RobotFramework_DoIP

v. 0.1.0

Hua Van Thong

20.09.2023

Contents

Chapter 1

Introduction

1.1 Introduction TODO

RobotFramework_DoIP is a Robot Framework library specifically designed for interacting with Electronic Control Units (ECUs) using the Diagnostics over Internet Protocol (DoIP).

At its core, DoIP serves as a communication bridge between external diagnostic tools and a vehicle's ECUs. This library, RobotFrameworkDoIP, provides a set of keywords that enable users to perform diagnostic operations and engage with ECUs, facilitating automated testing processes and interaction with vehicles through the DoIP protocol.

The **RobotFramework_DoIP** sources can be found in repository **robotframework-doip**: DoIP

Chapter 2

Description

2.1 Description TODO

Chapter 3

TheEcuSimulator

3.1 The ECU Simulator

This chapter provides a detailed explanation of the utilization of the ECU simulator through DoIP base on doipclient library. It serves for development or testing scenarios where a physical device is not available.
The ECU simulator is designed to receive messages and respond accordingly to the following types of messages:

- Alive Check Request
- Diagnostic Power Mode Request
- Doip Entity Status Request
- Routing Activation Request
- Vehicle Identification Request

3.2 Initialize

This function sets up an instance of an ECU, initializes its attributes with default values, and includes placeholders for various properties that can be customized based on specific requirements.

```
    def __init__(self, ecu_type, ip_address, tcp_port, udp_port):
        # Initialize ECU attributes with default values
        self.ecu_type = ecu_type
        self.ip_address = ip_address
        self.tcp_port = tcp_port
        self.udp_port = udp_port
        self.tcp_socket = None
        self.udp_socket = None
        # Set default values for various ECU properties
        # These values might be placeholders and can be updated based on your actual ↩
    ↪ requirements
        self._ecu_logical_address = 3584
        self._client_logical_address = 3584
        self._logical_address = 55
        self._response_code = doip_message.RoutingActivationResponse.ResponseCode.Success
        self._diagnostic_power_mode = ↩
    ↪ doip_message.DiagnosticPowerModeResponse.DiagnosticPowerMode.Ready
        self._node_type = 1
        self._max_concurrent_sockets = 16
        self._currently_open_sockets = 1
        self._max_data_size = None
        self._vin = '19676527011956855057'
        self._eid = b'11111'
        self._gid = b'222222'
        self._further_action_required = ↩
    ↪ doip_message.VehicleIdentificationResponse.FurtherActionCodes.NoFurtherActionRequired
        self._vin_sync_status = ↩
    ↪ doip_message.VehicleIdentificationResponse.SynchronizationStatusCodes.Synchronized
```

3.3 Start

This method is responsible for initializing and setting up TCP and UDP sockets, binding them to specific IP addresses and ports, and then starting separate threads to handle the communication on these sockets concurrently.

```
def start(self):
    # Create TCP socket
    self.tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.tcp_socket.bind((self.ip_address, self.tcp_port))
    self.tcp_socket.listen(5)

    # Create UDP socket
    self.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.udp_socket.bind((self.ip_address, self.udp_port))

    # Start listening on separate threads
    tcp_thread = threading.Thread(target=self.listen_tcp)
    udp_thread = threading.Thread(target=self.listen_udp)

    tcp_thread.start()
    udp_thread.start()
```

Explanation:

1. TCP Socket Setup
   - A TCP socket is created using the socket module with the `socket.AF_INET` family (IPv4) and `socket.SOCK_STREAM` type (TCP).
   - The TCP socket is bound to the specified IP address `self.ip_address` and TCP port `self.tcp_port` .
   - The TCP socket is set to listen for incoming connections with a backlog of 5 connections.
2. UDP Socket Setup
   - A UDP socket is created using the same socket module with the `socket.AF_INET` family (IPv4) and `socket.SOCK_DGRAM type` (UDP).
   - The UDP socket is bound to the specified IP address `self.ip_address` and UDP port `self.udp_port` .
3. Thread Creation
   - Two separate threads `tcp_thread and udp_thread` are created using the threading module.
   - The target parameter of each thread is set to point to specific methods `self.listen_tcp` and `self.listen_udp` , suggesting that these methods likely contain the logic for handling TCP and UDP communication.
4. Thread Start
   - Both threads are started concurrently using the start method, allowing the ECU to handle TCP and UDP communication simultaneously.

3.4 Example

We have provided an example demonstrating the usage of the ECU simulator in the file located at `test_ecu_simulator.py`

```
if __name__ == "__main__":
    # Create and start instances of different ECUs using the factory pattern and abstract class
    factory = ECUFactory()

    positive_ecu = factory.create_ecu(ECUType.POSITIVE_ECU, POSITIVE_ECU_IP, POSITIVE_TCP_PORT, POSITIVE_UDP_PORT)
    negative_ecu = factory.create_ecu(ECUType.NEGATIVE_ECU, NEGATIVE_ECU_IP, NEGATIVE_TCP_PORT, NEGATIVE_UDP_PORT)
    # Start positive and negative ECUs
    positive_ecu.start()
    negative_ecu.start()
```

In the given example, an instance of the ECU is created in `ecu_simulator.py` by specifying the ECU's IP address, TCP port, and UDP port. Subsequently, the start method is invoked to initiate its operation.

