
cyclarity-in-vehicle-sdk

Release 1.1.0

Cymotive

May 11, 2025

CONTENTS

1	In-Vehicle SDK Package	1
1.1	Features	1
1.2	Installation	2
1.3	Usage	2
2	Communication Objects	5
2.1	cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan.CanCommunicatorSocketCan	5
2.2	cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer2RawSocket	7
2.3	cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer3RawSocket	9
2.4	cyclarity_in_vehicle_sdk.communication.ip.tcp.tcp.TcpCommunicator	11
2.5	cyclarity_in_vehicle_sdk.communication.ip.udp.udp.UdpCommunicator	12
2.6	cyclarity_in_vehicle_sdk.communication.ip.udp.multicast.MulticastCommunicator	14
2.7	cyclarity_in_vehicle_sdk.communication.isotp.impl.isotp_communicator.IsoTpCommunicator	15
2.8	cyclarity_in_vehicle_sdk.communication.doip.doip_communicator.DoipCommunicator	17
3	Cyclarity Data Models	19
3.1	UDS models	19
3.2	DoIP models	25
3.3	SOME/IP models	27
4	Protocol specifics APIs	29
4.1	cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils	29
4.2	cyclarity_in_vehicle_sdk.protocol.someip.impl.someip_utils.SomeipUtils	34
4.3	cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_utils.DoipUtils	35
5	Shell Devices	39
5.1	cyclarity_in_vehicle_sdk.utils.shell_device.impl.adb_device_shell.AdbDeviceShell	39
5.2	cyclarity_in_vehicle_sdk.utils.shell_device.impl.serial_device_shell.SerialDeviceShell	40
5.3	cyclarity_in_vehicle_sdk.utils.shell_device.impl.ssh_device_shell.SshDeviceShell	42
6	Plugins	45
6.1	cyclarity_in_vehicle_sdk.plugin.crash_detection.session_change_detector.SessionChangeCrashDetector	45
6.2	cyclarity_in_vehicle_sdk.plugin.crash_detection.unresponded_tp_crash_detector.UnrespondedTesterPresentCrashDetector	46
6.3	cyclarity_in_vehicle_sdk.plugin.recover_ecu.uds_ecu_recover.UdsEcuRecoverPlugin	47
6.4	cyclarity_in_vehicle_sdk.plugin.reset.relay.relay_reset_plugin.RelayResetPlugin	47
6.5	cyclarity_in_vehicle_sdk.plugin.reset.uds_ecu_reset.uds_ecu_reset.UdsBasedEcuResetPlugin	48
7	Configuration Management	51
7.1	ConfigurationManager	51
7.2	Configuration Management - Models	52
7.3	Configuration Management - Actions	59

IN-VEHICLE SDK PACKAGE

This package provides the In-Vehicle SDK, offering a range of functionalities to support communication and operations with in-vehicle systems.

1.1 Features

The In-Vehicle SDK package includes the following interfaces and implementations:

1. Communication

1. **CommunicatorBase**: Provides the capability to send and receive byte data over various protocols. The following implementations are available:
 - `TcpCommunicator`
 - `UdpCommunicator`
 - `MulticastCommunicator`
 - `IsoTpCommunicator`
 - `DoipCommunicator`
2. **RawSocketCommunicatorBase**: Offers send, receive, and srp (send and receive answer) operations for `py_pcapplusplus.Packet` types. The following implementations are available:
 - `Layer2RawSocket`
 - `Layer3RawSocket`
 - `WiFiRawSocket`
3. **CanCommunicatorBase**: Exposes the python-can functionality, offering operations like send, receive, sniff, and more. The following implementation is available:
 - `CanCommunicatorSocketCan` - A specific implementation for the socketcan driver
2. **DoipUtils**: A utility library for performing Diagnostic over IP (DoIP) operations, such as vehicle identity requests, routing activation, and more.
3. **UdsUtilsBase**: Used for performing Unified Diagnostic Services (UDS) operations, such as ECU reset, read DIDs, session change, and more. The following implementation is available:
 - `UdsUtils` - Can be initialized to work over DoIP/ISO-TP
4. **IDeviceShell**: Allows for the execution of shell commands. The following implementations are available:
 - `AdbDeviceShell`
 - `SerialDeviceShell`

- SshDeviceShell
5. **SomeipUtils**: A utility library for SOME/IP operations, allowing the receive and parse services, and in these services invoke methods and subscribe to eventgroups
 6. **Plugins**:
 - SessionChangeCrashDetector: a plugin that detects ECU crash based on UDS session change
 - UnrespondedTesterPresentCrashDetector: a plugin that detects ECU crash based on UDS TP that is not being responded
 - UdsEcuRecoverPlugin: a plugin responsible of recovering the ECU back to predefined UDS state - session and elevation
 - RelayResetPlugin: a plugin that resets a device via relay
 - UdsBasedEcuResetPlugin: a plugin that resets a device via UDS ECU Reset
 7. **ConfigurationManager**: An API allowing to perform configuration of the IOT Device.
 - configure_actions(action/s) - can perform the following configuration actions on the device:
 1. IpAddAction - add an IP to an Ethernet interface, and optionally configure a route for this IP.
 2. IpRemoveAction - remove an existing IP from an Ethernet interface.
 3. CanConfigurationAction - configure CAN interface parameters. e.g. bitrate, sample-point, cc-len8-dlc flag and state.
 4. EthInterfaceConfigurationAction - configure the Ethernet interface: mtu, state and flags.
 5. WifiConnectAction - connect to a WiFi access point
 - get_device_configuration() - retrieves the current device configurations:
 1. Ethernet interface configuration: state, IPs, flags and MTU.
 2. CAN interface configurations: state, bitrate, sample-point and cc-len8-dlc flag.
 3. The available WiFi access points.

1.2 Installation

You can install the In-Vehicle SDK package using pip: `pip install cyclarity-in-vehicle-sdk`

1.3 Usage

Example for importing and using CanCommunicatorSocketCan for sending a Message

```
from cyclarity_in_vehicle_sdk.communication.can.base.can_communicator_base import CanMessage
from cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan import CanCommunicatorSocketCan

canmsg = CanMessage(
    arbitration_id=0x123,
    is_extended_id = False,
    is_rx=False,
    data=b"\x00" * 8,
    is_fd=False,
```

(continues on next page)

(continued from previous page)

```
        bitrate_switch=False,  
    )  
  
socket = CanCommunicatorSocketCan(channel="vcan0", support_fd=True)  
with socket:  
    socket.send(can_msg=canmsg)
```


COMMUNICATION OBJECTS

<code>cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan.CanCommunicatorSocketCan</code>	This class handles the communication over the CAN bus using the SocketCAN interface."
<code>cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer2RawSocket</code>	This class handles layer 2 raw socket communication.
<code>cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer3RawSocket</code>	Layer 3 raw socket for communicator
<code>cyclarity_in_vehicle_sdk.communication.ip.tcp.tcp.TcpCommunicator</code>	TCP Communicator.
<code>cyclarity_in_vehicle_sdk.communication.ip.udp.udp.UdpCommunicator</code>	A class used for UDP communication over IP networks.
<code>cyclarity_in_vehicle_sdk.communication.ip.udp.multicast.MulticastCommunicator</code>	A class used for multicast communication over IP networks.
<code>cyclarity_in_vehicle_sdk.communication.isotp.impl.isotp_communicator.IsoTpCommunicator</code>	This class handles communication over IsoTP protocol.
<code>cyclarity_in_vehicle_sdk.communication.doip.doip_communicator.DoipCommunicator</code>	This class handles communication over DoIP protocol.

2.1 `cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan`

pydantic model `cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan.CanCommunicatorSocketCan`

This class handles the communication over the CAN bus using the SocketCAN interface."

Fields

- `blacklist_ids` (`set[int]`)
- `channel` (`str`)
- `support_fd` (`bool`)

field `blacklist_ids`: `set[int] = {}`

Incoming CAN IDs to ignore

field `channel`: `str [Required]`

Name of CAN interface to work with. (e.g. can0, vcan0, etc...)

field `support_fd`: `bool [Required]`

CAN bus supports CAN-FD.

add_to_blacklist(*canids*)

adds can IDs to a list of blacklist IDs to be ignore when sniffing or receiving

Parameters

canids (*Sequence[int]*) – CAN IDs to be added to the blacklist

close()

Closes the communicator.

Return type

None

get_bus()

get the underling CAN bus

Returns

the CAN bus implementation - should be an implementation of BusABC

Return type

Type[BusABC]

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type

None

open()

Opens the communicator. this method must be called before usage.

Return type

None

receive(*timeout=None*)

receive a CAN message over the channel

Parameters

timeout (*Optional[float], optional*) – timeout in seconds to try and receive. None means indefinably.

Returns

CAN message if a message was received, None otherwise.

Return type

Optional[CanMessage]

send(*can_msg, timeout=None*)

Transmit a message to the CAN bus.

Parameters

- **can_msg** (*CanMessage*) – CAN message in the python-can format *CanMessage*
- **timeout** (*Optional[float], optional*) – time out in seconds. Defaults to None.

send_periodically(*msgs, period, duration=None*)

Send periodically CAN message(s)

Parameters

- **msgs** (*Union[CanMessage, Sequence[CanMessage]]*) – single message or sequence of messages to be sent periodically

- **period** (*float*) – time period in seconds between sending of the message(s)
- **duration** (*Optional[float], optional*) – duration time in seconds tp be sending the message(s) periodically. None means indefinitely.

sniff(*sniff_time*)

sniff CAN messages from the channel for specific time

Parameters

sniff_time (*float*) – time in seconds to be sniffing the channel

Returns

list of CAN messages sniffed, None if none was sniffed

Return type

Optional[list[CanMessage]]

2.2 cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer2RawSocket

pydantic model `cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer2RawSocket`

This class handles layer 2 raw socket communication.

