RobotFramework_UDS

v. 0.1.5

Mai Minh Tri

10.10.2024

Contents

Chapter 1

Introduction

The library **RobotFramework_UDS** provides a set of **Robot Framework** keywords for sending UDS (Unified Diagnostic Services) requests and interpreting responses from automotive electronic control units (ECUs).

Whether you're testing diagnostic sessions, reading data, or controlling routines on an ECU, the UDS Library simplifies these tasks by offering specific keywords like `DiagnosticSessionControl` , `ReadDataByIdentifier` , and `RoutineControl` .

These keywords are designed to handle the complexity of UDS communication, enabling you to write efficient and reliable automated tests.

Moreover, you can now refer to UDS services by their readable names rather than hexadecimal IDs e.g `ReadDataByName` , `RoutineControlByName` It helps to make your tests more intuitive and easier to maintain.

Chapter 2

Description

2.1 Overview

The **RobotFramework_UDS** is designed to interface with automotive ECUs using the UDS protocol over the DoIP (Diagnostic over IP) transport layer. This library abstracts the complexities of UDS communication, allowing users to focus on writing high-level test cases that validate specific diagnostic services and responses.

2.2 UDS Connector (DoIP)

Currently, the library supports the `DoIP` (Diagnostic over IP) transport layer, which is commonly used in modern vehicles for diagnostic communication. DoIP allows for faster data transfer rates and easier integration with network-based systems compared to traditional CAN-based diagnostics.

2.3 Configuration

In order to connect and send/receive message properly using the **RobotFramework_UDS** certain configurations must be set up:

- DoIP Configuration: The library requires the IP address and port of the ECU or the gateway through which the ECU is accessed.
- Data Identifiers and Codec: Define the Data Identifiers (DIDs) and corresponding codecs in the library's configuration. This enables correct encoding and decoding of data between the test cases and the ECU.
- Session Management: Some UDS services may require the ECU to be in a specific diagnostic session (e.g., extended diagnostics). The library should be configured to manage these session transitions seamlessly.

2.4 Supported UDS Services

The **RobotFramework_UDS** library supports almost UDS service as defined in ISO 14229, providing comprehensive coverage for ECU diagnostics.

For detailed information on specific services and how to use them, please refer to the next section.

2.5 Enhancements Usability with ODXTools Integration

The **RobotFramework_UDS** library comes with odxtools fully integrated, allowing you to use readable service names instead of dealing with hex IDs.

You can now specify service names directly in your test cases, making them more readable and user-friendly.

The **RobotFramework_UDS** also supports automatic data type conversion for request parameters, enabling users to provide input values that are seamlessly converted into the correct data types required for UDS requests.

The **RobotFramework_UDS** library supports features such as Routine Control By Name, Read Data By Name, and Write Data By Name, enabling users to send requests to UDS using the service's name instead of its identifier.

Examples

Routine Control By Name

```
Test user can use Routine Control By Name service on ECU
    Log    Routine Control By Name service: StartIperfServer_Start

    ${param_dict}=    Create Dictionary    port=5101    argument=-i 0.5 -B 192.168.1.
    ${response}=    Routine Control By Name    StartIperfServer_Start    ${param_dict}

    Log    ${response}    console=True
    FOR    ${item}    IN    @{response.keys()}
        Log    ${item} : ${response["${item}"]}    console=True
    END
```

Read Data By Name

```
Test user can use Read Data By Name service on ECU
    Log    Use Read Data By Name service

    Log    Use Read Data By Identifier - 25392

    ${list_identifers}=    Create List    0x6330
    ${res}=    Read Data By Identifier    ${list_identifers}
    Log    ${res}    console=True
    Log    ${res[0x6330]}    console=True

    Log    readCPUClockFrequencies_Read

    ${service_name_list}=    Create List    readCPUClockFrequencies_Read
    ${responses}=    Read Data By Name    ${service_name_list}
    Log    ${responses}    console=True

    FOR    ${request_did}    IN    @{responses.keys()}
        Log    Key: ${request_did}, Value: ${responses["${request_did}"]}    console=True
        ${response}=    Set Variable    ${responses["${request_did}"]}
        FOR    ${item}    IN    @{response.keys()}
            Log    ${item} : ${response["${item}"]}    console=True
        END
    END

Test Read List Services
    ${list_read_services}=    Create List    TestManager_SWVersion_Read
    ...                                      CTSSWVersion_Read
    ...                                      CPULoad_Read

    FOR    ${service_name}    IN    @{list_read_services}
        Log    Read Data of ${service_name}    console=True
        ${list_service}=    Create List    ${service_name}
        ${res}=    Read Data By Name    ${list_service}
        Log    ${res}    console=True
    END
```