Output:

```
TCP Server 172.17.0.5 listening on port 13400
UDP Server 172.17.0.5 listening on port 13400
TCP Server 172.17.0.5 listening on port 12346
UDP Server 172.17.0.5 listening on port 12347
```

Now you can execute the test by running the file located at `test_ecu_simulator.py`

```
def test_positive_ecu_simulator():
    try:
        ip = '172.17.0.5'
        ecu_logical_address = 57344

        # Create a DoIPClient instance for positive ECU simulator
        doip = DoIPClient(ip, ecu_logical_address, activation_type=None)

        # Test various interactions
        print(doip.request_diagnostic_power_mode())
        print(doip.request_entity_status())
        print(doip.request_alive_check())
        print(doip.request_activation(1))
        print(doip.get_entity())
        print(doip.request_vehicle_identification(vin="1" * 17))
        print(doip.request_vehicle_identification(eid=b"1" * 6))

    except Exception as e:
        print(f"Error during positive ECU simulation: {e}")
```

Output:

```
# Diagnostic power mode response
DiagnosticPowerModeResponse (0x4004): { diagnostic_power_mode : DiagnosticPowerMode.Ready }

# Entity status response
EntityStatusResponse (0x4002): { node_type : 1, max_concurrent_sockets : 16, currently_open_sockets : 1, max_data_size : None }

# Alive check response
AliveCheckResponse (0x8): { source_address : 3584 }

# Routing activation response
RoutingActivationResponse (0x6): { client_logical_address : 3584, logical_address : 55, response_code : ResponseCode.Success, reserved : 0, vm_specific : None }

# Get entity response
(('172.17.0.5', 13400), VehicleIdentificationResponse(b'19676527011956855', 3584, b'11111\x00', b'222222', 0, 0))

# Vehicle identification response
VehicleIdentificationResponse (0x4): { vin: "19676527011956855", logical_address : 3584, eid : b'11111\x00', gid : b'222222', further_action_required : FurtherActionCodes.NoFurtherActionRequired, vin_sync_status : SynchronizationStatusCodes.Synchronized }
VehicleIdentificationResponse (0x4): { vin: "19676527011956855", logical_address : 3584, eid : b'11111\x00', gid : b'222222', further_action_required : FurtherActionCodes.NoFurtherActionRequired, vin_sync_status : SynchronizationStatusCodes.Synchronized }
```

Chapter 4

DoipKeywords.py

4.1 Class: DoipKeywords

Imported by:

```
from RobotFramework_DoIP.DoipKeywords import DoipKeywords
```

4.1.1 Method: connect_to_ecu

Description:

Establishing a connection to an (ECU) within the context of automotive communication.

Parameters:

- **param `ecu_ip_address` (required): The IP address of the ECU to establish a connection. This shoul**
  address like "192.168.1.1" or an IPv6 address like "2001:db8::".
- type `ecu_ip_address`: str
- param `ecu_logical_address` (required): The logical address of the ECU.
- type `ecu_logical_address`: int
- param `tcp_port` (optional): The TCP port used for unsecured data communication (default is **TCP_DATA_UNSECURED**).
- type `tcp_port`: int
- param `udp_port` (optional): The UDP port used for ECU discovery (default is **UDP_DISCOVERY**).
- type `udp_port`: int
- param `activation_type` (optional): The type of activation, which can be the default value (ActivationTypeDefault) or a specific value based on application-specific settings.
- type `activation_type`: RoutingActivationRequest.ActivationType,
- param `protocol_version` (optional): The version of the protocol used for the connection (default is 0x02).
- type `protocol_version`: int
- **param `client_logical_address` (optional): The logical address that this DoIP client will use to ide**
  this should be 0x0E00 to 0x0FFF. Can typically be left as default.
- type `client_logical_address`: int
- **param `client_ip_address` (optional): If specified, attempts to bind to this IP as the source for both**
  Useful if you have multiple network adapters. Can be an IPv4 or IPv6 address just like ecu_ip_address, though the type should match.
- type `client_ip_address`: str
- **param `use_secure` (optional): Enables TLS. If set to True, a default SSL context is used. For more**
  SSL context can be passed directly. Untested. Should be combined with changing tcp_port to 3496.

- type `use_secure`: Union[bool,ssl.SSLContext]
- param `auto_reconnect_tcp` (optional): Attempt to automatically reconnect TCP sockets that were closed by peer
- type `auto_reconnect_tcp`: bool

Return:

None

Usage:

Explicitly specifies all establishing a connection

- Connect To ECU | 172.17.0.111 | ${1863} |
- Connect To ECU | 172.17.0.111 | ${1863} | client_ip_address=172.17.0.5 | client_logical_address=${1895} |
- Connect To ECU | 172.17.0.111 | ${1863} | client_ip_address=172.17.0.5 | client_logical_address=${1895} | activation_type=${0} |

4.1.2 Method: send_diagnostic_message

Description:

Send a raw diagnostic payload (ie: UDS) to the ECU.