Fields

- *if_name* (*str*)

field if_name: `str` [Required]

Name of ethernet interface to work with. (e.g. eth0, eth1 etc...)

close()

Close the raw socket.

Returns

True if successful, False otherwise.

Return type

bool

is_open()

Check if the raw socket is open.

Returns

True if the socket is open, False otherwise.

Return type

bool

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Return type

None

open()

Open the raw socket for communication.

Returns

True if successful, False otherwise.

Return type

bool

receive(*timeout=2*)

read a single packet from the socket

Parameters

timeout (*float*) – timeout in seconds for the operation, 0 for blocking receive.

Returns

the read packet, None if timeout reached.

Return type

Packet | None

receive_answer(*is_answer, timeout=2*)

sniff communication and return a packet that satisfy the “is_answer” callback.

Parameters

- **(Callable[[Packet])** – A callback that receives a packet and returns True if this packet is the answer looking for.
- **bool]]** – A callback that receives a packet and returns True if this packet is the answer looking for.
- **timeout** (*float*) – The duration of the sniffing to locate the answer packet.

Returns

The first packet that satisfy the “is_answer” callback, None if not found.

Return type

Packet | None

receive_answers(*is_answer, timeout=2*)

Read a multiple packets and returns all packets that satisfy the “is_answer” callback provided.

Parameters

- **is_answer** (*Callable[[Packet], bool]*) – A callback that receives a packet and returns True if this packet is the answer looking for.
- **timeout** (*int*) – The duration of the sniffing to locate the answer packets.

Returns

All packets received that satisfy the “is_answer” callback.

Return type

list[Packet]

send_packet(*packet*)

Send a packet over the raw socket.

Parameters

packet (*Packet*) – The Packet to be sent.

Returns

True if the packet was sent successfully, False otherwise.

Return type

bool

send_packets(*packets*)

Send multiple packets over the raw socket.

Parameters

packets (*Sequence[Packet]*) – The list of Packets to be sent.

Returns

True if the packets were sent successfully, False otherwise.

Return type

bool

send_receive_packet(*packet, is_answer, timeout=2*)

send packet or a sequence of packets and read an answer The answer is one packet that satisfy the “is_answer” callback provided.

Note: This function uses the implementation of ‘send_receive_packets’, Optionally override this function to have a better implementation (stop after the first valid packet arrives).

Parameters

- **packet** (*Packet | Sequence[Packet] | None*) – the packet/packets to send. None to skip the sending operation.
- **is_answer** (*Callable[[Packet], bool]*) – callback that receives a packet and returns True if this packet is the answer to sent one
- **timeout** (*int*) – timeout for the operation

Returns

The first packet that satisfy the “is_answer” callback, None if not found.

Return type

Packet | None

send_receive_packets(*packet, is_answer, timeout=2*)

send packet or a sequence of packets and read a multiple packets answer The answer is a list of packets that satisfy the “is_answer” callback provided.

Parameters

- **packet** (*Packet | Sequence[Packet] | None*) – the packet/packets to send. None to skip the sending operation.
- **is_answer** (*Callable[[Packet], bool]*) – callback that receives a packet and returns True if this packet is the answer to sent one
- **timeout** (*int*) – timeout for the operation

Returns

All packets received that satisfy the “is_answer” callback.

Return type

list[Packet]

2.3 cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer3RawSocket

pydantic model `cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer3RawSocket`

Layer 3 raw socket for communicator

Fields

- *if_name* (*str*)
- *ip_version* (*cyclarity_in_vehicle_sdk.communication.ip.base.ip_communicator_base.IpVersion*)

field if_name: str [Required]

Name of ethernet interface to work with. (e.g. eth0, eth1 etc...)

field ip_version: IpVersion [Required]

IP version. IPv4/IPv6

close()

Close the raw socket.

Returns

True if successful, False otherwise.

Return type

bool

is_open()

inform the state of the raw socket

Returns

True if the socket is open and ready for send/receive operations, False otherwise.

Return type

bool

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type

None

open()

Open the raw socket for communication.

Returns

True if successful, False otherwise.

Return type

bool

receive(*timeout=2*)

read a single packet from the socket

Parameters

timeout (*float*) – timeout in seconds for the operation, 0 for blocking receive.

Returns

the read packet, None if timeout reached.

Return type

Packet | None

send_packet(*packet*)

send a packet to the raw socket

Parameters

packet (*Packet*) – packet to send.

Returns

True if sent successfully, False otherwise

Return type

bool

send_receive_packets(*packet, is_answer, timeout*)

send packet or a sequence of packets and read a multiple packets answer The answer is a list of packets that satisfy the “is_answer” callback provided.

Parameters

- **packet** (*Packet | Sequence[Packet] | None*) – the packet/packets to send. None to skip the sending operation.
- **is_answer** (*Callable[[Packet], bool]*) – callback that receives a packet and returns True if this packet is the answer to sent one
- **timeout** (*int*) – timeout for the operation

Returns

All packets received that satisfy the “is_answer” callback.

Return type

list[Packet]

2.4 cyclarity_in_vehicle_sdk.communication.ip.tcp.tcp.TcpCommunicator

pydantic model `cyclarity_in_vehicle_sdk.communication.ip.tcp.tcp.TcpCommunicator`

TCP Communicator. The class provides methods to open, close, send, receive data over a TCP connection.

Fields**close()**

Close the TCP socket.

Returns

True if successful, False otherwise.

Return type

bool

connect()

Connects the socket to the destination IP and port.

Returns

True on successful completion.

Return type

bool

get_type()

get the communicator type

Returns

enum type of the communicator

Return type

CommunicatorType

is_open()

inform the state of the TCP socket

Returns

True if the socket is open and ready for send/receive operations, False otherwise.

Return type

bool

model_post_init(*_ModelMetaclass__context: Any*) → NoneWe need to both initialize private attributes and call the user-defined `model_post_init` method.**Return type**

None

open()

Open the TCP socket for communication.

Returns

True if successful, False otherwise.

Return type

bool

recv(*recv_timeout=0, size=4096*)

Receives data from the socket.

Parameters

- **recv_timeout** (*float, optional*) – The optional timeout in seconds for receiving data.. Defaults to 0.
- **size** (*int, optional*) – The maximum amount of data to receive.

Returns

The received bytes, or an empty bytes object if an exception occurred.

Return type

bytes

send(*data, timeout=None*)

Sends data over the socket.

Parameters

- **data** (*bytes*) – The bytes to send.
- **timeout** (*Optional[float], optional*) – The optional timeout in seconds for sending data.

Returns

The number of bytes sent, or 0 if an exception occurred.

Return type

int

2.5 cyclarity_in_vehicle_sdk.communication.ip.udp.udp.UdpCommunicator

pydantic model `cyclarity_in_vehicle_sdk.communication.ip.udp.udp.UdpCommunicator`

A class used for UDP communication over IP networks.

Fields**close**()

Closes the socket.

Returns

A boolean indicating if the socket was successfully closed.

Return type

bool

get_type()

get the communicator type

Returns

enum type of the communicator

Return type

CommunicatorType

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type

None

open()

Opens the socket. :returns: A boolean indicating if the socket was successfully opened. :rtype: bool

receive_from(*size=4096, recv_timeout=0*)

Receives data from the socket

Parameters

- **size** (*int, optional*) – The size of the data to be received.
- **recv_timeout** (*int, optional*) – The timeout for the receive operation.

Returns

The data received and the sender's IP address.

Return type

tuple[bytes, IPvAnyAddress]

recv(*recv_timeout=0, size=4096*)

Receives data from the socket.

Parameters

- **recv_timeout** (*float, optional*) – The timeout for the receive operation.
- **size** (*int, optional*) – The size of the data to be received.

Returns

The data received.

Return type

bytes

send(*data, timeout=None*)

Sends data to the specified IP address and port.

Parameters

- **data** (*bytes*) – data The data to be sent.
- **timeout** (*Optional[float], optional*) – The timeout for the send operation.

Returns

The number of bytes sent.

Return type

int

send_to(*target_ip*, *data*)

Sends data to a specific IP address and port.

Parameters

- **target_port** (*int*) – The target port.
- **target_ip** (*IPvAnyAddress*) – The target IP address.
- **data** (*bytes*) – The data to be sent.

Returns

The number of bytes sent.

Return type

int

2.6 cyclarity_in_vehicle_sdk.communication.ip.udp.multicast.MulticastCommunicator

pydantic model

`cyclarity_in_vehicle_sdk.communication.ip.udp.multicast.MulticastCommunicator`

A class used for multicast communication over IP networks.

Fields

- *interface_name* (*str* | *None*)

field interface_name: `Optional[str] = None`

Network interface name - needed incase of IPv6 multicast

close()

Closes the socket.

Returns

A boolean indicating if the socket was successfully closed.

Return type

bool

get_type()

get the communicator type

Returns

enum type of the communicator

Return type

CommunicatorType

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Return type

None

open()

Opens the socket. :returns: A boolean indicating if the socket was successfully opened. :rtype: bool

receive_from(*size=4096*, *recv_timeout=0*)

Receives data from a specific IP address and port.

Parameters

- **size** (*int*, *optional*) – The size of the data to be received.
- **recv_timeout** (*int*, *optional*) – The timeout for the receive operation.

Returns

The data received and the sender's IP address.

Return type

tuple[bytes, IPvAnyAddress]

recv(*recv_timeout=0*, *size=4096*)

Receives data from the multicast group.

