autosar-e2e

Release 1.0.0

Artur Drogunow

Jun 01, 2026

CONTENTS

CHAPTER

ONE

LIBRARY API

1.1 E2E Profiles

1.1.1 Profile 01

e2e.p01.**e2e_p01_protect**(*data: bytearray*, *data_id: int*, ** (Keyword-only parameters separator (PEP 3102))*, *data_id_mode: int = E2E_P01_DATAID_BOTH*, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 1.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object starting with the CRC byte. This CRC byte will be updated inplace.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **data_id_mode** (*int*) – This attribute describes the inclusion mode that is used to include the *data_id*. The possible inclusion modes are *E2E_P01_DATAID_BOTH*, *E2E_P01_DATAID_ALT*, *E2E_P01_DATAID_LOW* and *E2E_P01_DATAID_NIBBLE*.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.
- **increment_counter** (*bool*) – If *True* the counter in byte 1 will be incremented before calculating the CRC.

e2e.p01.**e2e_p01_check**(*data: bytearray*, *data_id: int*, ***, *data_id_mode: int = E2E_P01_DATAID_BOTH*, *length: int = 0*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 1.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object starting with the CRC byte. This CRC byte will be updated inplace.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **data_id_mode** (*int*) – Mode of the data ID. Possible values are *E2E_P01_DATAID_BOTH*, *E2E_P01_DATAID_ALT*, *E2E_P01_DATAID_LOW* and *E2E_P01_DATAID_NIBBLE*.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns
True if CRC is valid, otherwise return *False*

e2e.p01.**E2E_P01_DATAID_BOTH: Final[int] = 0x00**

e2e.p01.**E2E_P01_DATAID_ALT: Final[int] = 0x01**

e2e.p01.**E2E_P01_DATAID_LOW: Final[int] = 0x02**

e2e.p01.**E2E_P01_DATAID_NIBBLE: Final[int] = 0x03**

1.1.2 Profile 02

e2e.p02.**e2e_p02_protect**(*data: bytearray*, *data_id_list: bytes*, ***, *length: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 2.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object starting with the CRC byte. This CRC byte will be updated inplace.
- **data_id_list** (*bytes*) – A bytes-like object of length 16 which is used to protect against masquerading.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:param bool increment_counter: If *True* the counter in byte 1 will be incremented before calculating the CRC.

e2e.p02.**e2e_p02_check**(*data: bytes*, *data_id_list: bytes*, ***, *length: int*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 2.

Parameters

- **data** – bytes-like object starting with the CRC byte.
- **data_id_list** – A bytes-like object of length 16 which is used to protect against masquerading.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:return: *True* if CRC is valid, otherwise return *False*

1.1.3 Profile 04

e2e.p04.**e2e_p04_protect**(*data: bytearray*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 4.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 32bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `12 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

- **increment_counter** (*bool*) – If *True* the counter will be incremented before calculating the CRC.

e2e.p04.**e2e_p04_check**(*data: bytearray*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 4.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object starting with the CRC byte.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `12 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns

True if CRC is valid, otherwise return *False*

1.1.4 Profile 05

e2e.p05.**e2e_p05_protect**(*data: bytearray*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 5.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `3 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.
- **increment_counter** (*bool*) – If *True* the counter will be incremented before calculating the CRC.

e2e.p05.**e2e_p05_check**(*data: bytes*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 5.

Parameters

- **data** – bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `3 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns

True if CRC is valid, otherwise return *False*

1.1.5 Profile 06

e2e.p06.**e2e_p06_protect**(*data: bytearray*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 6.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `5 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.
- **increment_counter** (*bool*) – If *True* the counter will be incremented before calculating the CRC.

e2e.p06.**e2e_p06_check**(*data: bytes*, *data_id: int*, ***, *length: int*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 6.

Parameters

- **data** – bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `5 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns

True if CRC is valid, otherwise return *False*

1.1.6 Profile 07

e2e.p07.**e2e_p07_protect**(*data: bytearray*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 7.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 32bit unsigned integer.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `20 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.
- **increment_counter** (*bool*) – If *True* the counter will be incremented before calculating the CRC.

e2e.p07.**e2e_p07_check**(*data: bytes*, *data_id: int*, ***, *length: int = 0*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 7.