Parameters:

- param `diagnostic_payload`: UDS payload to transmit to the ECU
- type `diagnostic_payload`: bytearray
- param `timeout`: send diagnostic time out (default: A_PROCESSING_TIME)
- type `timeout`: int (s)

Return:

None

Exception:

raises IOError: DoIP negative acknowledgement received

Usage:

Explicitly specifies all diagnostic message properties

- Send Diagnostic Message | 1040 |
- Send Diagnostic Message | 1040 | timeout=10 |

4.1.3 Method: receive_diagnostic_message

Description:

Receive a raw diagnostic payload (ie: UDS) from the ECU.

Parameters:

- param `timeout`: time waiting diagnostic message (default: None)
- type `timeout`: int (s)

Return:

None

Exception:

raises IOError: DoIP negative acknowledgement received

Usage:

Explicitly specifies all diagnostic message properties

- Receive Diagnostic Message |
- Receive Diagnostic Message | timeout=10 |

4.1.4 Method: reconnect_to_ecu

Description:

Attempts to re-establish the connection. Useful after an ECU reset

Parameters:

- param `close_delay`: Time to wait between closing and re-opening socket (default: **A_PROCESSING_TIME**)
- type `close_delay`: int (s)

Return: None

Exception:

raises ConnectionRefusedError: DoIP negative acknowledgement received

Usage:

Explicitly specifies all diagnostic message properties

- Reconnect To Ecu |
- Receive Diagnostic Message | timeout=10 |

4.1.5 Method: disconnect

Description:

Close the DoIP client

Parameters:

None

Return:

None

Exception:

None

Usage:

Explicitly specifies all diagnostic message properties

- Disconnect

4.1.6 Method: await_vehicle_announcement

Description:

When an ECU first turns on, it's supposed to broadcast a Vehicle Announcement Message over UDP 3 times to assist DoIP clients in determining ECU IP's and Logical Addresses. Will use an IPv4 socket by default, though this can be overridden with the ipv6 parameter.

Parameters:

- param `udp_port`: The UDP port to listen on. Per the spec this should be 13400, but some VM's use a custom
- one.
- type `udp_port`: int, optional
- param `timeout`: Maximum amount of time to wait for message
- type `timeout`: float, optional
- param `ipv6`: Bool forcing IPV6 socket instead of IPV4 socket
- type `ipv6`: bool, optional
- **param `source_interface`: Interface name (like "eth0") to bind to for use with IPv6. Defaults to No** will use the default interface (which may not be the one connected to the ECU). Does nothing for IPv4, which will bind to all interfaces uses INADDR_ANY.
- type `source_interface`: str, optional

Return:

- return: IP Address of ECU and VehicleAnnouncementMessage object
- rtype: tuple

Exception:

raises TimeoutError: If vehicle announcement not received in time

Usage:

Explicitly specifies all diagnostic message properties

- Await Vehicle Annoucement
- Await Vehicle Annoucement | timeout=10

4.1.7 Method: get_entity

Description:

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing. Equivalent to the request_vehicle_identification() method but can be called without instantiation

Parameters:

- param `udp_port`: The UDP port to listen on. Per the spec this should be 13400, but some VM's use a custom
- one.
- type `udp_port`: int, optional
- param `timeout`: Maximum amount of time to wait for message
- type `timeout`: float, optional
- param `ipv6`: Bool forcing IPV6 socket instead of IPV4 socket

- type `ipv6`: bool, optional
- **param `source_interface`: Interface name (like "eth0") to bind to for use with IPv6. Defaults to No** will use the default interface (which may not be the one connected to the ECU). Does nothing for IPv4, which will bind to all interfaces uses INADDR_ANY.
- type `source_interface`: str, optional

Return:

- return: IP Address of ECU and VehicleAnnouncementMessage object
- rtype: tuple

Exception:

raises TimeoutError: If vehicle announcement not received in time

Usage:

- Get Entity |
- Get Entity | ecu_ip_address=172.17.0.111 |
- Get Entity | ecu_ip_address=172.17.0.111 | protocol_version=0x02

4.1.8 Method: request_entity_status

Description:

Request that the ECU send a DoIP Entity Status Response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Entity Status

4.1.9 Method: request_vehicle_identification

Description:

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing

Parameters:

param eid EID of the Vehicle

type eid bytes, optional

param vin VIN of the Vehicle

type vin str, optional

Return:

None

Exception:

None

Usage:

- Request Vehicle Identification
- Request Vehicle Identification | eid=0x123456789abc
- Request Vehicle Identification | vin=0x123456789abc

4.1.10 Method: request_alive_check

Description:

Request that the ECU send an alive check response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Vehicle Identification
- Request Vehicle Identification | eid=0x123456789abc
- Request Vehicle Identification | vin=0x123456789abc

4.1.11 Method: request_activation

Description:

Requests a given activation type from the ECU for this connection using payload type 0x0005

Parameters:

- **param `activation_type` (required): The type of activation to request - see Table 47 ("Routing** activation request activation types") of ISO-13400, but should generally be 0 (default) or 1 (regulatory diagnostics)
- type `activation_type`: RoutingActivationRequest.ActivationType
- param `vm_specific` (optional): 4 byte long int
- type `vm_specific`: int, optional
- **param `disable_retry`: Disables retry regardless of auto_reconnect_tcp flag. This is used by activatio** requests during connect/reconnect.
- type `disable_retry`: bool, optional

Return:

None

Exception:

None

Usage:

- Request Routing Activation | ${0x02}
- Request Routing Activation | vm_specific=
- Request Routing Activation | vin=0x123456789abc

4.1.12 Method: request_diagnostic_power_mode

Description:

Request that the ECU send a Diagnostic Power Mode response

Parameters:

None

Return:

None

Exception:

None

Usage:

- Request Diagnostic Power Mode

Chapter 5

RobotFramework_DoIP.py

5.1 Function: get_version

5.2 Function: get_version_date

Chapter 6

__init__.py

6.1 Class: RobotFramework_DoIP

Imported by:

```
from RobotFramework_DoIP.__init__ import RobotFramework_DoIP
```

RobotFrameworkDoIP is a Robot Framework library aimed to provide DoIP protocol for diagnostic message.

Chapter 7

client.py

7.1 Class: Parser

Imported by:

```
from RobotFramework_DoIP.doipclient.client import Parser
```

Implements state machine for DoIP transport layer.

See Table 16 "Generic DoIP header structure" of ISO 13400-2:2019 (E). While TCP transport is reliable, the UDP broadcasts are not, so the state machine is a little more defensive than one might otherwise expect. When using TCP, reads from the socket aren't guaranteed to be exactly one DoIP message, so the running buffer needs to be maintained across reads robotframework-doip-doipclient-client-parser-reset ----------

7.1.1 Method: push_bytes

7.1.2 Method: read_message

7.2 Class: DoIPClient

Imported by:

```
from RobotFramework_DoIP.doipclient.client import DoIPClient
```

A Diagnostic over IP (DoIP) Client implementing the majority of ISO-13400-2:2019 (E).

This is a basic DoIP client which was designed primarily for use with the python-udsoncan package for UDS communication with ECU's over automotive ethernet. Certain parts of the specification would require threaded operation to maintain the time-based state described by the ISO document. However, in practice these are rarely important, particularly for use with UDS - especially with scripts that tend to go through instructions as fast as possible.

param ecu_ip_address This is the IP address of the target ECU. This should be a string representing an IPv4 address like "192.168.1.1" or an IPv6 address like "2001:db8::". Like the logical_address, if you don't know the value for your ECU, utilize the get_entity() or await_vehicle_announcement() method.

type ecu_ip_address str

param ecu_logical_address The logical address of the target ECU. This should be an integer. According to the specification, the correct range is 0x0001 to 0x0DFF ("VM specific"). If you don't know the logical address, either use the get_entity() method OR the await_vehicle_announcement() method and power cycle the ECU - it should identify itself on bootup.

type ecu_logical_address int

param tcp_port The destination TCP port for DoIP data communication. By default this is 13400 for unsecure and 3496 when using TLS.

type tcp_port int, optional

param activation_type The activation type to use on initial connection. Most ECU's require an activation request before they'll respond, and typically the default activation type will do. The type can be changed later using request_activation() method. Use None to disable activation at startup.

type activation_type RoutingActivationRequest.ActivationType, optional

param protocol_version The DoIP protocol version to use for communication. Represents the version of the ISO 13400 specification to follow. 0x02 (2012) is probably correct for most ECU's at the time of writing, though technically this implementation is against 0x03 (2019).

type protocol_version int

param client_logical_address The logical address that this DoIP client will use to identify itself. Per the spec, this should be 0x0E00 to 0x0FFF. Can typically be left as default.

type client_logical_address int

param client_ip_address If specified, attempts to bind to this IP as the source for both UDP and TCP communication. Useful if you have multiple network adapters. Can be an IPv4 or IPv6 address just like ecu_ip_address, though the type should match.

type client_ip_address str, optional

param use_secure Enables TLS. If set to True, a default SSL context is used. For more control, a preconfigured SSL context can be passed directly. Untested. Should be combined with changing tcp_port to 3496.

type use_secure Union[bool,ssl.SSLContext]

param log_level Logging level

type log_level int

param auto_reconnect_tcp Attempt to automatically reconnect TCP sockets that were closed by peer

type auto_reconnect_tcp bool

raises ConnectionRefusedError If the activation request fails

raises ValueError If the IPAddress is neither an IPv4 nor an IPv6 address

7.2.1 Method: await_vehicle_announcement

Receive Vehicle Announcement Message

When an ECU first turns on, it's supposed to broadcast a Vehicle Announcement Message over UDP 3 times to assist DoIP clients in determining ECU IP's and Logical Addresses. Will use an IPv4 socket by default, though this can be overridden with the ipv6 parameter.

