriptEnvvarGetFloat() (*canlib.canlib.Channel method*), 91
scriptEnvvarGetInt() (*canlib.canlib.Channel method*), 91
scriptEnvvarOpen() (*canlib.canlib.Channel method*), 91
scriptEnvvarSetData() (*canlib.canlib.Channel method*), 91
scriptEnvvarSetFloat() (*canlib.canlib.Channel method*), 91
scriptEnvvarSetInt() (*canlib.canlib.Channel method*), 91
scriptGetText() (*canlib.canlib.Channel method*), 91
scriptLoadFile() (*canlib.canlib.Channel method*), 91
scriptLoadFileOnDevice() (*canlib.canlib.Channel method*), 92
ScriptRequest (*class in canlib.canlib*), 123
scriptRequestText() (*canlib.canlib.Channel method*), 92
scriptSendEvent() (*canlib.canlib.Channel method*), 92
scriptStart() (*canlib.canlib.Channel method*), 92
ScriptStatus (*class in canlib.canlib*), 123
scriptStatus() (*canlib.canlib.Channel method*), 92
ScriptStop (*class in canlib.canlib*), 124
scriptStop() (*canlib.canlib.Channel method*), 92
ScriptText (*class in canlib.canlib*), 142
scriptUnload() (*canlib.canlib.Channel method*), 92
SECTOR_ERASED (*canlib.kvmlib.Error attribute*), 185
security_text (*canlib.kvrlib.WlanScanResult property*), 220
SELFRECEPTION (*canlib.canlib.Driver attribute*), 109
send_burst() (*canlib.canlib.objbuf.Periodic method*), 138
send_node (*canlib.kvadblib.Message property*), 159
SEPARATOR_CHAR (*canlib.kvlclib.Property attribute*), 179
SEQ_ERROR (*canlib.kvmlib.Error attribute*), 185
ser_no (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
serial (*canlib.canlib.iopin.IoPin property*), 135
serial (*canlib.Device attribute*), 70
serial (*canlib.kvrlib.DeviceInfo property*), 200
serial (*canlib.linlib.TransceiverData property*), 237
SERIAL_NUMBER (*canlib.canlib.iopin.Info attribute*), 134
serial_number (*canlib.kvmlib.Memorator property*), 196
service_status (*canlib.kvrlib.DeviceInfo property*), 200
ServiceState (*class in canlib.kvrlib*), 209
ServiceStatus (*class in canlib.kvrlib*), 202
set_attribute_value() (*canlib.kvadblib.Dbc method*), 152
set_attribute_value() (*canlib.kvadblib.Message method*), 159
set_attribute_value() (*canlib.kvadblib.Node method*), 160
set_attribute_value() (*canlib.kvadblib.Signal method*), 162
SET_BRLIMIT (*canlib.canlib.IOControlItem attribute*), 116
SET_BUFFER_WRAPAROUND_MODE (*canlib.canlib.IOControlItem attribute*), 116
set_bus_params_tq() (*canlib.canlib.Channel method*), 94
SET_BUSON_TIME_AUTO_RESET (*canlib.canlib.IOControlItem attribute*), 116
SET_BYPASS_MODE (*canlib.canlib.IOControlItem attribute*), 116
set_callback() (*canlib.canlib.Channel method*), 94
set_clear_stored_devices_on_exit() (*in module canlib.kvrlib*), 202
SET_CODE_EXT (*canlib.canlib.AcceptFilterFlag attribute*), 103
SET_CODE_STD (*canlib.canlib.AcceptFilterFlag attribute*), 103
SET_ERROR_FRAMES_REPORTING (*canlib.canlib.IOControlItem attribute*), 116