Parameters

- **recv_timeout** (*float*, *optional*) – The timeout for the receive operation.
- **size** (*int*, *optional*) – The size of the data to be received.

Returns

The data received.

Return type

bytes

send(*data*, *timeout=None*)

Sends data to the multicast group.

Parameters

- **data** (*bytes*) – data The data to be sent.
- **timeout** (*Optional[float]*, *optional*) – The timeout for the send operation.

Returns

The number of bytes sent.

Return type

int

send_to(*target_port*, *target_ip*, *data*)

Sends data to a specific IP address and port.

Parameters

- **target_port** (*int*) – The target port.
- **target_ip** (*IPvAnyAddress*) – The target IP address.
- **data** (*bytes*) – The data to be sent.

Returns

The number of bytes sent.

Return type

int

2.7 cyclarity_in_vehicle_sdk.communication.isotp.impl.isotp_communicator.IsoTpCommunicator

pydantic model

`cyclarity_in_vehicle_sdk.communication.isotp.impl.isotp_communicator.IsoTpCommunicator`

This class handles communication over IsoTP protocol.

Fields

- *bitrate_switch* (*bool* | *None*)
- *can_communicator* (*cyclarity_in_vehicle_sdk.communication.can.impl.can_communicator_socketcan.CanCommunicatorSocketCan*)
- *can_fd* (*bool* | *None*)
- *padding_byte* (*int* | *None*)
- *rxid* (*int*)
- *txid* (*int*)

field bitrate_switch: `Optional[bool] = False`

BRS, defaults to False

field can_communicator: `CanCommunicatorSocketCan` [Required]

CAN Communicator

field can_fd: `Optional[bool] = False`

whether it is can FD, defaults to False

field padding_byte: `Optional[int] = None`

Optional byte to pad TX messages with, defaults to None meaning no padding, should be in range 0x00-0xFF

Constraints

- *ge* = 0
- *le* = 255

field rxid: `int` [Required]

Receive CAN id.

field txid: `int` [Required]

Transmit CAN id.

close()

Closes the socket.

Returns

A boolean indicating if the socket was successfully closed.

Return type

`bool`

get_type()

get the communicator type

Returns

enum type of the communicator

Return type

`CommunicatorType`

model_post_init(*_ModelMetaclass__context: Any*) → *None*

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Return type

`None`

open()

Opens the socket. :returns: A boolean indicating if the socket was successfully opened. :rtype: bool

recv(recv_timeout)

Receives data from the socket.

Parameters

- **recv_timeout** (*float, optional*) – The timeout for the receive operation.
- **size** (*int, optional*) – The size of the data to be received.

Returns

The data received.

Return type

bytes

send(data, timeout=1)

sends bytes over the communication layer

Parameters

- **data** (*bytes*) – data to send in bytes format
- **timeout** (*Optional[float]*) – timeout in seconds for send operation. defaults to None

Returns

amount of bytes sent

Return type

int

set_address(address)

Set the address of the communicator.

Parameters

address (*Address*) – The address to be set.

teardown()

Close the communicator.

2.8 cyclarity_in_vehicle_sdk.communication.doip.doip_communicator.DoipCommunicator

pydantic model

`cyclarity_in_vehicle_sdk.communication.doip.doip_communicator.DoipCommunicator`

This class handles communication over DoIP protocol.

Fields

- `client_logical_address` (*int*)
- `routing_activation_needed` (*bool*)
- `target_logical_address` (*int*)
- `tcp_communicator` (*cyclarity_in_vehicle_sdk.communication.ip.tcp.tcp.TcpCommunicator*)

field `client_logical_address`: `int` [Required]

field `routing_activation_needed`: `bool` [Required]

field `target_logical_address`: `int` [Required]

field `tcp_communicator`: `TcpCommunicator` [Required]

close()

Closes the communicator.

Return type
`bool`

get_type()

Get the type of the communicator.

Returns
`CommunicatorType.DOIP`

Return type
`CommunicatorType`

open()

Open the communicator.

Returns
True on successful initialization, False otherwise.

Return type
`bool`

recv(*recv_timeout*)

Receive data from the target.

Parameters
recv_timeout (*float*) – Time to wait for a response.

Returns
Received data.

Return type
`bytes`

send(*data*, *timeout=1*)

Send data to the target.

Parameters

- **data** (*bytes*) – Data to be sent.
- **timeout** (*Optional[float]*, *optional*) – Timeout for the send operation in seconds. Defaults to 1.

Returns
Number of bytes sent.

Return type
`int`

CYCLARITY DATA MODELS

3.1 UDS models

class cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.**AuthenticationAction**(*value*)

Types of authentication actions

pydantic model

cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.AuthenticationConfigurationParams

Model for the parameters of the AuthenticationConfiguration action

Fields

- *param_type* (*Literal*['AuthenticationConfigurationParams'])

field *param_type*: *Literal*['AuthenticationConfigurationParams'] =
'AuthenticationConfigurationParams'

authentication_action()

Return type

AuthenticationAction

pydantic model

cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.AuthenticationParamsBase

abstract authentication_action()

Return type

AuthenticationAction

classmethod *get_non_abstract_subclasses()*

Return type

list[*Type*]

class cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.**AuthenticationReturnParameter**(*value*)

Model defining the authentication return codes

pydantic model **cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.BaseAPCEParams**

Base class defining parameters for UDS Authentication based on asymmetric public certificate exchange

Fields

- *asym_padding_type* (*cyclarity_in_vehicle_sdk.utils.crypto.models.AsymmetricPaddingType*)

- *certificate_client* (bytes | *cyclarity_sdk.sdk_models.models.CyclarityFile*)
- *communication_configuration* (int)
- *hash_algorithm* (*cyclarity_in_vehicle_sdk.utils.crypto.models.HashingAlgorithm*)
- *private_key_der* (bytes | *cyclarity_sdk.sdk_models.models.CyclarityFile*)

field asym_padding_type: *AsymmetricPaddingType* [Required]

The padding type to use in signature creation for challenge signing

field certificate_client: *Union[Annotated[bytes], CyclarityFile]* [Required]

The client's certificate to send to the server for authentication

field communication_configuration: int = 0

Configuration information about how to proceed with security in further diagnostic communication after the Authentication (vehicle manufacturer specific)

Constraints

- ge = 0
- le = 255

field hash_algorithm: *HashingAlgorithm* [Required]

The hashing algorithm to use in signature creation for challenge signing

field private_key_der: *Union[Annotated[bytes], CyclarityFile]* [Required]

The private key for authentication in DER format

pydantic model *cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.DID_INFO*

Model containing information regarding a UDS Data Identifier

Fields

- *accessible* (bool)
- *current_data* (str | None)
- *did* (int)
- *maybe_supported_error* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ERROR_CODE_AND_NAME* | None)
- *name* (str | None)

field accessible: bool [Required]

field current_data: *Optional[str]* = None

field did: int [Required]

field maybe_supported_error: *Optional[ERROR_CODE_AND_NAME]* = None

The error code if there is uncertainty that this DID is supported

field name: *Optional[str]* = None

pydantic model

cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.DeAuthenticateParams

Model for the parameters of the DeAuthenticate action

Fields

- *param_type* (*Literal['DeAuthenticateParams']*)

field param_type: *Literal['DeAuthenticateParams']* = 'DeAuthenticateParams'**authentication_action()****Return type***AuthenticationAction***pydantic model** cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ELEVATION_INFO

Model for defining the needed elevation information for a UDS session

Fields

- *need_elevation* (*bool | None*)
- *security_algorithm* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SECURITY_ALGORITHM_XOR | cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SECURITY_ALGORITHM_PIN | None*)

field need_elevation: *Optional[bool]* = None

Whether this session requires elevation

field security_algorithm: *Union[SECURITY_ALGORITHM_XOR, SECURITY_ALGORITHM_PIN, None]* = None

The security elevation algorithm

pydantic model

cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ERROR_CODE_AND_NAME

Model defining the error code and its name

Fields

- *code* (*int*)
- *code_name* (*str*)

field code: *int* [Required]

Error code number

field code_name: *str* [Required]

Error code name

pydantic model cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ROUTINE_INFO

Model containing information regarding a UDS routine

Fields

- *operations* (*list[cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ROUTINE_OPERATION_INFO]*)
- *routine_id* (*int*)

field operations: *list[ROUTINE_OPERATION_INFO]* [Required]**field routine_id:** *int* [Required]

pydantic model`cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ROUTINE_OPERATION_INFO`

Model containing information regarding a UDS routine subfunction

Fields

- *accessible* (*bool*)
- *control_type* (*int*)
- *maybe_supported_error* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ERROR_CODE_AND_NAME* | *None*)
- *routine_status_record* (*str* | *None*)

field accessible: `bool` [Required]

field control_type: `int` [Required]

field maybe_supported_error: `Optional[ERROR_CODE_AND_NAME]` = `None`

The error code if there is uncertainty that this routine control type is supported

field routine_status_record: `Optional[str]` = `None`

Additional data associated with the response.

pydantic model`cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SECURITY_ALGORITHM_BASE`

Base model for security access algorithms

Fields

- *key_subfunction* (*int* | *None*)
- *seed_subfunction* (*int* | *None*)

field key_subfunction: `Optional[int]` = `None`

The subfunction for the send key operation

field seed_subfunction: `Optional[int]` = `None`

The subfunction for the get seed operation

pydantic model`cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SECURITY_ALGORITHM_PIN`

Model for PIN based security access

Fields

- *pin* (*int*)

field pin: `int` [Required]