param udp_port The UDP port to listen on. Per the spec this should be 13400, but some VM's use a custom one.

type udp_port int, optional

param timeout Maximum amount of time to wait for message

type timeout float, optional

param ipv6 Bool forcing IPV6 socket instead of IPV4 socket

type ipv6 bool, optional

param source_interface Interface name (like "eth0") to bind to for use with IPv6. Defaults to None which will use the default interface (which may not be the one connected to the ECU). Does nothing for IPv4, which will bind to all interfaces uses INADDR_ANY.

type source_interface str, optional

return IP Address of ECU and VehicleAnnouncementMessage object

rtype tuple

raises TimeoutError If vehicle announcement not received in time

7.2.2 Method: get_entity

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing. Equivalent to the request_vehicle_identification() method but can be called without instantiation.

param ecu_ip_address This is the IP address of the target ECU for unicast. Defaults to broadcast if

the address is not known. :type ecu_ip_address: str, optional :param protocol_version: The DoIP protocol version to use for communication. Represents the version of the ISO 13400 specification to follow. 0x02 (2012) is probably correct for most ECU's at the time of writing, though technically this implementation is against 0x03 (2019). :type protocol_version: int, optional :param eid: EID of the Vehicle :type eid: bytes, optional :param vin: VIN of the Vehicle :type vin: str, optional :return: The vehicle identification response message :rtype: VehicleIdentificationResponse robotframework-doip-doipclient-client-doipclient-empty-rxqueue ----------

Implemented for compatibility with udsoncan library. Nothing useful to be done yet robotframework-doip-doipclient-client-doipclient-empty-txqueue ----------

Implemented for compatibility with udsoncan library. Nothing useful to be done yet robotframework-doip-doipclient-client-doipclient-read-doip ----------

Helper function to read from the DoIP socket.

param timeout Maximum time allowed for response from ECU

type timeout float, optional

param transport The IP transport layer to read from, either UDP or TCP

type transport DoIPClient.TransportType, optional

raises IOError If DoIP layer fails with negative acknowledgement

raises TimeoutException If ECU fails to respond in time

7.2.3 Method: send_doip

Helper function to send to the DoIP socket.

Adds the correct DoIP header to the payload and sends to the socket.

param payload_type The payload type (see Table 17 "Overview of DoIP payload types" in ISO-13400

type payload_type int

param transport The IP transport layer to send to, either UDP or TCP

type transport DoIPClient.TransportType, optional

param disable_retry Disables retry regardless of auto_reconnect_tcp flag. This is used by activation requests during connect/reconnect.

type disable_retry bool, optional

7.2.4 Method: send_doip_message

Helper function to send an unpacked message to the DoIP socket.

Packs the given message and adds the correct DoIP header before sending to the socket

param doip_message DoIP message object

type doip_message object

param transport The IP transport layer to send to, either UDP or TCP

type transport DoIPClient.TransportType, optional

param disable_retry Disables retry regardless of auto_reconnect_tcp flag. This is used by activation requests during connect/reconnect.

type disable_retry bool, optional

7.2.5 Method: request_activation

Requests a given activation type from the ECU for this connection using payload type 0x0005

param activation_type The type of activation to request - see Table 47 ("Routing activation request activation types") of ISO-13400, but should generally be 0 (default) or 1 (regulatory diagnostics)

type activation_type RoutingActivationRequest.ActivationType

param vm_specific Optional 4 byte long int

type vm_specific int, optional

param disable_retry Disables retry regardless of auto_reconnect_tcp flag. This is used by activation requests during connect/reconnect.

type disable_retry bool, optional

return The resulting activation response object

rtype RoutingActivationResponse

7.2.6 Method: request_vehicle_identification

Sends a VehicleIdentificationRequest and awaits a VehicleIdentificationResponse from the ECU, either with a specified VIN, EIN, or nothing. :param eid: EID of the Vehicle :type eid: bytes, optional :param vin: VIN of the Vehicle :type vin: str, optional :return: The vehicle identification response message :rtype: VehicleIdentificationResponse robotframework-doip-doipclient-client-doipclient-request-alive-check ----------

Request that the ECU send an alive check response

return Alive Check Response object

rtype AliveCheckResopnse

7.2.7 Method: request_diagnostic_power_mode

Request that the ECU send a Diagnostic Power Mode response

return Diagnostic Power Mode Response object

rtype DiagnosticPowerModeResponse

7.2.8 Method: request_entity_status

Request that the ECU send a DoIP Entity Status Response

return DoIP Entity Status Response

rtype EntityStatusResponse

7.2.9 Method: send_diagnostic

Send a raw diagnostic payload (ie: UDS) to the ECU.

param diagnostic_payload UDS payload to transmit to the ECU

type diagnostic_payload bytearray

raises IOError DoIP negative acknowledgement received

7.2.10 Method: receive_diagnostic

Receive a raw diagnostic payload (ie: UDS) from the ECU.

return Raw UDS payload

rtype bytearray

raises TimeoutError No diagnostic response received in time

7.2.11 Method: close

Close the DoIP client robotframework-doip-doipclient-client-doipclient-reconnect ----------