set_filter() (*canlib.canlib.objbuf.Response method*), 139
set_frame() (*canlib.canlib.objbuf.Periodic method*), 138
set_frame() (*canlib.canlib.objbuf.Response method*), 140
SET_LOCAL_TXACK (*canlib.canlib.IOControlItem attribute*), 116
SET_LOCAL_TXECHO (*canlib.canlib.IOControlItem attribute*), 117
SET_MASK_EXT (*canlib.canlib.AcceptFilterFlag attribute*), 103
SET_MASK_STD (*canlib.canlib.AcceptFilterFlag attribute*), 103
set_msg_count() (*canlib.canlib.objbuf.Periodic method*), 138
set_period() (*canlib.canlib.objbuf.Periodic method*), 139
SET_REPORT_ACCESS_ERRORS (*canlib.canlib.IOControlItem attribute*), 117
set_rtr_only() (*canlib.canlib.objbuf.Response method*), 140
SET_RX_QUEUE_SIZE (*canlib.canlib.IOControlItem attribute*), 117
SET_THROTTLE_SCALED (*canlib.canlib.IOControlItem attribute*), 117
SET_TIMER_SCALE (*canlib.canlib.IOControlItem attribute*), 117
SET_TXACK (*canlib.canlib.IOControlItem attribute*), 117
SET_TXRQ (*canlib.canlib.IOControlItem attribute*), 117
SET_USB_THROTTLE (*canlib.canlib.IOControlItem attribute*), 117
SET_USER_IOPORT (*canlib.canlib.IOControlItem attribute*), 117
SET_WAKEUP (*canlib.canlib.IOControlItem attribute*), 117
setAddresses() (*canlib.kvrlib.kvrDiscovery method*), 224
setBitrate() (*canlib.linlib.Channel method*), 229
setBusOutputControl() (*canlib.canlib.Channel method*), 93
setBusParams() (*canlib.canlib.Channel method*), 93
setBusParamsFd() (*canlib.canlib.Channel method*), 93
setEncryptionKey() (*canlib.kvrlib.kvrDiscovery method*), 224
setInputFile() (*canlib.kvlclib.Converter method*), 174
setPassword() (*canlib.kvrlib.kvrDiscovery method*), 224
setProperty() (*canlib.kvlclib.Converter method*), 174
setScanTime() (*canlib.kvrlib.kvrDiscovery method*), 224
Setup (*class in canlib.linlib*), 234
setupIllegalMessage() (*canlib.linlib.Channel method*), 229
setupLIN() (*canlib.linlib.Channel method*), 229
setXml() (*canlib.kvrlib.kvrConfig method*), 225
SHOW_COUNTER (*canlib.kvlclib.Property attribute*), 179
SHOW_SIGNAL_SELECT (*canlib.kvlclib.Property attribute*), 179
SHOW_UNITS (*canlib.kvlclib.Property attribute*), 179
SIGNAL (*canlib.kvadblib.AttributeOwner attribute*), 153