Integer value to be added to the seed for security key generation

pydantic model`cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SECURITY_ALGORITHM_XOR`

Model for XOR based security access

Fields

- *xor_val* (*int*)

field xor_val: `int` [Required]

Integer value to XOR the seed with for security key generation

pydantic model `cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SERVICE_INFO`

Model containing information regarding a UDS service

Fields

- *accessible* (*bool*)
- *error* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ERROR_CODE_AND_NAME* | *None*)
- *name* (*str*)
- *sid* (*int*)

field accessible: `bool = False`

Whether this UDS service is accessible

field error: `Optional[ERROR_CODE_AND_NAME] = None`

The error code if exists

field name: `str [Required]`

The name of the UDS service

field sid: `int [Required]`

The SID of the UDS service

pydantic model `cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SESSION_ACCESS`

Model containing information regarding how to access a UDS session

Fields

- *elevation_info* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ELEVATION_INFO* | *None*)
- *id* (*int*)

field elevation_info: `Optional[ELEVATION_INFO] = None`

Elevation info for this UDS session, if needed

field id: `int [Required]`

ID of this UDS session

pydantic model `cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SESSION_INFO`

Model containing information regarding a UDS session

Fields

- *accessible* (*bool*)
- *elevation_info* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.ELEVATION_INFO* | *None*)
- *route_to_session* (*list[cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SESSION_ACCESS]*)

field accessible: `bool = False`

Whether this UDS session is accessible

field elevation_info: `Optional[ELEVATION_INFO] = None`

Elevation info for this UDS session

```
field route_to_session: list[SESSION_ACCESS] = []
```

The UDS session route to reach this session

pydantic model

```
cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.TransmitCertificateParams
```

Model defining the parameters for UDS Authentication - Transmit Certificate

Fields

- *certificate_data* (bytes)
- *certificate_evaluation_id* (int)
- *param_type* (Literal['TransmitCertificateParams'])

```
field certificate_data: Annotated[bytes] [Required]
```

The Certificate to verify

Constraints

- **func** = <function <lambda> at 0x7fc4ab5951b0>
- **return_type** = PydanticUndefined
- **when_used** = always
- **json_schema** = {'type': 'string'}

```
field certificate_evaluation_id: int [Required]
```

Optional unique ID to identify the evaluation type of the transmitted certificate

Constraints

- **ge** = 0
- **le** = 65535

```
field param_type: Literal['TransmitCertificateParams'] =  
'TransmitCertificateParams'
```

```
authentication_action()
```

Return type

AuthenticationAction

```
class cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.UdsSid(value)
```

The service IDs standardized by UDS.

For additional information, see https://en.wikipedia.org/wiki/Unified_Diagnostic_Services

```
class cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.UdsStandardVersion(value)
```

Model defining the UDS standard versions

pydantic model

```
cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.UnidirectionalAPCEParams
```

Model defining the parameters for UDS Authentication based on unidirectional asymmetric public certificate exchange

Fields

- *param_type* (Literal['UnidirectionalAPCEParams'])

```
field param_type: Literal['UnidirectionalAPCEParams'] = 'UnidirectionalAPCEParams'
```

`authentication_action()`

Return type

AuthenticationAction

3.2 DoIP models

pydantic model `cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_ENTITY_STATUS`

Model containing information regarding a DoIP entity status response

Fields

- *currently_open_sockets* (*int*)
- *max_concurrent_sockets* (*int*)
- *max_data_size* (*int*)
- *node_type* (*int*)

field `currently_open_sockets: int [Required]`

Currently open sockets

field `max_concurrent_sockets: int [Required]`

Max. concurrent sockets

field `max_data_size: int [Required]`

Max. data size

field `node_type: int [Required]`

Node type - DoIP node or a DoIP gateway

pydantic model

`cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_ROUTING_ACTIVATION`

Model containing information regarding a DoIP routing activation response

Fields

- *response_code* (*int*)
- *source_logical_address* (*int*)
- *src_addr_range_desc* (*str*)

field `response_code: int [Required]`

Routing activation response code

field `source_logical_address: int [Required]`

Logical address of client DoIP entity

field `src_addr_range_desc: str [Required]`

Description of the source address

pydantic model `cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_TARGET`

Model containing information regarding a DoIP entity

Fields

- *destination_port* (*int*)
- *entity_status_response* (*cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_ENTITY_STATUS | None*)

- *routing_activation_response* (*cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_ROUTING_ACTIVATION* | *None*)
- *routing_vehicle_id_response* (*cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_VEHICLE_IDENTIFICATION*)
- *source_ip* (*str*)
- *source_port* (*int*)
- *target_ip* (*str*)

field destination_port: *int* [Required]

target port

field entity_status_response: *Optional[DOIP_ENTITY_STATUS]* = *None*

DoIP entity status response

field routing_activation_response: *Optional[DOIP_ROUTING_ACTIVATION]* = *None*

DoIP routing activation response

field routing_vehicle_id_response: *DOIP_VEHICLE_IDENTIFICATION* [Required]

DoIP vehicle announcement/identification message

field source_ip: *str* [Required]

IP address of the client DoIP entity

field source_port: *int* [Required]

source port

field target_ip: *str* [Required]

IP address of the server DoIP entity

pydantic model

cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models.DOIP_VEHICLE_IDENTIFICATION

Model containing information regarding DoIP vehicle announcement/identification message

Fields

- *eid* (*str*)
- *further_action_required* (*int*)
- *gid* (*str*)
- *target_address* (*int*)
- *vin* (*str*)
- *vin_gid_sync_status* (*int* | *None*)

field eid: *str* = 'This is a unique identification of the DoIP entity'

field further_action_required: *int* [Required]

Further action required

field gid: *str* [Required]

This is a unique identification of a group of DoIP entity

field target_address: *int* [Required]

This is the logical address that is assigned to the responding DoIP entity

field vin: str [Required]

the vehicle's VIN

field vin_gid_sync_status: Optional[int] [Required]

VIN/GID sync. status

3.3 SOME/IP models

class cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.Layer4ProtocolType(value)

An enumeration.

pydantic model

cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SOMEIP_ENDPOINT_OPTION

Model containing information regarding SOME/IP endpoint

Fields

- *endpoint_addr (str)*
- *port (int)*
- *port_type (cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.Layer4ProtocolType)*

field endpoint_addr: str [Required]

The SOME/IP end point IP address

field port: int [Required]

The SOME/IP end point port

field port_type: Layer4ProtocolType [Required]

The SOME/IP end point protocol type either UDP or TCP

pydantic model

cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SOMEIP_EVTGROUP_INFO

Model containing information regarding SOME/IP event group

Fields

- *eventgroup_id (int)*
- *initial_data (bytes | None)*

field eventgroup_id: int [Required]

The Eventgroup ID

field initial_data: Optional[Annotated[bytes]] = None

Initial data associated with the eventgroup if got received

pydantic model

cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SOMEIP_METHOD_INFO

Model containing information regarding SOME/IP method

Fields

- *method_id (int)*
- *payload (bytes)*

field method_id: int [Required]

The Method ID

field payload: Annotated[bytes] [Required]

The payload associated with the method

Constraints

- **func** = <function <lambda> at 0x7fc4ab5951b0>
- **return_type** = PydanticUndefined
- **when_used** = always
- **json_schema** = {'type': 'string'}

pydantic model

`cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SOMEIP_SERVICE_INFO`

Model containing information regarding service

Fields

- *endpoints* (*list[cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SOMEIP_ENDPOINT_OPTION]*)
- *instance_id* (*int*)
- *major_ver* (*int*)
- *minor_ver* (*int*)
- *service_id* (*int*)
- *ttl* (*int*)

field endpoints: list[SOMEIP_ENDPOINT_OPTION] = []

List of endpoints offered by the service

field instance_id: int [Required]

The instance ID

field major_ver: int [Required]

Major version of the service

field minor_ver: int [Required]

Minor version of the service

field service_id: int [Required]

The Service ID

field ttl: int [Required]

Life time of the entry in seconds

class `cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models.SomeIpSdOptionFlags(value)`

An enumeration.

PROTOCOL SPECIFICS APIS

```
cyclarity_in_vehicle_sdk.protocol.uds.  
impl.uds_utils.UdsUtils  
cyclarity_in_vehicle_sdk.protocol.someip.  
impl.someip_utils.SomeipUtils  
cyclarity_in_vehicle_sdk.protocol.doip.  
impl.doip_utils.DoipUtils
```

4.1 cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils

pydantic model `cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils`

Fields

- `attempts (int)`
- `data_link_layer (cyclarity_in_vehicle_sdk.communication.isotp.impl.isotp_communicator.IsoTpCommunicator | cyclarity_in_vehicle_sdk.communication.doip.doip_communicator.DoipCommunicator)`

field `attempts: int = 1`

Number of attempts to perform the UDS operation if no response was received

Constraints

- `ge = 1`

field `data_link_layer: Union[IsoTpCommunicator, DoipCommunicator] [Required]`

authentication(`params`, `timeout=2`)

Initiate UDS Authentication service sequence

Parameters

- **params** (`Type[AuthenticationParamsBase]`) – Set of parameters defined for the desired authentication task
- **timeout** (`float`) – timeout for the UDS operation in seconds

Raises

- **NotImplementedError** – for operations that are not supported yet
- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type

- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received

Returns

The results code of the authentication action

Return type

AuthenticationReturnParameter

ecu_reset(*reset_type*, *timeout=2*)

The service “ECU reset” is used to restart the control unit (ECU)

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **reset_type** (*int*) – type of the reset (1: hard reset, 2: key Off-On Reset, 3: Soft Reset, .. more manufacture specific types may be supported)

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Returns

True if ECU request was accepted, False otherwise.