Parameters

- **data** – bytes-like object.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 32bit unsigned integer.

- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `20 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns

True if CRC is valid, otherwise return *False*

1.1.7 Profile 11

e2e.p11.**e2e_p11_protect**(*data: bytearray*, *data_id: int*, ***, *data_id_mode: int = E2E_P11_DATAID_BOTH*, *length: int = 0*, *offset: int = 0*, *increment_counter: bool = True*) → None

Calculate CRC inplace according to AUTOSAR E2E Profile 11.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object. This CRC byte will be updated inplace.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **data_id_mode** (*int*) – This attribute describes the inclusion mode that is used to include the *data_id*. The possible inclusion modes are *E2E_P11_DATAID_BOTH* and *E2E_P11_DATAID_NIBBLE*.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.
- **increment_counter** (*bool*) – If *True* the counter in byte 1 will be incremented before calculating the CRC.

e2e.p11.**e2e_p11_check**(*data: bytearray*, *data_id: int*, ***, *data_id_mode: int = E2E_P11_DATAID_BOTH*, *length: int = 0*, *offset: int = 0*) → bool

Return `True` if CRC is correct according to AUTOSAR E2E Profile 11.

Parameters

- **data** (*bytearray*) – Mutable bytes-like object starting with the CRC byte. This CRC byte will be updated inplace.
- **data_id** (*int*) – A unique identifier which is used to protect against masquerading. The *data_id* is a 16bit unsigned integer.
- **data_id_mode** (*int*) – Mode of the data ID. Possible values are *E2E_P11_DATAID_BOTH* and *E2E_P11_DATAID_NIBBLE*.
- **length** (*int*) – Number of bytes to consider for CRC calculation. If `length == 0`, the full buffer length (`len(data)`) is used. Otherwise, `2 <= length <= len(data)` must hold.:param int offset: Byte offset of the E2E header.

Returns

True if CRC is valid, otherwise return *False*

e2e.p11.**E2E_P11_DATAID_BOTH: Final[int] = 0x00**

e2e.p11.**E2E_P11_DATAID_NIBBLE: Final[int] = 0x03**

1.2 CRC Functions

1.2.1 8-bit SAE J1850 CRC Calculation

e2e.crc.**calculate_crc8**(*data: bytes*, *start_value: int = 0xFF*, *first_call: bool = True*) → int

8-bit SAE J1850 CRC Calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns

CRC value

e2e.crc.**CRC8_INITIAL_VALUE: Final[int] = 0xFF**

e2e.crc.**CRC8_XOR_VALUE: Final[int] = 0xFF**

e2e.crc.**CRC8_CHECK: Final[int] = 0x4B**

e2e.crc.**CRC8_MAGIC_CHECK: Final[int] = 0xC4**

1.2.2 8-bit 0x2F polynomial CRC Calculation

e2e.crc.**calculate_crc8_h2f**(*data: bytes*, *start_value: int = 0xFF*, *first_call: bool = True*) → int

8-bit 0x2F polynomial CRC Calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns

CRC value

e2e.crc.**CRC8H2F_INITIAL_VALUE: Final[int] = 0xFF**

e2e.crc.**CRC8H2F_XOR_VALUE: Final[int] = 0xFF**

e2e.crc.**CRC8H2F_CHECK: Final[int] = 0xDF**

e2e.crc.**CRC8H2F_MAGIC_CHECK: Final[int] = 0x42**

1.2.3 16-bit CCITT-FALSE CRC16

e2e.crc.**calculate_crc16**(*data: bytes*, *start_value: int = 0xFFFF*, *first_call: bool = True*) → int

16-bit CCITT-FALSE CRC16

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns
CRC value

e2e.crc.**CRC16_INITIAL_VALUE: Final[int] = 0xFFFF**

e2e.crc.**CRC16_XOR_VALUE: Final[int] = 0x0000**

e2e.crc.**CRC16_CHECK: Final[int] = 0x29B1**

e2e.crc.**CRC16_MAGIC_CHECK: Final[int] = 0x0000**

1.2.4 16-bit 0x8005 polynomial CRC calculation

e2e.crc.**calculate_crc16_arc**(*data: bytes*, *start_value: int = 0x0000*, *first_call: bool = True*) → int