Write Data By Name

```
Test user can use Write Data By Name service on ECU
    Log    Write Data By Name service

    Log    RealTimeClock_Write
    ${PARAM_DICT}=    Create Dictionary    Day=26    Month=September    Year=2024    ↩
    ↪ Hour=10    Second=45    Minute=0
    ${res}=    Write Data By Name    RealTimeClock_Write    ${PARAM_DICT}
    Log    ${res}    console=True

    ${DICT}=    Create Dictionary    ipAddress=155
    ${res}=    Write Data By Name    CTS_IPAddress_Write    ${DICT}
```

```
    Log    ${res}    console=True

    Log    Using service did instead service's name
    ${res}=    Write Data By Identifier    25382    ${PARAM_DICT}
    Log    ${res}    console=True
```

2.6 Multiple Connections

The **RobotFramework_UDS** can be extended to manage multiple connections simultaneously. This is beneficial when working with complex vehicle systems or simultaneously testing multiple ECUs.

2.7 Examples

Single Connection

```
*** Settings ***
Library    RobotFramework_TestsuitesManagement    WITH NAME    testsuites
Library    RobotFramework_UDS

*** Variables ***
${SUT_IP_ADDRESS_1}=    192.168.0.7
${SUT_LOGICAL_ADDRESS_1}=    1863
${TB_IP_ADDRESS_1}=    192.168.0.240
${TB_LOGICAL_ADDRESS_1}=    1895
${ACTIVATION_TYPE_1}=    0
${DEVICE_NAME_1}=    UDS Connector 1
${FILE_1}=    C:/Users/MAR3HC/Desktop/UDS/robotframework-uds/test/pdx/CTS_STLA_V1_15_2.pdx
${VARIANT_1}=    CTS_STLA_Brain

*** Test Cases ***
Test user can connect single UDS connection
    Log    Test user can connect single UDS connection
    Log    If no device_name is provided, it will default to 'default'

    Create UDS Connector    ecu_ip_address= ${SUT_IP_ADDRESS_1}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_1}
    ...                     client_ip_address= ${TB_IP_ADDRESS_1}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_1}
    ...                     activation_type= ${ACTIVATION_TYPE_1}

    Connect UDS Connector
    Open UDS Connection
    Load PDX    ${FILE_1}    ${VARIANT_1}
    ${service_name_list}=    Create List    readCPUClockFrequencies_Read
    Read Data By Name    ${service_name_list}
    Close UDS Connection
```