Return type

bool

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined model_post_init method.

Return type

None

raw_uds_service(*sid*, *timeout=2*, *sub_function=None*, *data=None*)

sends raw UDS service request and reads response

Parameters

- **sid** (*UdsSid*) – Service ID of the request
- **timeout** (*float*) – timeout for the UDS operation in seconds
- **sub_function** (*Optional[int]*, *optional*) – The service subfunction. Defaults to None.
- **data** (*Optional[bytes]*, *optional*) – The service data. Defaults to None.

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received

Returns

Raw UdsResponse

Return type

RawUdsResponse

read_did(*didlist*, *timeout=2*)

Read Data By Identifier

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **didlist** (*Union[int, list[int]]*) – List of data identifier to read.

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Returns

Dictionary mapping the DID (int) with the value returned

Return type

dict[int, str]

routine_control(*routine_id*, *control_type*, *timeout=2*, *data=None*)

Sends a request for RoutineControl

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **routine_id** (*int*) – The routine ID
- **control_type** (*int*) – Service subfunction
- **data** (*Optional[bytes]*, *optional*) – Optional additional data to provide to the server. Defaults to None.

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Return type

ResponseData

Returns

RoutingControlResponseData

security_access(*security_algorithm*, *timeout=2*)

Sends a request for SecurityAccess

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **security_algorithm** (*Type[SECURITY_ALGORITHM_BASE]*) – security algorithm to use for security access

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Returns

True if security access was allowed to the requested level. False otherwise

Return type

bool

session(*session*, *timeout=2*, *standard_version=UdsStandardVersion.ISO_14229_2020*)

Diagnostic Session Control

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **session** (*int*) – session to switch into
- **standard_version** (*UdsStandardVersion*, *optional*) – the version of the UDS standard we are interacting with. Defaults to ISO_14229_2020.

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Return type

ResponseData

Returns

SessionControlResultData

setup()

setup the library

Return type

bool

teardown()

Teardown the library

tester_present(*timeout=2*)

Sends a request for TesterPresent

Parameters

timeout (*float*) – timeout for the UDS operation in seconds

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Returns

True if tester preset was accepted successfully. False otherwise

Return type

bool

transit_to_session(*route_to_session, timeout=2,*
standard_version=UdsStandardVersion.ISO_14229_2020)

Transit to the UDS session according to route

Parameters

- **route_to_session** (*list[SESSION_ACCESS]*) – list of UDS SESSION_ACCESS objects to follow
- **timeout** (*float*) – timeout for the UDS operation in seconds
- **standard_version** (*UdsStandardVersion, optional*) – the version of the UDS standard we are interacting with. Defaults to ISO_14229_2020.

Returns

True if succeeded to transit to the session, False otherwise

Return type

bool

write_did(*did, value, timeout=2*)

Sends a request for WriteDataByIdentifier

Parameters

- **timeout** (*float*) – timeout for the UDS operation in seconds
- **did** (*int*) – The data identifier to write
- **value** (*str*) – the value to write

Raises

- **RuntimeError** – If failed to send the request
- **ValueError** – If parameters are out of range, missing or wrong type
- **NoResponse** – If no response was received
- **InvalidResponse** – with invalid reason, if invalid response has received
- **NegativeResponse** – with error code and code name, If negative response was received

Returns

True if WriteDataByIdentifier request sent successfully, False otherwise

Return type

bool

4.2 cyclarity_in_vehicle_sdk.protocol.someip.impl.someip_utils.SomeipUtils

pydantic model `cyclarity_in_vehicle_sdk.protocol.someip.impl.someip_utils.SomeipUtils`

find_service(*socket*, *service_id*, *recv_retry=1*, *recv_timeout=0.01*)

SOME/IP Find Service

Parameters

- **socket** (`UdpCommunicator` / `MulticastCommunicator`) – A SOME/IP SD socket (UDP) for sending FindService queries A SOME/IP SD socket for receiving offered services response (UDP) from broadcast (Multicast)
- **service_id** (*int*) – The Service ID to try query
- **recv_retry** (*int*) – Retries for receiving data from the SD socket. defaults to 1.
- **recv_timeout** (*float*) – Timeout in seconds for the read operation. defaults to 0.01

Return type

`list[SOMEIP_SERVICE_INFO]`

Returns

`list[SOMEIP_SERVICE_INFO]` list of found services

method_invoke(*socket*, *service_info*, *method_id*, *recv_timeout=0.01*)

Invoke SOME/IP Method

Parameters

- **socket** (`Union[UdpCommunicator, TcpCommunicator]`) – the end point communicator for method request/response
- **service_info** (`SOMEIP_SERVICE_INFO`) – information regarding the service in which the method is located
- **method_id** (*int*) – The Method ID
- **recv_timeout** (*float*) – Timeout in seconds for the read operation. defaults to 0.01

Return type

`SOMEIP_METHOD_INFO` | `None`

Returns

`SessionControlResultData`

subscribe_evtgrp(*sd_socket*, *ep_socket*, *service_info*, *evtgrpid*, *transport_protocol*, *recv_timeout=0.01*)

Subscribing to an eventgroup and fetch dome initial data

Parameters

- **sd_socket** (`UdpCommunicator`) – A SOME/IP SD socket (UDP) for sending FindService queries
- **ep_socket** (`Union[UdpCommunicator, TcpCommunicator]`) – the end point communicator for receiving the eventgroup data

- **service_info** (`SOMEIP_SERVICE_INFO`) – information regarding the service in which the event group is located
- **evtgrpid** (`int`) – the event group ID
- **transport_protocol** (`Layer4ProtocolType`) – the layer 4 protocol type UDP/TCP
- **recv_timeout** (`float`) – Timeout in seconds for the read operation. defaults to 0.01

Return type`SOMEIP_EVTGROUP_INFO` | `None`**Returns**`SOMEIP_EVTGROUP_INFO` if found. `None` otherwise

4.3 cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_utils.DoipUtils

pydantic model `cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_utils.DoipUtils`

Fields

- **raw_socket** (`cyclarity_in_vehicle_sdk.communication.ip.raw.raw_socket.Layer3RawSocket`)

field **raw_socket**: `Layer3RawSocket` [Required]

initiate_routing_activation_req(`source_address`, `target_address`, `client_logical_address`, `timeout=2`, `activation_type=ActivationType.Default`, `protocol_version=DoipProtocolVersion.DoIP_13400_2012`, `vm_specific=None`)

Initiate Routing activation request

Parameters

- **source_address** (`IPvAnyAddress`) – source IP address
- **target_address** (`IPvAnyAddress`) – target IP address
- **client_logical_address** (`int`) – client's logical address
- **timeout** (`float`, *optional*) – timeout in seconds for the operation
- **activation_type** (`ActivationType`, *optional*) – The activation type. Defaults to `ActivationType.Default`.
- **protocol_version** (`DoipProtocolVersion`, *optional*) – the Doip Protocol Version. Defaults to `DoipProtocolVersion.DoIP_13400_2012`.
- **vm_specific** (`int`, *optional*) – optional vm specific argument. Defaults to `None`.

Returns

`RoutingActivationResponse` if got a response, `None` otherwise

Return type

`Optional[RoutingActivationResponse]`

static initiate_routing_activation_req_bound(`communicator`, `client_logical_address`, `timeout=2`, `activation_type=ActivationType.Default`, `protocol_version=DoipProtocolVersion.DoIP_13400_2012`, `vm_specific=None`)

Initiate Routing activation request via the provided communicator

Parameters

- **communicator** (*Type[CommunicatorBase]*) – communicator to perform the request over
- **client_logical_address** (*int*) – client’s logical address
- **timeout** (*float, optional*) – timeout in seconds for the operation
- **activation_type** (*ActivationType, optional*) – The activation type. Defaults to *ActivationType.Default*.
- **protocol_version** (*DoipProtocolVersion, optional*) – the Doip Protocol Version. Defaults to *DoipProtocolVersion.DoIP_13400_2012*.
- **vm_specific** (*int, optional*) – optional vm specific argument. Defaults to *None*.

Returns

RoutingActivationResponse if got a response, *None* otherwise

Return type

Optional[RoutingActivationResponse]

initiate_vehicle_identity_req(*source_address, source_port, target_address, protocol_version=DoipProtocolVersion.DoIP_13400_2012, eid=None, vin=None*)

Initiate Vehicle identification request

Parameters

- **source_address** (*IPvAnyAddress*) – source IP address for the request
- **source_port** (*int*) – source port for the request
- **target_address** (*IPvAnyAddress*) – target IP address
- **protocol_version** (*DoipProtocolVersion, optional*) – the Doip Protocol Version. Defaults to *DoipProtocolVersion.DoIP_13400_2012*.
- **eid** (*bytes, optional*) – eid. Defaults to *None*.
- **vin** (*str, optional*) – vin. Defaults to *None*.

Returns

if got a response, *None* otherwise

Return type

VehicleIdentificationResponse

model_post_init(*_ModelMetaclass__context: Any*) → *None*

We need to both initialize private attributes and call the user-defined *model_post_init* method.

Return type

None

static read_uds_response(*communicator, timeout*)

Reads a UDS response

Parameters

- **communicator** (*Type[CommunicatorBase]*) – communicator to read the response over
- **timeout** (*float*) – timeout in seconds for the operation

Returns

UDS response in bytes if received a valid response, *False* otherwise

Return type

Optional[bytes]

req_entity_status(*source_address*, *source_port*, *target_address*,
protocol_version=DoipProtocolVersion.DoIP_13400_2012)

Initiate Entity status request

Parameters

- **source_address** (*IPvAnyAddress*) – source IP address
- **source_port** (*int*) – source port
- **target_address** (*IPvAnyAddress*) – target IP address
- **protocol_version** (*DoipProtocolVersion*, *optional*) – the Doip Protocol Version. Defaults to DoipProtocolVersion.DoIP_13400_2012.