SIGNAL (*canlib.kvadblib.SignalMultiplexMode attribute*), 155
Signal (*class in canlib.kvadblib*), 161
signal() (*canlib.kvadblib.FrameBox method*), 156
SIGNAL_BASED (*canlib.kvlclib.Property attribute*), 179
SignalByteOrder (*class in canlib.kvadblib*), 155
SignalMultiplexMode (*class in canlib.kvadblib*), 155
SignalNotFound, 145
signals() (*canlib.kvadblib.FrameBox method*), 156
signals() (*canlib.kvadblib.Message method*), 159
SignalType (*class in canlib.kvadblib*), 155
SIGNED (*canlib.kvadblib.SignalType attribute*), 155
SILENT (*canlib.canlib.Driver attribute*), 109
SILENT_MODE (*canlib.canlib.ChannelCap attribute*), 106
SILENT_TRANSMIT (*canlib.kvamemolibxml.ValidationError attribute*), 168
SIMULATED (*canlib.canlib.ChannelCap attribute*), 106
SIMULATED (*canlib.canlib.HardwareType attribute*), 114
SINGLE_SHOT (*canlib.canlib.ChannelCap attribute*), 106
SINGLE_SHOT (*canlib.canlib.MessageFlag attribute*), 120
size (*canlib.kvadblib.Signal property*), 162
SIZE_LIMIT (*canlib.kvlclib.Property attribute*), 179
size_of_code (*canlib.canlib.Txe property*), 137
SIZE_OF_CODE (*canlib.canlib.TxeDataItem attribute*), 127
SLAVE (*canlib.linlib.ChannelType attribute*), 231
SLAVEONLY (*canlib.linlib.Error attribute*), 233
SoftwareInfoItem (*class in canlib.kvmlib.enums*), 186
source (*canlib.canlib.Txe property*), 137
SOURCE (*canlib.canlib.TxeDataItem attribute*), 127
SourceElement (*class in canlib.canlib*), 136
ssid (*canlib.kvrlib.WlanScanResult property*), 220
STANDARD (*canlib.linlib.MessageParity attribute*), 234
STANDBY (*canlib.kvrlib.RemoteState attribute*), 208
STANDBY (*canlib.kvrlib.ServiceState attribute*), 209
start() (*canlib.kvrlib.ConnectionTest method*), 216
start() (*canlib.kvrlib.Discovery method*), 201
start() (*canlib.kvrlib.kvrDiscovery method*), 224
start() (*in module canlib.kvrlib.service*), 222
start_discovery() (*in module canlib.kvrlib*), 202
start_info (*canlib.kvrlib.ServiceStatus property*), 202
START_OF_MEASUREMENT (*canlib.kvlclib.Property attribute*), 180
START_OK (*canlib.kvrlib.StartInfo attribute*), 210
start_time (*canlib.kvmlib.LogFile property*), 190
startbit (*canlib.kvadblib.ValueSize property*), 163