Attempts to re-establish the connection. Useful after an ECU reset

param close_delay Time to wait between closing and re-opening socket

type close_delay float, optional

Chapter 8

connectors.py

8.1 Class: DoIPClientUDSConnector

Imported by:

```
from RobotFramework_DoIP.doipclient.connectors import DoIPClientUDSConnector
```

A udsoncan connector which uses an existing DoIPClient as a DoIP transport layer for UDS (instead of ISO-TP).

param doip_layer The DoIP Transport layer object coming from the `doipclient` package.

type doip_layer `doipclient.DoIPClient<python_doip.DoIPClient>`

param name This name is included in the logger name so that its output can be redirected. The logger name will be `Connection[<name>]`

type name string

param close_connection True if the wrapper's close() function should close the associated DoIP client. This is not the default

type name bool

8.1.1 Method: open

8.1.2 Method: close

8.1.3 Method: is_open

8.1.4 Method: specific_send

8.1.5 Method: specific_wait_frame

8.1.6 Method: empty_rxqueue

8.1.7 Method: empty_txqueue

Chapter 9

messages.py

9.1 Class: DoIPMessage

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import DoIPMessage
```

Base class for DoIP messages implementing common features like comparison, and representation robotframework-doip-doipclient-messages-reservedmessage ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import ReservedMessage
```

DoIP message whose payload ID is reserved either for manufacturer use or future expansion of DoIP protocol robotframework-doip-doipclient-messages-reservedmessage-unpack ----------

9.1.1 Method: pack

9.1.2 Method: payload

Raw payload bytes robotframework-doip-doipclient-messages-reservedmessage-payload-type ----------

Raw payload type (ID) robotframework-doip-doipclient-messages-genericdoipnegativeacknowledge ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import GenericDoIPNegativeAcknowledge
```

Generic header negative acknowledge structure. See Table 18 robotframework-doip-doipclient-messages-genericdoipnegativeacknowledge-unpack ----------

9.1.3 Method: pack

9.1.4 Method: nack_code

Generic DoIP header NACK code

Description: "The generic header negative acknowledge code indicates the specific error, detected in the generic DoIP header, or it indicates an unsupported payload or a memory overload condition." robotframework-doip-doipclient-messages-alivecheckrequest ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import AliveCheckRequest
```

Alive check request - Table 27 robotframework-doip-doipclient-messages-alivecheckrequest-unpack ----------

9.1.5 Method: pack

9.2 Class: AliveCheckResponse

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import AliveCheckResponse
```

Alive check resopnse - Table 28 robotframework-doip-doipclient-messages-alivecheckresponse-unpack ----------

9.2.1 Method: pack

9.2.2 Method: source_address

Source address (SA)
Description: "Contains the logical address of the client DoIP entity that is currently active on this TCP_DATA socket"
Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-doipentitystatusrequest ==========
Imported by:

```
from RobotFramework_DoIP.doipclient.messages import DoipEntityStatusRequest
```

DoIP entity status request - Table 10 robotframework-doip-doipclient-messages-doipentitystatusrequest-unpack ----------

9.2.3 Method: pack

9.3 Class: DiagnosticPowerModeRequest

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import DiagnosticPowerModeRequest
```

Diagnostic power mode information request - Table 8 robotframework-doip-doipclient-messages-diagnosticpowermoderequest-unpack ----------

9.3.1 Method: pack

9.4 Class: DiagnosticPowerModeResponse

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import DiagnosticPowerModeResponse
```

Diagnostic power mode information response - Table 9 robotframework-doip-doipclient-messages-diagnosticpowermoderesponse-unpack ----------

9.4.1 Method: pack

9.4.2 Method: diagnostic_power_mode

Diagnostic power mode

Description: "Identifies whether or not the vehicle is in diagnostic power mode and ready to perform reliable diagnostics. robotframework-doip-doipclient-messages-routingactivationrequest ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import RoutingActivationRequest
```

Routing activation request. Table 46 robotframework-doip-doipclient-messages-routingactivationrequest-unpack ----------

9.4.3 Method: pack

9.4.4 Method: source_address

Source address (SA)

Description: "Address of the client DoIP entity that requests routing activation. This is the same address that is used by the client DoIP entity when sending diagnostic messages on the same TCP_DATA socket."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-routingactivationrequest-activation-type ----------

Activation type

Description: "Indicates the specific type of routing activation that may require different types of authentication and/or confirmation." robotframework-doip-doipclient-messages-routingactivationrequest-reserved ----------

Reserved - should be 0x00000000 robotframework-doip-doipclient-messages-routingactivationrequest-vm-specific ----------

Reserved for VM-specific use robotframework-doip-doipclient-messages-vehicleidentificationrequest ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import VehicleIdentificationRequest
```

Vehicle identification request message. See Table 2 robotframework-doip-doipclient-messages-vehicleidentificationrequest-unpack ----------

9.4.5 Method: pack

9.5 Class: VehicleIdentificationRequestWithEID

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import
↪  VehicleIdentificationRequestWithEID
```