Returns

if got a response, None otherwise

Return type

EntityStatusResponse

static send_uds_request(*logger*, *communicator*, *payload*, *client_logical_address*,
target_logical_address, *timeout*)

Sends a UDS request

Parameters

- **communicator** (*Type[CommunicatorBase]*) – communicator to perform the request over
- **payload** (*bytes*) – the UDS request payload
- **client_logical_address** (*int*) – client's logical address
- **target_logical_address** (*int*) – target's logical address
- **timeout** (*float*) – timeout in seconds for the operation

Returns

number of bytes actually sent

Return type

int

setup()

Opens the socket for communicating with the target

Returns

True if succeeded False otherwise

Return type

bool

teardown()

Closes communications with the target

Returns

True if succeeded False otherwise

Return type

bool

SHELL DEVICES

```
cyclarity_in_vehicle_sdk.utils.  
shell_device.impl.adb_device_shell.  
AdbDeviceShell  
cyclarity_in_vehicle_sdk.utils.  
shell_device.impl.serial_device_shell.  
SerialDeviceShell  
cyclarity_in_vehicle_sdk.utils.  
shell_device.impl.ssh_device_shell.  
SshDeviceShell
```

5.1 cyclarity_in_vehicle_sdk.utils.shell_device.impl.adb_device_shell.AdbDeviceShell

pydantic model

cyclarity_in_vehicle_sdk.utils.shell_device.impl.adb_device_shell.AdbDeviceShell

Fields

- *adb_authentication_method* (*Literal['None', 'Key']*)
- *adb_ip* (*str*)
- *adb_port* (*int | None*)
- *adb_private_key* (*str | None*)
- *adb_public_key* (*str | None*)

Validators

- *validate_ip* » *adb_ip*

field *adb_authentication_method*: *Literal['None', 'Key']* [Required]

Authentication method for interface

field *adb_ip*: *str* [Required]

shell interface ip OR 'usb'

Validated by

- *validate_ip*

field *adb_port*: *Optional[int]* = 5555

shell interface port

field adb_private_key: `Optional[str] = None`

private key (RSA-2048) for shell interface in base64

field adb_public_key: `Optional[str] = None`

public key (RSA-2048) for shell interface in base64

exec_command(*command*, *testcase_filter=None*, *return_stderr=False*, *verbose=False*)

This method executes a given command via adb interface and returns the output. If a *testcase_filter* is provided, it only returns lines that contain the filter string. If *return_stderr* is True, it also returns the stderr content (Not yet implemented!!!).

Parameters

- **command** (str) – String that represents the command to be executed.
- **testcase_filter** (Optional[str]) – Optional string used to filter the command's output.
- **return_stderr** (bool) – Optional boolean used to determine if stderr should be returned.
- **verbose** (bool) – Optional boolean used to log execution data

Return type

`Union[Tuple[str, ...], Tuple[Tuple[str, ...], str]]`

Returns

A tuple containing the command's output lines that match the *testcase_filter* and optionally stderr content. If no filter is provided, it returns all output lines.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined *model_post_init* method.

Return type

None

teardown()

This method is intended to close the adb session. If an error occurs during the operation, it is logged and re-raised.

validator validate_ip » `adb_ip`

5.2 cyclarity_in_vehicle_sdk.utils.shell_device.impl.serial_device_shell.SerialDeviceShell

pydantic model

`cyclarity_in_vehicle_sdk.utils.shell_device.impl.serial_device_shell.SerialDeviceShell`

Fields

- *serial_authentication_method* (`Literal[None, 'Password']`)
- *serial_boudrate* (`int | None`)
- *serial_bytesize* (`Literal[5, 6, 7, 8] | None`)
- *serial_device_name* (`str`)
- *serial_dsrdr* (`bool | None`)
- *serial_parity* (`Literal['N', 'E', 'O', 'M', 'S'] | None`)
- *serial_password* (`str | None`)
- *serial_rtscts* (`bool | None`)

- `serial_stopbits` (`Literal[1, 1.5, 2] | None`)
- `serial_timeout` (`float | None`)
- `serial_username` (`str | None`)
- `serial_write_inter_byte_timeout` (`float | None`)
- `serial_write_timeout` (`float | None`)
- `serial_xonxoff` (`bool | None`)

field serial_authentication_method: `Literal['None', 'Password'] [Required]`

Authentication method for interface

field serial_boudrate: `Optional[int] = 115200`

serial interface baud rate such as 9600 or 115200 etc

field serial_bytesize: `Optional[Literal[5, 6, 7, 8]] = 8`

serial interface Number of data bits. Possible values: 5, 6, 7, 8

field serial_device_name: `str [Required]`

serial device name e.g. `/dev/ttyUSB0`

field serial_dsrdtr: `Optional[bool] = False`

serial interface enable hardware (DSR/DTR) flow control.

field serial_parity: `Optional[Literal['N', 'E', 'O', 'M', 'S']] = 'N'`

serial interface enable parity checking. Possible values: 'N', 'E', 'O', 'M', 'S'

field serial_password: `Optional[str] = None`

Password for shell interface

field serial_rtscts: `Optional[bool] = False`

serial interface enable hardware (RTS/CTS) flow control

field serial_stopbits: `Optional[Literal[1, 1.5, 2]] = 1`

serial interface number of stop bits. Possible values: 1, 1.5, 2

field serial_timeout: `Optional[float] = 1`

serial interface read timeout value.

field serial_username: `Optional[str] = None`

Username for shell interface

field serial_write_inter_byte_timeout: `Optional[float] = None`

serial interface inter-character timeout, None to disable (default).

field serial_write_timeout: `Optional[float] = None`

serial interface write timeout value.

field serial_xonxoff: `Optional[bool] = False`

serial interface enable software flow control.

exec_command(`command`, `testcase_filter=None`, `return_stderr=False`, `verbose=False`)

This method executes a given command via serial interface and returns the output. If a `testcase_filter` is provided, it only returns lines that contain the filter string. If `return_stderr` is `True`, it also returns the stderr content (Not yet implemented!!!).

Parameters

- **command** (str) – String that represents the command to be executed.
- **testcase_filter** (Optional[str]) – Optional string used to filter the command’s output.
- **return_stderr** (bool) – Optional boolean used to determine if stderr should be returned.
- **verbose** (bool) – Optional boolean used to log execution data

Return type

Union[Tuple[str, ...], Tuple[Tuple[str, ...], str]]

Returns

A tuple containing the command’s output lines that match the `testcase_filter` and optionally `stderr` content. If no filter is provided, it returns all output lines.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Return type

None

teardown()

This method is intended to logout the serial session. If an error occurs during the operation, it is logged and re-raised.

5.3 cyclarity_in_vehicle_sdk.utils.shell_device.impl.ssh_device_shell.SshDeviceShell

pydantic model

`cyclarity_in_vehicle_sdk.utils.shell_device.impl.ssh_device_shell.SshDeviceShell`

Fields

- `ssh_authentication_method` (`Literal['None', 'Password', 'Key']`)
- `ssh_ip` (`pydantic.networks.IPvAnyAddress`)
- `ssh_password` (`str | None`)
- `ssh_port` (`int | None`)
- `ssh_private_key` (`str | None`)
- `ssh_username` (`str | None`)

field `ssh_authentication_method`: `Literal['None', 'Password', 'Key']` [Required]

Authentication method for interface

field `ssh_ip`: `IPvAnyAddress` [Required]

shell interface ip

field `ssh_password`: `Optional[str]` = None

Password for shell interface

field `ssh_port`: `Optional[int]` = 22

shell interface port

field `ssh_private_key`: `Optional[str]` = None

private key for shell interface in base64

field ssh_username: `Optional[str] = None`

Username for shell interface

exec_command(*command*, *testcase_filter=None*, *return_stderr=False*, *verbose=False*)

This method executes a given command via ssh and returns the output. If a *testcase_filter* is provided, it only returns lines that contain the filter string. If *return_stderr* is True, it also returns the stderr content.

Parameters

- **command** (str) – String that represents the command to be executed.
- **testcase_filter** (Optional[str]) – Optional string used to filter the command's output.
- **return_stderr** (bool) – Optional boolean used to determine if stderr should be returned.
- **verbose** (bool) – Optional boolean used to log execution data

Return type

Union[Tuple[str, ...], Tuple[Tuple[str, ...], str]]

Returns

A tuple containing the command's output lines that match the *testcase_filter* and optionally stderr content. If no filter is provided, it returns all output lines.

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined *model_post_init* method.

Return type

None

open_file(*filepath*, *mode='r'*, *bufsize=-1*)

Return type

SFTPFile

pull_file(*remote_filepath*, *local_filepath*)

push_file(*localpath*, *remotepath*)

teardown()

This method is intended to close the ssh connection. If an error occurs during the operation, it is logged and re-raised.