16-bit 0x8005 polynomial CRC calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns
CRC value

e2e.crc.**CRC16ARC_INITIAL_VALUE: Final[int] = 0x0000**

e2e.crc.**CRC16ARC_XOR_VALUE: Final[int] = 0x0000**

e2e.crc.**CRC16ARC_CHECK: Final[int] = 0xBB3D**

e2e.crc.**CRC16ARC_MAGIC_CHECK: Final[int] = 0x0000**

1.2.5 32-bit Ethernet CRC Calculation

e2e.crc.**calculate_crc32**(*data: bytes*, *start_value: int = 0xFFFFFFFF*, *first_call: bool = True*) → int

32-bit Ethernet CRC Calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns
CRC value

e2e.crc.**CRC32_INITIAL_VALUE**: Final[int] = 0xFFFFFFFF

e2e.crc.**CRC32_XOR_VALUE**: Final[int] = 0xFFFFFFFF

e2e.crc.**CRC32_CHECK**: Final[int] = 0xCBF43926

e2e.crc.**CRC32_MAGIC_CHECK**: Final[int] = 0xDEBB20E3

1.2.6 32-bit 0xF4ACFB13 polynomial CRC calculation

e2e.crc.**calculate_crc32_p4**(*data: bytes*, *start_value: int = 0xFFFFFFFF*, *first_call: bool = True*) → int

32-bit 0xF4ACFB13 polynomial CRC calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns

CRC value

e2e.crc.**CRC32P4_INITIAL_VALUE**: Final[int] = 0xFFFFFFFF

e2e.crc.**CRC32P4_XOR_VALUE**: Final[int] = 0xFFFFFFFF

e2e.crc.**CRC32P4_CHECK**: Final[int] = 0x1697D06A

e2e.crc.**CRC32P4_MAGIC_CHECK**: Final[int] = 0x904CDDBF

1.2.7 64-bit 0x42F0E1EBA9EA3693 polynomial CRC calculation

e2e.crc.**calculate_crc64**(*data: bytes*, *start_value: int = 0xFFFFFFFFFFFFFFFF*, *first_call: bool = True*) → int

64-bit 0x42F0E1EBA9EA3693 polynomial CRC calculation

Parameters

- **data** (*bytes*) – bytes-like object which contains the data for CRC calculation
- **start_value** (*int*) – First CRC of the algorithm (ignored when *first_call* is *True*). In a sequence, this is expected to be the return value of the previous function call.
- **first_call** (*bool*) – *True* if this is the first call of a sequence or an individual function call. *False* if this is a subsequent call in a sequence.

Returns

CRC value

e2e.crc.**CRC64_INITIAL_VALUE**: Final[int] = 0xFFFFFFFFFFFFFFFF

e2e.crc.**CRC64_XOR_VALUE**: Final[int] = 0xFFFFFFFFFFFFFFFF

e2e.crc.**CRC64_CHECK**: Final[int] = 0x995DC9BBDF1939FA

e2e.crc.**CRC64_MAGIC_CHECK**: Final[int] = 0x49958C9ABD7D353F

CHAPTER

TWO

DESCRIPTION

This library provides fast C implementations of the E2E CRC algorithms and E2E profiles.

CHAPTER

THREE

INSTALLATION

You can install `autosar-e2e` from PyPI:

```
python -m pip install autosar-e2e
```

CHAPTER

FOUR

USAGE

Listing 1: CRC example

```
import e2e

crc: int = e2e.crc.calculate_crc8_h2f(b"\x00\x00\x00\x00")
```

Listing 2: E2E P02 example

```
import e2e

# create data
data = bytearray(b"\x00" *  8)
data_id_list = b"\x00" * 16

# increment counter and calculate CRC inplace
e2e.p02.e2e_p02_protect(data, data_id_list, increment_counter=True)

# check CRC
crc_correct: bool = e2e.p02.e2e_p02_check(data, data_id_list)
```

CHAPTER

FIVE

TEST

Run the tests with:

```
uvx tox
```

CHAPTER

SIX

BUILD

Build `autosar-e2e` with:

```
uv build
uvx twine check dist/*
```

CHAPTER

SEVEN

LICENSE

`autosar-e2e` is distributed under the terms of the MIT license.

INDEX

C

E