STARTED (*canlib.kvrlib.RemoteState attribute*), 208
STARTED (*canlib.kvrlib.ServiceState attribute*), 209
StartInfo (*class in canlib.kvrlib*), 210
STARTING (*canlib.kvrlib.RemoteState attribute*), 208
STARTING (*canlib.kvrlib.ServiceState attribute*), 209
STARTUP (*canlib.kvrlib.NetworkState attribute*), 207
Stat (*class in canlib.canlib*), 124
state (*canlib.kvrlib.ConnectionStatus property*), 214
state (*canlib.kvrlib.ServiceStatus property*), 202
status (*canlib.canlib.CanInvalidHandle attribute*), 73
status (*canlib.canlib.CanNoMsg attribute*), 73
status (*canlib.canlib.CanNotFound attribute*), 73
status (*canlib.canlib.CanOutOfMemory attribute*), 73
status (*canlib.canlib.CanScriptFail attribute*), 74
status (*canlib.canlib.CanTimeout attribute*), 74
status (*canlib.canlib.IoNoValidConfiguration attribute*), 75
status (*canlib.canlib.IoPinConfigurationNotConfirmed attribute*), 75
STATUS (*canlib.canlib.Notify attribute*), 121
status (*canlib.kvadblib.KvdBufferTooSmall attribute*), 143
status (*canlib.kvadblib.KvdDbFileParse attribute*), 143
status (*canlib.kvadblib.KvdErrInParameter attribute*), 143
status (*canlib.kvadblib.KvdInUse attribute*), 143
status (*canlib.kvadblib.KvdNoAttribute attribute*), 144
status (*canlib.kvadblib.KvdNoMessage attribute*), 144
status (*canlib.kvadblib.KvdNoNode attribute*), 144
status (*canlib.kvadblib.KvdNoSignal attribute*), 144
status (*canlib.kvadblib.KvdOnlyOneAllowed attribute*), 144
status (*canlib.kvlclib.KvlcEndOfFile attribute*), 171
status (*canlib.kvlclib.KvlcFileExists attribute*), 171
status (*canlib.kvlclib.KvlcNotImplemented attribute*), 171
status (*canlib.kvmlib.KvmDiskError attribute*), 183
status (*canlib.kvmlib.KvmDiskNotFormated attribute*), 183
status (*canlib.kvmlib.KvmNoDisk attribute*), 183
status (*canlib.kvmlib.KvmNoLogMsg attribute*), 183
status (*canlib.kvrlib.exceptions.KvrBlank attribute*), 197
status (*canlib.kvrlib.exceptions.KvrNoAnswer attribute*), 197
status (*canlib.kvrlib.exceptions.KvrParameterError attribute*), 198
status (*canlib.kvrlib.exceptions.KvrPasswordError attribute*), 198
status (*canlib.kvrlib.exceptions.KvrUnreachable attribute*), 198
status (*canlib.linlib.LinNoMessageError attribute*), 226
status (*canlib.linlib.LinNotImplementedError attribute*), 226
STD (*canlib.canlib.MessageFlag attribute*), 120
stop() (*canlib.kvrlib.ConnectionTest method*), 216
stop() (*in module canlib.kvrlib.service*), 222
STOPPING (*canlib.kvrlib.RemoteState attribute*), 208
STOPPING (*canlib.kvrlib.ServiceState attribute*), 209
store() (*canlib.kvrlib.DeviceInfoSet method*), 212
store_devices() (*in module canlib.kvrlib*), 203
STORED (*canlib.kvrlib.AddressTypeFlag attribute*), 204
STORED (*canlib.kvrlib.Availability attribute*), 204
stored (*canlib.kvrlib.DeviceInfo property*), 200
stored_devices() (*in module canlib.kvrlib*), 203
stored_info_set() (*in module canlib.kvrlib*), 211
storeDevices() (*canlib.kvrlib.kvrDiscovery method*), 225
STRING (*canlib.canlib.EnvVarType attribute*), 110
STRING (*canlib.kvadblib.AttributeType attribute*), 153
StringDefinition (*class in canlib.kvadblib*), 147
STRUCT_ARRAY (*canlib.kvrlib.kvrAddressList attribute*), 221
STRUCT_ARRAY (*canlib.kvrlib.kvrDeviceInfoList attribute*), 222
struct_size (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
STUFF (*canlib.canlib.MessageFlag attribute*), 120
SUBSCRIBE (*canlib.canlib.ScriptRequest attribute*), 123
SW_OVERRUN (*canlib.canlib.MessageFlag attribute*), 120
SW_OVERRUN (*canlib.canlib.Stat attribute*), 124
SWC (*canlib.canlib.TransceiverType attribute*), 126
SWC_OPTO (*canlib.canlib.TransceiverType attribute*), 126
SWC_PROTO (*canlib.canlib.TransceiverType attribute*), 126
SYNC_ERROR (*canlib.linlib.MessageFlag attribute*), 234
sync_jump_width() (*canlib.canlib.busparams.BusParamsTq method*), 80
SYNCH_ERROR (*canlib.linlib.MessageFlag attribute*), 233
synchBreakLength (*canlib.linlib.MessageInfo attribute*), 235
synchEdgeTime (*canlib.linlib.MessageInfo attribute*), 235