Multiple Connections

```
*** Variables ***
${SUT_IP_ADDRESS_1}=    192.168.0.7
${SUT_LOGICAL_ADDRESS_1}=    1863
${TB_IP_ADDRESS_1}=    192.168.0.240
${TB_LOGICAL_ADDRESS_1}=    1895
${ACTIVATION_TYPE_1}=    0
${DEVICE_NAME_1}=    UDS Connector 1
${FILE_1}=    C:/Users/MAR3HC/Desktop/UDS/robotframework-uds/test/pdx/CTS_STLA_V1_15_2.pdx
${VARIANT_1}=    CTS_STLA_Brain

${SUT_IP_ADDRESS_2}=    192.168.0.4
${SUT_LOGICAL_ADDRESS_2}=    1863
${TB_IP_ADDRESS_2}=    192.168.0.240
${TB_LOGICAL_ADDRESS_2}=    1895
${ACTIVATION_TYPE_2}=    0
```

```
${DEVICE_NAME_2}=    UDS Connector 2
${FILE_2}=    C:/Data/Git/mpci_rack_nau2kor/project/testbench/hi1/uds/XTS_S32G_1.0.356.pdx
${VARIANT_2}=    XTS_S32G

${ERROR_STR}=    NegativeResponseException: ReadDataByIdentifier service execution returned a negative response IncorrectMessageLengthOrInvalidFormat (0x13)

*** Test Cases ***
Test user can connect multiple UDS connection
    Log    Test user can connect multiple UDS connection
    Log    Connect to device 1
    Create UDS Connector    device_name= ${DEVICE_NAME_1}
    ...                     ecu_ip_address= ${SUT_IP_ADDRESS_1}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_1}
    ...                     client_ip_address= ${TB_IP_ADDRESS_1}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_1}
    ...                     activation_type= ${ACTIVATION_TYPE_1}
    Connect UDS Connector   device_name= ${DEVICE_NAME_1}

    Open UDS Connection    device_name= ${DEVICE_NAME_1}
    Load PDX    ${FILE_1}    ${VARIANT_1}    device_name= ${DEVICE_NAME_1}
    ${service_name_list_1}=    Create List    readCPUClockFrequencies_Read
    Read Data By Name    ${service_name_list_1}    device_name= ${DEVICE_NAME_1}

    Log    Connect to device 2
    Create UDS Connector    device_name= ${DEVICE_NAME_2}
    ...                     ecu_ip_address= ${SUT_IP_ADDRESS_2}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_2}
    ...                     client_ip_address= ${TB_IP_ADDRESS_2}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_2}
    ...                     activation_type= ${ACTIVATION_TYPE_2}
    Connect UDS Connector    device_name= ${DEVICE_NAME_2}

    Open UDS Connection    device_name= ${DEVICE_NAME_2}
    Load PDX    ${FILE_2}    ${VARIANT_2}    device_name= ${DEVICE_NAME_2}
    ${service_name_list_2}=    Create List    CPULoad_Read
    Log    Expected device 2 cannot send readCPUClockFrequencies_Read service like device 1
    Run Keyword And Expect Error    ${ERROR_STR}    Read Data By Name    ${service_name_list_1}    device_name= ${DEVICE_NAME_2}

    Read Data By Name    ${service_name_list_2}    device_name= ${DEVICE_NAME_2}

Test user can connect multiple UDS connection but connect to the same ECU
    Log    Test user can connect multiple UDS connection
    Log    Connect to device 1
    Create UDS Connector    device_name= ${DEVICE_NAME_1}
    ...                     ecu_ip_address= ${SUT_IP_ADDRESS_1}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_1}
    ...                     client_ip_address= ${TB_IP_ADDRESS_1}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_1}
    ...                     activation_type= ${ACTIVATION_TYPE_1}
    Connect UDS Connector    device_name= ${DEVICE_NAME_1}

    Log    Open uds connection
    Open UDS Connection    device_name= ${DEVICE_NAME_1}
    Load PDX    ${FILE_1}    ${VARIANT_1}    device_name= ${DEVICE_NAME_1}
    ${service_name_list_1}=    Create List    readCPUClockFrequencies_Read
    Read Data By Name    ${service_name_list_1}    device_name= ${DEVICE_NAME_1}

    Log    Connect to device 2 but same IP as device 1
    Log    The expected test case result in an error
    Run Keyword And Expect Error    TimeoutError: ECU failed to respond in time    Create UDS Connector    device_name= ${DEVICE_NAME_2}
    ...
    ...                 ecu_ip_address= ${SUT_IP_ADDRESS_1}
    ...
    ...                 ecu_logical_address= ${SUT_LOGICAL_ADDRESS_1}
```

```
    ...                                                                    ↩
    ↪                   client_ip_address= ${TB_IP_ADDRESS_1}
    ...                                                                    ↩
    ↪                   client_logical_address= ${TB_LOGICAL_ADDRESS_1}
    ...                                                                    ↩
    ↪                   activation_type= ${ACTIVATION_TYPE_1}

Test users can reconnect to the closed ECU
    Log    Connect to device 2
    Create UDS Connector    device_name= ${DEVICE_NAME_2}
    ...                     ecu_ip_address= ${SUT_IP_ADDRESS_2}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_2}
    ...                     client_ip_address= ${TB_IP_ADDRESS_2}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_2}
    ...                     activation_type= ${ACTIVATION_TYPE_1}
    Connect UDS Connector    device_name= ${DEVICE_NAME_2}

    Open UDS Connection    device_name= ${DEVICE_NAME_2}
    Load PDX    ${FILE_2}    ${VARIANT_2}    device_name= ${DEVICE_NAME_2}
    ${service_name_list_2}=    Create List    CPULoad_Read
    Read Data By Name    ${service_name_list_2}    device_name= ${DEVICE_NAME_2}
    Close UDS Connection    device_name= ${DEVICE_NAME_2}

    Log    Connect to device 1
    Create UDS Connector    device_name= ${DEVICE_NAME_1}
    ...                     ecu_ip_address= ${SUT_IP_ADDRESS_1}
    ...                     ecu_logical_address= ${SUT_LOGICAL_ADDRESS_1}
    ...                     client_ip_address= ${TB_IP_ADDRESS_1}
    ...                     client_logical_address= ${TB_LOGICAL_ADDRESS_1}
    ...                     activation_type= ${ACTIVATION_TYPE_1}
    Connect UDS Connector    device_name= ${DEVICE_NAME_1}

    Open UDS Connection    device_name= ${DEVICE_NAME_1}
    Load PDX    ${FILE_1}    ${VARIANT_1}    device_name= ${DEVICE_NAME_1}
    ${service_name_list_1}=    Create List    readCPUClockFrequencies_Read
    Read Data By Name    ${service_name_list_1}    device_name= ${DEVICE_NAME_1}
    Close UDS Connection    device_name= ${DEVICE_NAME_1}


    Log    Re-opent uds connection device 2
    Open UDS Connection    device_name= ${DEVICE_NAME_2}
    Read Data By Name    ${service_name_list_2}    device_name= ${DEVICE_NAME_2}

    Log    Expected device 2 cannot send readCPUClockFrequencies_Read service like ↩
    ↪ device 1
    Run Keyword And Expect Error    ${ERROR_STR}    Read Data By Name    ↩
    ↪ ${service_name_list_1}    device_name= ${DEVICE_NAME_2}
    Close UDS Connection    device_name= ${DEVICE_NAME_2}
```

