CAN Device Python SDK – API Reference

This document provides a detailed reference for the CAN Device Python SDK. It describes the initialisation sequence, the hardware management APIs, the configuration controls, and the CAN communication methods exposed by the SDK.

1. Initialisation

__init__()

Initialises the CAN SDK and loads the underlying DLL. This function allocates internal resources and validates communication with the CAN hardware.

- Raises:

  RuntimeError – If the DLL fails to load or the hardware initialisation fails.

2. Hardware Management

get_channel()

Retrieves the active communication channel associated with the CAN device.

- Returns:

  str – COM port or channel name, or None if no active channel is available.

open()

Opens the connection to the CAN device and prepares it for configuration and operation.

- Raises:

  RuntimeError – If the device fails to open.

close()

Closes the active connection to the CAN device. Communication is disabled until the device is reopened.

deinit()

Releases all DLL handles and internal SDK resources. This method should be called before application termination.

3. Configuration & Control

set_bitrate(bitrate)

Configures the CAN bus baud rate. This method must be called before starting the CAN controller.

- Parameters:

  bitrate (uint32) – CAN baud rate in bits per second
  Supported CAN bit rates: 10 kbps, 20 kbps, 50 kbps, 83.3 kbps, 100 kbps, 125 kbps, 250 kbps, 500 kbps, 750 kbps, 1 Mbps

start()

Starts the CAN controller and enables message transmission and reception.

stop()

Stops CAN message processing while keeping the device connection open.

4. Communication Methods

transmit(can_id, data, ext=False)

Transmits a CAN frame onto the bus.

- Parameters:

  can_id (int) – CAN identifier.
  data (bytes) – Payload length from 1 to 8 bytes.
  ext (bool) – Set to True for extended (29-bit) identifiers; False for standard (11-bit).

- Returns:

  int – DLL return code (0 indicates successful transmission).

read()

Performs a non-blocking read of a single CAN frame from the receive buffer.

- Returns:
- dict – CAN frame data, or None if no message is available.
- Return Format:

```
{
 "id": <int>,
 "dlc": <int>,
 "data_hex": <str>,
 "ext": <bool>,
 "error": <bool>,
 "rtr": <bool>
}
```

read_timeout(timeout)

Blocks until a CAN frame is received or the specified timeout expires.

- Parameters:

  timeout (uint8) – Timeout duration in milliseconds.

- Returns:

  dict – CAN frame data, or None if the timeout expires.

- Return Format:

```
{
  "id": <int>,
  "dlc": <int>,
  "data_hex": <str>,
  "ext": <bool>,
  "error": <bool>,
  "rtr": <bool>
}
```

read_withMask(filter_id, mask_id, timeout)

Reads a CAN frame that matches the specified CAN ID filter and mask.

- Parameters:

  filter_id (int) – Target CAN identifier.
  mask_id (int) – Bitmask applied to the filter.
  timeout (uint8) – Timeout duration in milliseconds.

- Returns:

  dict – Matching CAN frame, or None if no matching frame is received.

- Return Format:

```
{
  "id": <int>,
  "dlc": <int>,
  "data_hex": <str>,
  "ext": <bool>,
  "error": <bool>,
  "rtr": <bool>
}
```