T

T_1041 (*canlib.canlib.TransceiverType attribute*), 126
T_1041_OPTO (*canlib.canlib.TransceiverType attribute*), 126
T_1050 (*canlib.canlib.TransceiverType attribute*), 126
T_1050_OPTO (*canlib.canlib.TransceiverType attribute*), 126
T_1054_OPTO (*canlib.canlib.TransceiverType attribute*), 126
T_251 (*canlib.canlib.TransceiverType attribute*), 126
T_252 (*canlib.canlib.TransceiverType attribute*), 126
text (*canlib.kvamemolibxml.ValidationMessage property*), 165

text (*canlib.kvrlib.ServiceStatus property*), 202
THREE (*canlib.kvlclib.ChannelMask attribute*), 175
TIME_DECREASING (*canlib.kvlclib.Error attribute*), 176
TIME_LIMIT (*canlib.kvlclib.Property attribute*), 180
TIME_SINCE_LAST_SEEN (*canlib.canlib.ChannelDataItem attribute*), 108
TIMEOUT (*canlib.canlib.Error attribute*), 112
TIMEOUT (*canlib.kvmlib.Error attribute*), 185
TIMEOUT (*canlib.linlib.Error attribute*), 233
timestamp (*canlib.Frame attribute*), 70
timestamp (*canlib.LINFrame attribute*), 71
timestamp (*canlib.linlib.MessageInfo attribute*), 234
TIMESYNC_ENABLED (*canlib.canlib.ChannelDataItem attribute*), 108
TIMEZONE (*canlib.kvlclib.Property attribute*), 180
to_BitrateSetting() (*in module canlib.canlib.busparams*), 78
to_BusParamsTq() (*in module canlib.canlib.busparams*), 78
to_c() (*canlib.kvrlib.Address method*), 198
total (*canlib.kvmlib.KmfSystem.DiskUsage property*), 193
TRANS_CAP (*canlib.canlib.ChannelDataItem attribute*), 108
TRANS_SERIAL_NO (*canlib.canlib.ChannelDataItem attribute*), 108
TRANS_TYPE (*canlib.canlib.ChannelDataItem attribute*), 108
TRANS_UPC_NO (*canlib.canlib.ChannelDataItem attribute*), 108
TransceiverData (*class in canlib.linlib*), 237
TransceiverType (*class in canlib.canlib*), 125
translateBaud() (*in module canlib.canlib*), 142
TriggerEvent (*class in canlib.kvmlib*), 187
TT (*canlib.canlib.TransceiverType attribute*), 126
TWO (*canlib.kvlclib.ChannelMask attribute*), 175
tx (*canlib.canlib.channel.ErrorCounters property*), 82
TX (*canlib.canlib.Notify attribute*), 121
TX (*canlib.linlib.MessageFlag attribute*), 234
TX_INTERVAL (*canlib.canlib.IOControlItem attribute*), 117
TX_PENDING (*canlib.canlib.Stat attribute*), 125
tx_power (*canlib.kvrlib.ConnectionStatus property*), 214
tx_rate (*canlib.kvrlib.ConnectionStatus property*), 214
TXACK (*canlib.canlib.MessageFlag attribute*), 120
TXACKNOWLEDGE (*canlib.canlib.ChannelCap attribute*), 106
TXBUFOFL (*canlib.canlib.Error attribute*), 112
Txe (*class in canlib.canlib*), 136
TXE_CONTAINER_FORMAT (*canlib.canlib.Error attribute*), 112
TXE_CONTAINER_VERSION (*canlib.canlib.Error attribute*), 113
TxeDataItem (*class in canlib.canlib*), 126
TxeFileIsEncrypted, 75
TXERR (*canlib.canlib.Stat attribute*), 125
TXNACK (*canlib.canlib.MessageFlag attribute*), 121
TXREQUEST (*canlib.canlib.ChannelCap attribute*), 106
TXRQ (*canlib.canlib.MessageFlag attribute*), 121
type (*canlib.kvadblib.Signal property*), 162
type (*canlib.kvrlib.kvrAddress attribute*), 220
type (*canlib.linlib.TransceiverData property*), 237
Type_IPV4 (*canlib.kvrlib.kvrAddress attribute*), 220
Type_IPV4_PORT (*canlib.kvrlib.kvrAddress attribute*), 220
Type_IPV6 (*canlib.kvrlib.kvrAddress attribute*), 220
Type_MAC (*canlib.kvrlib.kvrAddress attribute*), 220
TYPE_MISMATCH (*canlib.kvlclib.Error attribute*), 176
Type_UNKNOWN (*canlib.kvrlib.kvrAddress attribute*), 220
TypeText (*canlib.kvrlib.kvrAddress attribute*), 220