Routine Control By Name

```
Test user can use Routine Control By Name service on ECU
    Log    Routine Control By Name service: StartIperfServer_Start

    ${param_dict}=    Create Dictionary    port=5101    argument=-i 0.5 -B 192.168.1.
    ${response}=    Routine Control By Name    StartIperfServer_Start    ${param_dict}

    Log    ${response}    console=True
    FOR    ${item}    IN    @{response.keys()}
        Log    ${item} : ${response["${item}"]}    console=True
    END
```

Read Data By Name

```
Test user can use Read Data By Name service on ECU
    Log    Use Read Data By Name service

    Log    Use Read Data By Identifier - 25392
```

```
    ${list_identifers}=    Create List    0x6330
    ${res}=    Read Data By Identifier    ${list_identifers}
    Log    ${res}    console=True
    Log    ${res[0x6330]}    console=True

    Log    readCPUClockFrequencies_Read

    ${service_name_list}=    Create List    readCPUClockFrequencies_Read
    ${responses}=    Read Data By Name    ${service_name_list}
    Log    ${responses}    console=True

    FOR    ${request_did}    IN    @{responses.keys()}
        Log    Key: ${request_did}, Value: ${responses["${request_did}"]}    console=True
        ${response}=    Set Variable    ${responses["${request_did}"]}
        FOR    ${item}    IN    @{response.keys()}
            Log    ${item} : ${response["${item}"]}    console=True
        END
    END

Test Read List Services
    ${list_read_services}=    Create List    TestManager_SWVersion_Read
    ...                                      CTSSWVersion_Read
    ...                                      CPULoad_Read

    FOR    ${service_name}    IN    @{list_read_services}
        Log    Read Data of ${service_name}    console=True
        ${list_service}=    Create List    ${service_name}
        ${res}=    Read Data By Name    ${list_service}
        Log    ${res}    console=True
    END
```

Write Data By Name

```
Test user can use Write Data By Name service on ECU
    Log    Write Data By Name service

    Log    RealTimeClock_Write
    ${PARAM_DICT}=    Create Dictionary    Day=26    Month=September    Year=2024    ↩
    ↪ Hour=10    Second=45    Minute=0
    ${res}=    Write Data By Name    RealTimeClock_Write    ${PARAM_DICT}
    Log    ${res}    console=True

    ${DICT}=    Create Dictionary    ipAddress=155
    ${res}=    Write Data By Name    CTS_IPAddress_Write    ${DICT}
    Log    ${res}    console=True

    Log    Using service did instead service's name
    ${res}=    Write Data By Identifier    25382    ${PARAM_DICT}
    Log    ${res}    console=True
```

Get Encoded Request Message

```
Test Get Encoded Request Message - Simple request parameters
    Log    Test Get Encoded Request Message - mainCPUStressTest_Start
    ${service_name}=    Set Variable    mainCPUStressTest_Start
    ${param_dict}=    Create Dictionary    cores=5    load=50
    ${res}=    Get Encoded Request Message    ${service_name}    ${param_dict}
    Log    ${res}    console=True
    Log    ${res.hex()}    console=True

Test Get Encoded Request Message - Complex request parameters
    Load PDX    ↩
    ↪ C:/Users/MAR3HC/Desktop/UDS/robotframework-uds/test/pdx/XTS_MPCI_Maas_1.23.45.pdx  ↩
    ↪   XTS_MPCI_MaaS

    Log    Test Get Encoded Request Message - CAN_MasterSlaveEnduranceRun_Start
    ${service_name}=    Set Variable    CAN_MasterSlaveEnduranceRun_Start
```

```
    ${res}=    Get Encoded Request Message    ${service_name}    ${canConfig}
    Log    ${res}    console=True
    Log    ${res.hex()}    console=True
```