Vehicle identification request message with EID. See Table 3 robotframework-doip-doipclient-messages-vehicleidentificationreques unpack ----------

9.5.1 Method: pack

9.5.2 Method: eid

EID

Description: "This is the DoIP entity's unique ID (e.g. network interface's MAC address) that shall respond to the vehicle identification request message." robotframework-doip-doipclient-messages-vehicleidentificationrequestwithvin ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import
↪  VehicleIdentificationRequestWithVIN
```

Vehicle identification request message with VIN. See Table 4 robotframework-doip-doipclient-messages-vehicleidentificationreques unpack ----------

9.5.3 Method: pack

9.5.4 Method: vin

VIN

Description: "This is the vehicle's identification number asspecified in ISO 3779. This parameter is only present if the client DoIP entity intends toidentify the DoIP entities of an individual vehicle, the VIN of which is known to the client DoIP entity."

Values: ASCII robotframework-doip-doipclient-messages-routingactivationresponse ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import RoutingActivationResponse
```

Payload type routing activation response. robotframework-doip-doipclient-messages-routingactivationresponse-unpack ----------

9.5.5 Method: pack

9.5.6 Method: client_logical_address

Logical address of client DoIP entity

Description: "Logical address of the client DoIP entity that requested routing activation."
Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-routingactivationresponse-logical-address ----------

Logical address of DoIP entity

Description: "Logical address of the responding DoIP entity."

Values: See client_logical_address robotframework-doip-doipclient-messages-routingactivationresponse-response-code ----------

Routing activation response code

Description: "Response by the DoIP gateway. Routing activation denial results in the TCP_DATA connection being reset by the DoIP gateway. Successful routing activation implies that diagnostic messages can now be routed over the TCP_DATA connection. robotframework-doip-doipclient-messages-routingactivationresponse-reserved ----------

Reserved value - 0x00000000 robotframework-doip-doipclient-messages-routingactivationresponse-vm-specific ----------

Reserved for VM-specific use

Description: "Available for additional VM-specific use." robotframework-doip-doipclient-messages-diagnosticmessage ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import DiagnosticMessage
```

Diagnostic Message - see Table 21 "Payload type diagnostic message structure"

Description: Wrapper for diagnostic (UDS) payloads. The same message is used for TX and RX, and the ECU will confirm receipt with either a DiagnosticMessageNegativeAcknowledgement or a DiagnosticMessagePositiveAcknowledgement message robotframework-doip-doipclient-messages-diagnosticmessage-unpack ----------

9.5.7 Method: pack

9.5.8 Method: source_address

Source address (SA)

Description: "Contains the logical address of the sender of a diagnostic messag (e.g. the client DoIP entity address)."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-diagnosticmessage-target-address ----------

Target address (TA)

Description: "Contains the logical address of the receiver of a diagnostic message (e.g. a specific server DoIP entity on the vehicle's networks)."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-diagnosticmessage-user-data ----------

User data (UD)

Description: Contains the actual diagnostic data (e.g. ISO 14229-1 diagnostic request), which shall be routed to the destination (e.g. the ECM).

Values: Bytes/Bytearray robotframework-doip-doipclient-messages-diagnosticmessagenegativeacknowledgement ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import
↪  DiagnosticMessageNegativeAcknowledgement
```

A negative acknowledgement of the previously received diagnostic (UDS) message.

Indicates that the previously received diagnostic message was rejected. Reasons could include a message being too large, incorrect logical addresses, etc.

See Table 25 - "Payload type diagnostic message negative acknowledgment structure" robotframework-doip-doipclient-messages-diagnosticmessagenegativeacknowledgement-unpack ----------

9.5.9 Method: pack

9.5.10 Method: source_address

Source address (SA)

Description: "Contains the logical address of the (intended) receiver of the previous diagnostic message (e.g. a specific server DoIP entity on the vehicle's networks)."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-diagnosticmessagenegativeacknowledgement-target-address ----------

Target address (TA)

Description: "Contains the logical address of the sender of the previous diagnostic message (i.e. the client DoIP entity address)."

Values: (See source_address) robotframework-doip-doipclient-messages-diagnosticmessagenegativeacknowledgement-nack-code ----------

NACK code

Indicates the reason the diagnostic message was rejected robotframework-doip-doipclient-messages-diagnosticmessagenegativeack previous-message-data ----------

Previous diagnostic message data

An optional copy of the diagnostic message which is being acknowledged. robotframework-doip-doipclient-messages-diagnosticmessagepositiveacknowledgement ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import
↪  DiagnosticMessagePositiveAcknowledgement
```

A positive acknowledgement of the previously received diagnostic (UDS) message.

"...indicates a correctly received diagnostic message, which is processed and put into the transmission buffer of the destination network."

See Table 23 - "Payload type diagnostic message acknowledgement structure" robotframework-doip-doipclient-messages-diagnosticmessagepositiveacknowledgement-unpack ----------

9.5.11 Method: pack

9.5.12 Method: source_address

Source address (SA)

Description: "Contains the logical address of the (intended) receiver of the previous diagnostic message (e.g. a specific server DoIP entity on the vehicle's networks)."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-diagnosticmessagepositiveacknowledgement-target-address ----------

Target address (TA)

Description: "Contains the logical address of the sender of the previous diagnostic message (i.e. the client DoIP entity address)."

Values: (See source_address) robotframework-doip-doipclient-messages-diagnosticmessagepositiveacknowledgement-ack-code ----------

ACK code

Values: Required to be 0x00. All other values are reserved robotframework-doip-doipclient-messages-diagnosticmessagepositiveac previous-message-data ----------

Previous diagnostic message data

An optional copy of the diagnostic message which is being acknowledged. robotframework-doip-doipclient-messages-entitystatusresponse ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import EntityStatusResponse
```