U

U100 (*canlib.canlib.HardwareType attribute*), 114
UI_NUMBER (*canlib.canlib.ChannelDataItem attribute*), 108
UNDEFINED_TRIGGER (*canlib.kvamemolibxml.ValidationError attribute*), 168
unit (*canlib.kvadblib.BoundSignal property*), 157
unit (*canlib.kvadblib.Signal property*), 162
UNKNOWN (*canlib.canlib.TransceiverType attribute*), 126
UNKNOWN (*canlib.kvadblib.ProtocolType attribute*), 155
UNKNOWN (*canlib.kvrlib.Accessibility attribute*), 203
UNKNOWN (*canlib.kvrlib.AddressType attribute*), 204
UNKNOWN (*canlib.kvrlib.DeviceUsage attribute*), 205
UNKNOWN (*canlib.kvrlib.NetworkState attribute*), 207
unload() (*in module canlib.kvrlib*), 225
unloadLibrary() (*in module canlib.canlib*), 142
unloadLibrary() (*in module canlib.linlib*), 237
UnmountedLog (*class in canlib.kvmlib*), 188
UNSIGNED (*canlib.kvadblib.SignalType attribute*), 155
UNSUBSCRIBE (*canlib.canlib.ScriptRequest attribute*), 123
UNSUPPORTED_VERSION (*canlib.kvmlib.Error attribute*), 185
UNWILLING (*canlib.kvrlib.RemoteState attribute*), 208
UNWILLING (*canlib.kvrlib.ServiceState attribute*), 209
update() (*canlib.kvrlib.DeviceInfo method*), 200
update() (*canlib.kvrlib.DeviceInfoSet method*), 212
updateMessage() (*canlib.linlib.Channel method*), 230
usage (*canlib.kvrlib.DeviceInfo property*), 200
usage (*canlib.kvrlib.kvrDeviceInfo attribute*), 222
USB (*canlib.kvrlib.DeviceUsage attribute*), 205
USBCAN (*canlib.canlib.HardwareType attribute*), 114
USBCAN_II (*canlib.canlib.HardwareType attribute*), 114
USBCAN_KLINE (*canlib.canlib.HardwareType attribute*), 114

USBCAN_LIGHT (*canlib.canlib.HardwareType attribute*), 114
USBCAN_PRO (*canlib.canlib.HardwareType attribute*), 115
USBCAN_PRO2 (*canlib.canlib.HardwareType attribute*), 115
USE_OFFSET (*canlib.kvlclib.Property attribute*), 180
used (*canlib.kvmlib.KmfSystem.DiskUsage property*), 193
USER_CANCEL (*canlib.kvmlib.Error attribute*), 185

V

V1 (*canlib.canlib.LoggerType attribute*), 119
V2 (*canlib.canlib.LoggerType attribute*), 119
validate() (*canlib.canlib.busparams.BusParamTqLimits method*), 80
validate() (*canlib.kvamemolibxml.Configuration method*), 164
validate() (*canlib.kvmlib.MountedLog method*), 189
ValidationError (*class in canlib.kvamemolibxml*), 167
ValidationErrorMessage (*class in canlib.kvamemolibxml*), 166
ValidationMessage (*class in canlib.kvamemolibxml*), 165
ValidationWarning (*class in canlib.kvamemolibxml*), 168
ValidationWarningMessage (*class in canlib.kvamemolibxml*), 166
value (*canlib.canlib.iopin.AnalogIn property*), 130
value (*canlib.canlib.iopin.AnalogOut property*), 130
value (*canlib.canlib.iopin.DigitalIn property*), 132
value (*canlib.canlib.iopin.DigitalOut property*), 133
value (*canlib.canlib.iopin.IoPin property*), 135
value (*canlib.canlib.iopin.Relay property*), 136
value (*canlib.kvadblib.Attribute property*), 145
value (*canlib.kvadblib.BoundSignal property*), 157
VALUE_CONSECUTIVE (*canlib.kvamemolibxml.Error attribute*), 166
VALUE_RANGE (*canlib.kvamemolibxml.Error attribute*), 166
VALUE_UNIQUE (*canlib.kvamemolibxml.Error attribute*), 166
ValueLimits (*class in canlib.kvadblib*), 162
ValueScaling (*class in canlib.kvadblib*), 163
ValueSize (*class in canlib.kvadblib*), 163
VAN (*canlib.kvadblib.ProtocolType attribute*), 155
VARIABLE_DLC (*canlib.linlib.Setup attribute*), 234
VECTOR_ASC (*canlib.kvlclib.FileFormat attribute*), 177
VECTOR_BLF (*canlib.kvlclib.FileFormat attribute*), 178
VECTOR_BLF_FD (*canlib.kvlclib.FileFormat attribute*), 178
verify_xml() (*in module canlib.kvrlib*), 225
VERSION (*canlib.kvlclib.Property attribute*), 180
VERSION (*canlib.linlib.Error attribute*), 233
VersionEvent (*class in canlib.kvmlib*), 188
VersionNumber (*class in canlib*), 71
VIRTUAL (*canlib.canlib.BusTypeGroup attribute*), 105
VIRTUAL (*canlib.canlib.ChannelCap attribute*), 106
VIRTUAL (*canlib.canlib.HardwareType attribute*), 115
VOID (*canlib.kvrlib.RemoteState attribute*), 209
VOID (*canlib.kvrlib.ServiceState attribute*), 210