Get Decoded Request Message

```
Test Get Decoded Response Message
    Log     Test Get Decoded Response Message
    ${service_name}=    Set Variable    StartIperfServer_Start
    ${response_str}=    Set Variable    \x012#\x00\x00\x18\x05H
    ${response_byte}=    Convert To Bytes    ${response_str}
    ${res}=    Get Decoded Response Message    ${service_name}    ${response_byte}
    Log    ${res}    console=True
```

Chapter 3

DiagnosticServices.py

3.1 Class: DiagnosticServices

Imported by:

```
from RobotFramework_UDS.DiagnosticServices import DiagnosticServices
```

3.1.1 Method: convert_sub_param

Recursive convert sub parameters in given request to correct data type

3.1.2 Method: convert_request_data_type

Convert given request parameters (dictionary) to correct data type

3.1.3 Method: get_diag_service_by_name

3.1.4 Method: get_encoded_request_message

3.1.5 Method: get_decode_response_message

3.1.6 Method: get_full_positive_response_data

3.1.7 Method: get_did_codec

3.2 Class: PDXCodec

Imported by:

```
from RobotFramework_UDS.DiagnosticServices import PDXCodec
```

3.2.1 Method: decode

3.2.2 Method: encode

Chapter 4

UDSKeywords.py

4.1 Class: UDSDeviceManager

Imported by:

```
from RobotFramework_UDS.UDSKeywords import UDSDeviceManager
```

4.1.1 Method: is_device_exist

4.2 Class: UDSDevice

Imported by:

```
from RobotFramework_UDS.UDSKeywords import UDSDevice
```

4.3 Class: UDSKeywords

Imported by:

```
from RobotFramework_UDS.UDSKeywords import UDSKeywords
```

4.3.1 Method: connect_uds_connector

4.3.2 Method: create_uds_connector

Description: Create a connection to establish

Parameters:

- param `comunication_name`: Name of communication
  - doip: Establish a doip connection to an (ECU)
- type `comunication_name`: str
- **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`: any
- 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

4.3.3 Method: load_pdx

Load PDX **Arguments:** * `pdx_file`

/ *Type*: str /
PDX file path

- `variant`
  / *Type*: str /

4.3.4 Method: create_config

Description: Create a config for UDS connector

Parameters:

- param 'exception_on_negative_response': When set to True, the client will raise a NegativeResponseException when the server responds with a negative response. When set to False, the returned Response will have its property positive set to False
- type 'exception_on_negative_response': bool
- param 'exception_on_invalid_response': When set to True, the client will raise a InvalidResponseException when the underlying service interpret_response raises the same exception. When set to False, the returned Response will have its property valid set to False
- type 'exception_on_invalid_response': bool
- param 'exception_on_unexpected_response': When set to True, the client will raise a UnexpectedResponseException when the server returns a response that is not expected. For instance, a response for a different service or when the subfunction echo doesn't match the request. When set to False, the returned Response will have its property unexpected set to True in the same case.
- type 'exception_on_unexpected_response': bool
- param 'security_algo': The implementation of the security algorithm necessary for the SecurityAccess service.

- type 'security_algo': This function must have the following signatures:
  SomeAlgorithm(level, seed, params)

  Parameters:

  - level (int) - The requested security level.
  - seed (bytes) - The seed given by the server
  - params - The value provided by the client configuration security_algo_params

  Returns: The security key Return type: byte
- param 'security_algo_params': This value will be given to the security algorithm defined in config['security_algo'].
- type 'security_algo_params': object | dict
- param 'data_identifiers': This configuration is a dictionary that is mapping an integer (the data identifier) with a DidCodec. These codecs will be used to convert values to byte payload and vice-versa when sending/receiving data for a service that needs a DID, i.e