PLUGINS

```

cyclarity_in_vehicle_sdk.plugin.
crash_detection.session_change_detector.
SessionChangeCrashDetector
cyclarity_in_vehicle_sdk.
plugin.crash_detection.
unresponded_tp_crash_detector.
UnrespondedTesterPresentCrashDetector
cyclarity_in_vehicle_sdk.plugin.
recover_ecu.uds_ecu_recover.
UdsEcuRecoverPlugin
cyclarity_in_vehicle_sdk.plugin.reset.
relay.relay_reset_plugin.RelayResetPlugin
cyclarity_in_vehicle_sdk.plugin.
reset.uds_ecu_reset.uds_ecu_reset.
UdsBasedEcuResetPlugin

```

6.1 cyclarity_in_vehicle_sdk.plugin.crash_detection.session_change_detector.

pydantic model `cyclarity_in_vehicle_sdk.plugin.crash_detection.session_change_detector.SessionChangeCrashDetector`

Fields

- `current_session` (`int`)
- `operation_timeout` (`float`)
- `uds_utils` (`cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils`)

field `current_session`: `int` [Required]

Session ID of current session

Constraints

- `gt = 1`
- `le = 127`

field `operation_timeout`: `float` = 2

Timeout for the UDS operation in seconds

Constraints

- `gt = 0`

field `uds_utils`: *UdsUtils* [Required]

check_crash()

Return type
`bool`

setup()
Setup the plugin

Return type
`None`

teardown()
Teardown the plugin

Return type
`None`

6.2 `cyclarity_in_vehicle_sdk.plugin.crash_detection.unresponded_tp_crash_de`

pydantic model `cyclarity_in_vehicle_sdk.plugin.crash_detection.unresponded_tp_crash_detector.UnrespondedTesterPresentCrashDetector`

Fields

- *operation_timeout* (*float*)
- *uds_utils* (*cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils*)

field `operation_timeout`: `float = 2`
Timeout for the UDS operation in seconds

Constraints

- `gt = 0`

field `uds_utils`: *UdsUtils* [Required]

check_crash()

Return type
`bool`

setup()
Setup the plugin

Return type
`None`

teardown()
Teardown the plugin

Return type
`None`

6.3 cyclarity_in_vehicle_sdk.plugin.recover_ecu.uds_ecu_recover.UdsEcuRecoverPlugin

pydantic model

cyclarity_in_vehicle_sdk.plugin.recover_ecu.uds_ecu_recover.UdsEcuRecoverPlugin

Fields

- *operation_timeout* (*float*)
- *session_info* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.SESSION_INFO*)
- *uds_standard_version* (*cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models.UdsStandardVersion*)
- *uds_utils* (*cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils*)

field operation_timeout: `float = 2`

Timeout for the UDS operation in seconds

Constraints

- `gt = 0`

field session_info: `SESSION_INFO` [Required]

The information of the session to recover to

field uds_standard_version: `UdsStandardVersion = 'ISO_14229_2020'`

The standard version of the UDS in the target, defaults to latest (2020)

field uds_utils: `UdsUtils` [Required]

recover()

Recover the ECU to a predefined state :returns: True if recovery operation succeeded, False otherwise.
:rtype: bool

setup()

Setup the plugin

Return type

None

teardown()

Teardown the plugin

Return type

None

6.4 cyclarity_in_vehicle_sdk.plugin.reset_relay.relay_reset_plugin.RelayResetPlugin

pydantic model

cyclarity_in_vehicle_sdk.plugin.reset_relay.relay_reset_plugin.RelayResetPlugin

Fields

- *boot_sleep* (*float*)
- *gpio_chip* (*cyclarity_in_vehicle_sdk.plugin.reset_relay.relay_reset_plugin.GpioChip* | *str*)

- *reset_pin* (*int*)
- *shutdown_sleep* (*float*)

field boot_sleep: float = 1

Sleep after boot request, default to 1 second

Constraints

- *gt* = 0

field gpio_chip: Union[GpioChip, str] [Required]

The gpio chip connected to the relay e.g. /dev/gpiochip4

field reset_pin: int [Required]

Reset relay gpio pin

Constraints

- *ge* = 0

field shutdown_sleep: float = 1

Sleep after shutdown request, default to 1 second

Constraints

- *gt* = 0

model_post_init(*_ModelMetaclass__context: Any*) → None

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Return type

None

reset()

Resets the target device :returns: True if reset operation succeeded, False otherwise. :rtype: bool

setup()

Setup the plugin

Return type

None

teardown()

Teardown the plugin

Return type

None

6.5 cyclarity_in_vehicle_sdk.plugin.reset.uds_ecu_reset.uds_ecu_reset.UdsBa

pydantic model

`cyclarity_in_vehicle_sdk.plugin.reset.uds_ecu_reset.uds_ecu_reset.UdsBasedEcuResetPlugin`

Fields

- *operation_timeout* (*float*)
- *reset_type* (*int*)
- *uds_utils* (*cyclarity_in_vehicle_sdk.protocol.uds.impl.uds_utils.UdsUtils*)

field operation_timeout: float = 2

Timeout for the UDS operation in seconds

Constraints

- **gt** = 0

field reset_type: int = 1

Reset type (1: hard reset, 2: key Off-On Reset, 3: Soft Reset, ..). Allowed values are from 0 to 0x7F

Constraints

- **ge** = 0
- **le** = 127

field uds_utils: *UdsUtils* [Required]

reset()

Resets the target device :returns: True if reset operation succeeded, False otherwise. :rtype: bool

setup()

Setup the plugin

Return type

None

teardown()

Teardown the plugin

Return type

None

CONFIGURATION MANAGEMENT

7.1 ConfigurationManager

pydantic model

`cyclarity_in_vehicle_sdk.configuration_manager.configuration_manager.ConfigurationManager`

Fields

- `actions` (`list[cyclarity_in_vehicle_sdk.configuration_manager.actions.IpAddAction | cyclarity_in_vehicle_sdk.configuration_manager.actions.IpRemoveAction | cyclarity_in_vehicle_sdk.configuration_manager.actions.WifiConnectAction | cyclarity_in_vehicle_sdk.configuration_manager.actions.CanConfigurationAction | cyclarity_in_vehicle_sdk.configuration_manager.actions.EthInterfaceConfigurationAction | cyclarity_in_vehicle_sdk.configuration_manager.actions.CreateVlanAction] | None`)

field actions: `Optional[list[Union[IpAddAction, IpRemoveAction, WifiConnectAction, CanConfigurationAction, EthInterfaceConfigurationAction, CreateVlanAction]]] = None`

configure_actions(`actions`)

Configures the received actions

Parameters

actions (`Union[ConfigurationAction, list[ConfigurationAction]]`) – list of configuration actions to configure

get_device_configuration()

Get the current device configuration

Returns

the device's current configurations

Return type

DeviceConfiguration

setup()

Configures the received actions from the initialization

teardown()

Cleanup internal objects

7.2 Configuration Management - Models

<i>EthIfFlags</i> (value)	Enum for Ethernet interface flags
<i>InterfaceState</i> (value)	Enum for the state of the Ethernet interface
<i>IpRoute</i>	
<i>CanFdOptions</i>	
<i>CanInterfaceConfigurationInfo</i>	Model of the parameters for the CAN interface configurations
<i>IpConfigurationParams</i>	Model of the parameters for the IP configuration
<i>EthInterfaceParams</i>	Model of the parameters for the Ethernet interface configurations
<i>EthernetInterfaceConfigurationInfo</i>	Model of the parameters for the Ethernet interface information
<i>WifiAccessPointConfigurationInfo</i>	Model of the parameters for the Wifi interface information
<i>DeviceConfiguration</i>	Model of the parameters for the device configuration information

7.2.1 cyclarity_in_vehicle_sdk.configuration_manager.models.EthIfFlags

class cyclarity_in_vehicle_sdk.configuration_manager.models.**EthIfFlags**(value)

Enum for Ethernet interface flags

__init__()

Methods

conjugate	Returns self, the complex conjugate of any int.
bit_length()	Number of bits necessary to represent self in binary.
bit_count()	Number of ones in the binary representation of the absolute value of self.
to_bytes (length, byteorder, *[, signed])	Return an array of bytes representing an integer.
from_bytes (byteorder, *[, signed])	Return the integer represented by the given array of bytes.
as_integer_ratio()	Return integer ratio.
get_flags_from_int (flags)	

Attributes

<code>real</code>	the real part of a complex number
<code>imag</code>	the imaginary part of a complex number
<code>numerator</code>	the numerator of a rational number in lowest terms
<code>denominator</code>	the denominator of a rational number in lowest terms
<code>IFF_UP</code>	
<code>IFF_BROADCAST</code>	
<code>IFF_DEBUG</code>	
<code>IFF_LOOPBACK</code>	
<code>IFF_POINTOPOINT</code>	
<code>IFF_NOTRAILERS</code>	
<code>IFF_RUNNING</code>	
<code>IFF_NOARP</code>	
<code>IFF_PROMISC</code>	
<code>IFF_ALLMULTI</code>	
<code>IFF_MASTER</code>	
<code>IFF_SLAVE</code>	
<code>IFF_MULTICAST</code>	
<code>IFF_PORTSEL</code>	
<code>IFF_AUTOMEDIA</code>	
<code>IFF_DYNAMIC</code>	
<code>IFF_LOWER_UP</code>	
<code>IFF_DORMANT</code>	
<code>IFF_ECHO</code>	

7.2.2 cyclarity_in_vehicle_sdk.configuration_manager.models.InterfaceState

class `cyclarity_in_vehicle_sdk.configuration_manager.models.InterfaceState`(*value*)
 Enum for the state of the Ethernet interface
`__init__()`

Methods

<code>encode([encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>split([sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rsplit([sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>capitalize()</code>	Return a capitalized version of the string.
<code>casefold()</code>	Return a version of the string suitable for caseless comparisons.
<code>title()</code>	Return a version of the string where each word is titlecased.
<code>center(width[, fillchar])</code>	Return a centered string of length <code>width</code> .
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring <code>sub</code> in string <code>S[start:end]</code> .
<code>expandtabs([tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>index(sub[, start[, end]])</code>	Return the lowest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length <code>width</code> .
<code>lower()</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length <code>width</code> .
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>splitlines([keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase()</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper()</code>	Return a copy of the string converted to uppercase.
<code>startswith(prefix[, start[, end]])</code>	Return <code>True</code> if <code>S</code> starts with the specified prefix, <code>False</code> otherwise.
<code>endswith(suffix[, start[, end]])</code>	Return <code>True</code> if <code>S</code> ends with the specified suffix, <code>False</code> otherwise.

continues on next page

Table 1 – continued from previous page

<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>isascii()</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>islower()</code>	Return True if the string is a lowercase string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isnumeric()</code>	Return True if the string is a numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans</code> <code>state_from_string(str_state)</code>	Return a translation table usable for <code>str.translate()</code> .