DoIP entity status response. Table 11 robotframework-doip-doipclient-messages-entitystatusresponse-unpack ----------

9.5.13 Method: pack

9.5.14 Method: node_type

Node type(NT)

Description: "Identifies whether the contacted DoIP instance is either a DoIP node or a DoIP gateway."

Values:

- 0x00: DoIP gateway
- 0x01: DoIP node

* 0x02 .. 0xFF: reserved robotframework-doip-doipclient-messages-entitystatusresponse-max-concurrent-sockets ----------

Max. concurrent TCP_DATA sockets (MCTS)

Description: "Represents the maximum number of concurrent TCP_DATA sockets allowed with this DoIP entity, excluding the reserve socket required for socket handling."

Values: 1 to 255 robotframework-doip-doipclient-messages-entitystatusresponse-currently-open-sockets ----------

Currently open TCP_DATA sockets (NCTS)

Description: "Number of currently established sockets."

Values: 0 to 255 robotframework-doip-doipclient-messages-entitystatusresponse-max-data-size ----------

Max. data size (MDS)

Description: "Maximum size of one logical request that this DoIP entity can process."

Values: 0 to 4GB robotframework-doip-doipclient-messages-vehicleidentificationresponse ==========

Imported by:

```
from RobotFramework_DoIP.doipclient.messages import VehicleIdentificationResponse
```

Payload type vehicle announcement/identification response message Table 5 robotframework-doip-doipclient-messages-vehicleidentificationresponse-unpack ----------

9.5.15 Method: pack

9.5.16 Method: vin

VIN

Description: "This is the vehicle's VIN as specified in ISO 3779. If the VIN is not configured at the time of transmission of this message, this should be indicated using the invalidity value {0x00 or 0xff}... In this case, the GID is used to associate DoIP nodes with a certain vehicle..."

Values: ASCII robotframework-doip-doipclient-messages-vehicleidentificationresponse-logical-address ----------

Logical Address

Description: "This is the logical address that is assigned to the responding DoIP entity (see 7. 8 for further details). The logical address can be used, for example, to address diagnostic requests directly to the DoIP entity."

Values: From Table 13

- 0x0000 = ISO/SAE reserved
- 0x0001 to 0x0DFF = VM specific
- 0x0E00 to 0x0FFF = Reserved for addresses of client
- 0x1000 to 0x7FFF = VM Specific
- 0x8000 to 0xE3FF = Reserved
- 0xE400 to 0xE3FF = VM defined functional group logical addresses

* 0xF000 to 0xFFFF = Reserved robotframework-doip-doipclient-messages-vehicleidentificationresponse-eid ----------

EID

Description: "This is a unique identification of the DoIP entities in order to separate their responses even before the VIN is programmed to, or recognized by, the DoIP devices (e.g. during the vehicle assembly process). It is recommended that the MAC address information of the DoIP entity's network interface be used (one of the interfaces if multiple network interfaces are implemented)."

Values: "Not set" values are 0x00 or 0xff. robotframework-doip-doipclient-messages-vehicleidentificationresponse-gid ----------

GID

Description: "This is a unique identification of a group of DoIP entities within the same vehicle in the case that a VIN is not configured for that vehicle... If the GID is not available at the time of transmission of this message, this shall be indicated using the specific invalidity" ("not set") value of 0x00 or 0xff. robotframework-doip-doipclient-messages-vehicleidentificationresponse-further-action-required ----------

Further action required

Description: "This is the additional information to notify the client DoIP entity that there are either DoIP entities with no initial connectivity or that a centralized security approach is used." robotframework-doip-doipclient-messages-vehicleidentificationresponse-vin-sync-status ----------

VIN/GID sync. status

Description: "This is the additional information to notify the client DoIP entity that all DoIP entities have synchronized their information about the VIN or GID of the vehicle"

Chapter 10

Appendix

About this package:

Table 10.1: Package setup

Setup parameter	Value
Name	RobotFramework_DoIP
Version	0.1.0
Date	20.09.2023
Description	RobotFramework for DoIP Client
Package URL	robotframework-doip
Author	Hua Van Thong
Email	thong.huavan@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 11

History

0.1.0	09/2023
Initial version	
0.1.1	12/2023
Add ecu simulator to use for self test	

RobotFramework_DoIP.pdf
Created at 17.01.2024 - 17:53:34
by GenPackageDoc v. 0.41.1