W

W210 (*canlib.canlib.TransceiverType attribute*), 126
WAKEUP (*canlib.canlib.MessageFlag attribute*), 121
WAKEUP (*canlib.kvadblib.MessageFlag attribute*), 154
WAKEUP_FRAME (*canlib.linlib.MessageFlag attribute*), 234
WEPKeys (*class in canlib.kvrlib*), 225
WLAN (*canlib.canlib.RemoteType attribute*), 123
wlan_scan() (*canlib.kvrlib.RemoteDevice method*), 219
WlanScan (*class in canlib.kvrlib*), 219
WlanScanResult (*class in canlib.kvrlib*), 220
WORLD (*canlib.kvrlib.RegulatoryDomain attribute*), 208
write() (*canlib.canlib.Channel method*), 95
write() (*canlib.kvrlib.ConfigProfile method*), 214
write_config() (*canlib.kvmlib.Memorator method*), 196
write_event() (*canlib.kvmlib.Kme method*), 192
write_file() (*canlib.kvadblib.Dbc method*), 152
WRITE_HEADER (*canlib.kvlclib.Property attribute*), 180
WRITE_PROT (*canlib.kvmlib.Error attribute*), 185
write_raw() (*canlib.canlib.Channel method*), 96
writeMessage() (*canlib.linlib.Channel method*), 230
writer_formats() (*in module canlib.kvlclib*), 181
WriterFormat (*class in canlib.kvlclib*), 182
writeSync() (*canlib.canlib.Channel method*), 95
writeSync() (*canlib.linlib.Channel method*), 230
writeWait() (*canlib.canlib.Channel method*), 96
writeWait_raw() (*canlib.canlib.Channel method*), 96
writeWakeup() (*canlib.linlib.Channel method*), 231
WRONG_DISK_TYPE (*canlib.kvmlib.Error attribute*), 185
WRONG_OWNER (*canlib.kvadblib.Error attribute*), 154
WRONGRESP (*canlib.linlib.Error attribute*), 233

X

XCP (*canlib.kvlclib.FileFormat attribute*), 178
xml (*canlib.kvamemolibxml.Configuration property*), 164
XML_BUFFER_SIZE (*canlib.kvrlib.ConfigProfile attribute*), 213
XML_BUFFER_SIZE (*canlib.kvrlib.kvrConfig attribute*), 225
XML_PARSER (*canlib.kvamemolibxml.Error attribute*), 166
XML_VALIDATION (*canlib.kvrlib.Error attribute*), 206
xmlGetLastError() (*in module canlib.kvamemolibxml*), 170

xmlGetValidationError() (*in module canlib.kvamemolibxml*), 170
xmlGetValidationStatusCount() (*in module canlib.kvamemolibxml*), 170
xmlGetValidationWarning() (*in module canlib.kvamemolibxml*), 170