Attributes

UP
DOWN
UNKNOWN

7.2.3 cyclarity_in_vehicle_sdk.configuration_manager.models.IpRoute

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.models.IpRoute`

Fields

- *gateway* (*str* | *None*)

field gateway: `Optional[str] = None`

Optional parameter the route gateway, none for default gateway

7.2.4 cyclarity_in_vehicle_sdk.configuration_manager.models.CanFdOptions

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.models.CanFdOptions`

Fields

- *dbitrate* (*int*)

field dbitrate: `int = 2000000`

The data bitrate

7.2.5 cyclarity_in_vehicle_sdk.configuration_manager.models.CanInterfaceConfigurationInfo

pydantic model

`cyclarity_in_vehicle_sdk.configuration_manager.models.CanInterfaceConfigurationInfo`

Model of the parameters for the CAN interface configurations

Fields

- *bitrate* (*int*)
- *cc_len8_dlc* (*bool*)
- *channel* (*str*)
- *fd* (*cyclarity_in_vehicle_sdk.configuration_manager.models.CanFdOptions* | *None*)
- *sample_point* (*float*)
- *state* (*cyclarity_in_vehicle_sdk.configuration_manager.models.InterfaceState*)

field bitrate: `int = 500000`

Bitrate

field cc_len8_dlc: `bool [Required]`

cc-len8-dlc flag value

field channel: `str [Required]`

The CAN interface e.g. can0

field fd: `Optional[CanFdOptions] = None`

Set interface to support CAN-FD

field sample_point: `float = 0.875`

Sample-point

field state: `InterfaceState = 'UP'`

The state of the CAN interface - UP/DOWN

7.2.6 cyclarity_in_vehicle_sdk.configuration_manager.models.IpConfigurationParams

pydantic model

`cyclarity_in_vehicle_sdk.configuration_manager.models.IpConfigurationParams`

Model of the parameters for the IP configuration

Fields

- `interface` (*str*)
- `ip` (*pydantic.networks.IpvAnyAddress*)
- `route` (*cyclarity_in_vehicle_sdk.configuration_manager.models.IpRoute* | *None*)
- `suffix` (*int*)

Validators

- `validate_ip_subnet` » all fields

field interface: `str` [Required]

The network interface for the IP to be configured

Validated by

- `validate_ip_subnet`

field ip: `IpvAnyAddress` [Required]

The IP to configure, IPv4/IPv6

Validated by

- `validate_ip_subnet`

field route: `Optional[IpRoute] = None`

Optional parameter for setting a route for the IP

Validated by

- `validate_ip_subnet`

field suffix: `int` [Required]

The subnet notation for this IP address

Validated by

- `validate_ip_subnet`

validator validate_ip_subnet » *all fields*

property cidr_notation: `str`

7.2.7 cyclarity_in_vehicle_sdk.configuration_manager.models.EthInterfaceParams

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.models.EthInterfaceParams`

Model of the parameters for the Ethernet interface configurations

Fields

- `flags` (*list[cyclarity_in_vehicle_sdk.configuration_manager.models.EthIfFlags]*)
- `interface` (*str*)

- *mtu* (*int* | *None*)
- *state* (*cyclarity_in_vehicle_sdk.configuration_manager.models.InterfaceState* | *None*)

field flags: *list[EthIfFlags]* = ['IFF_BROADCAST', 'IFF_MULTICAST', 'IFF_UP', 'IFF_LOWER_UP', 'IFF_RUNNING']

Flags to apply on the interface

field interface: *str* [Required]

The Eth interface to be configured

field mtu: *Optional[int]* = *None*

MTU (maximum transmission unit)

field state: *Optional[InterfaceState]* = *None*

Interface State to configure

7.2.8 *cyclarity_in_vehicle_sdk.configuration_manager.models.EthernetInterfaceConfigurationInfo*

pydantic model

cyclarity_in_vehicle_sdk.configuration_manager.models.EthernetInterfaceConfigurationInfo

Model of the parameters for the Ethernet interface information

Fields

- *if_params* (*cyclarity_in_vehicle_sdk.configuration_manager.models.EthInterfaceParams*)
- *ip_params* (*list[cyclarity_in_vehicle_sdk.configuration_manager.models.IpConfigurationParams]*)

field if_params: *EthInterfaceParams* [Required]

field ip_params: *list[IpConfigurationParams]* [Required]

7.2.9 *cyclarity_in_vehicle_sdk.configuration_manager.models.WifiAccessPointConfigurationInfo*

pydantic model

cyclarity_in_vehicle_sdk.configuration_manager.models.WifiAccessPointConfigurationInfo

Model of the parameters for the Wifi interface information

Fields

- *connected* (*bool*)
- *security* (*str*)
- *ssid* (*str*)

field connected: *bool* [Required]

Is the device connected to this access point

field security: *str* [Required]

The security access of the access point

field ssid: *str* [Required]

The SSID of the access point

7.2.10 cyclarity_in_vehicle_sdk.configuration_manager.models.DeviceConfiguration

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.models.DeviceConfiguration`

Model of the parameters for the device configuration information

Fields

- `configurations_info` (`list[cyclarity_in_vehicle_sdk.configuration_manager.models.ConfigurationInfoBase]`)

field `configurations_info`: `list[ConfigurationInfoBase] = []`

7.3 Configuration Management - Actions

<code>IpAddAction</code>	Action for adding an IP address to an ethernet interface
<code>IpRemoveAction</code>	Action for removing an IP address to an ethernet interface
<code>WifiConnectAction</code>	Action for connecting to a wifi network
<code>CanConfigurationAction</code>	Action for configuring the CAN interface
<code>EthInterfaceConfigurationAction</code>	Action for configuring the Ethernet interface
<code>CreateVlanAction</code>	Action for creating a VLAN interface linked to an actual Eth interface

7.3.1 cyclarity_in_vehicle_sdk.configuration_manager.actions.IpAddAction

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.actions.IpAddAction`

Action for adding an IP address to an ethernet interface

Fields

- `action_type` (`Literal['add_ip']`)

Validators

field `action_type`: `Literal['add_ip'] = 'add_ip'`

Validated by

- `validate_ip_subnet`

7.3.2 cyclarity_in_vehicle_sdk.configuration_manager.actions.IpRemoveAction

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.actions.IpRemoveAction`

Action for removing an IP address to an ethernet interface

Fields

- `action_type` (`Literal['del_ip']`)

Validators

field `action_type`: `Literal['del_ip'] = 'del_ip'`

Validated by

- `validate_ip_subnet`

7.3.3 cyclarity_in_vehicle_sdk.configuration_manager.actions.WifiConnectAction

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.actions.WifiConnectAction`

Action for connecting to a wifi network

Fields

- `action_type` (`Literal['wifi_connect']`)
- `password` (`str`)
- `ssid` (`str`)

field `action_type`: `Literal['wifi_connect'] = 'wifi_connect'`

field `password`: `str` [Required]

The pass phrase to use for connecting

field `ssid`: `str` [Required]

The SSID of the access point to connect to

7.3.4 cyclarity_in_vehicle_sdk.configuration_manager.actions.CanConfigurationAction

pydantic model

`cyclarity_in_vehicle_sdk.configuration_manager.actions.CanConfigurationAction`

Action for configuring the CAN interface

Fields

- `action_type` (`Literal['con_conf']`)

field `action_type`: `Literal['con_conf'] = 'con_conf'`

7.3.5 cyclarity_in_vehicle_sdk.configuration_manager.actions.EthInterfaceConfigurationAction

pydantic model

`cyclarity_in_vehicle_sdk.configuration_manager.actions.EthInterfaceConfigurationAction`

Action for configuring the Ethernet interface

Fields

- `action_type` (`Literal['eth_conf']`)

field `action_type`: `Literal['eth_conf'] = 'eth_conf'`

7.3.6 cyclarity_in_vehicle_sdk.configuration_manager.actions.CreateVlanAction

pydantic model `cyclarity_in_vehicle_sdk.configuration_manager.actions.CreateVlanAction`

Action for creating a VLAN interface linked to an actual Eth interface

Fields

- `action_type` (`Literal['vlan_create']`)
- `if_link` (`str`)
- `if_name` (`str`)
- `vlan_id` (`int`)

field `action_type`: `Literal['vlan_create'] = 'vlan_create'`

field if_link: str [Required]

The physical interface to link to

field if_name: str [Required]

The new vlan interface name

field vlan_id: int [Required]

The vlan ID

PYTHON MODULE INDEX

C

`cyclarity_in_vehicle_sdk.protocol.doip.impl.doip_models,`

[25](#)

`cyclarity_in_vehicle_sdk.protocol.someip.models.someip_models,`

[27](#)

`cyclarity_in_vehicle_sdk.protocol.uds.models.uds_models,`

[19](#)