  - ReadDataByIdentifier
  - ReadDataByName
  - WriteDataByIdentifier
  - ReadDTCInformation with subfunction reportDTCSnapshotRecordByDTCNumber and reportDTCSnapshotRecordByRecordNumber
- type 'data_identifiers': dict Possible configuration values are - string : The string will be used as a pack/unpack string when processing the data - DidCodec (class or instance) : The encode/decode method will be used to process the data
- param 'input_output': This configuration is a dictionary that is mapping an integer (the IO data identifier) with a DidCodec specifically for the InputOutputControlByIdentifier service. Just like config[data_identifers], these codecs will be used to convert values to byte payload and vice-versa when sending/receiving data. Since InputOutputControlByIdentifier supports composite codecs, it is possible to provide a sub-dictionary as a codec specifying the bitmasks.
- type 'input_output': dict Possible configuration values are: - string : The string will be used as a pack/unpack string when processing the data - DidCodec (class or instance) : The encode/decode method will be used to process the data - dict : The dictionary entry indicates a composite DID. Three subkeys must be defined as: - codec : The codec, a string or a DidCodec class/instance - mask : A dictionary mapping the mask name with a bit - mask_size : An integer indicating on how many bytes must the mask be encode

  The special dictionnary key default can be used to specify a fallback codec if an operation is done on a codec not part of the configuration. Useful for scanning a range of DID
- param 'tolerate_zero_padding': This value will be passed to the services interpret_response when the parameter is supported as in ReadDataByIdentifier, ReadDTCInformation. It has to ignore trailing zeros in the response data to avoid falsely raising InvalidResponseException if the underlying protocol uses some zero-padding.
- type 'tolerate_zero_padding': bool
- param ignore_all_zero_dtc This value is used with the ReadDTCInformation service when reading DTCs. It will skip any DTC that has an ID of 0x000000. If the underlying protocol uses zero-padding, it may generate a valid response data of all zeros. This parameter is different from config['tolerate_zero_padding']. Read https://udsoncan.readthedocs.io/en/latest/udsoncan/client.html#configuration for more info.
- type 'ignore_all_zero_dtc': bool
- param 'server_address_format': The MemoryLocation server_address_format is the value to use when none is specified explicitly for methods expecting a parameter of type MemoryLocation.
- type 'server_address_format': int
- param 'server_memorysize_format': The MemoryLocation server_memorysize_format is the value to use when none is specified explicitly for methods expecting a parameter of type MemoryLocation
- type 'server_memorysize_format': int
- param 'extended_data_size': This is the description of all the DTC extended data record sizes. This value is used to decode the server response when requesting a DTC extended data.

The value must be specified as follows config['extended_data_size'] = {
0x123456 : 45, # Extended data for DTC 0x123456 is 45 bytes long
0x123457 : 23 # Extended data for DTC 0x123457 is 23 bytes long
}

- type 'extended_data_size': dict[int] = int
- param 'dtc_snapshot_did_size': The number of bytes used to encode a data identifier specifically for ReadDTCInformation subfunction reportDTCSnapshotRecordByDTCNumber and reportDTCSnapshotRecordByRecordNumber. The UDS standard does not specify a DID size although all other services expect a DID encoded over 2 bytes (16 bits). Default value of 2
- type 'dtc_snapshot_did_size': int
- param 'standard_version': The standard version to use, valid values are : 2006, 2013, 2020. Default value is 2020
- type 'standard_version': int
- param 'request_timeout': Maximum amount of time in seconds to wait for a response of any kind, positive or negative, after sending a request. After this time is elapsed, a TimeoutException will be raised regardless of other timeouts value or previous client responses. In particular even if the server requests that the client wait, by returning response requestCorrectlyReceived-ResponsePending (0x78), this timeout will still trigger. If you wish to disable this behaviour and have your server wait for as long as it takes for the ECU to finish whatever activity you have requested, set this value to None. Default value of 5
- type 'request_timeout': float
- param 'p2_timeout': Maximum amount of time in seconds to wait for a first response (positive, negative, or NRC 0x78). After this time is elapsed, a TimeoutException will be raised if no response has been received. See ISO 14229-2:2013 (UDS Session Layer Services) for more details. Default value of 1
- type 'p2_timeout': float
- param 'p2_star_timeout': Maximum amount of time in seconds to wait for a response (positive, negative, or NRC0x78) after the reception of a negative response with code 0x78 (requestCorrectlyReceived-ResponsePending). After this time is elapsed, a TimeoutException will be raised if no response has been received. See ISO 14229-2:2013 (UDS Session Layer Services) for more details. Default value of 5
- type 'p2_star_timeout': float
- param 'use_server_timing': When using 2013 standard or above, the server is required to provide its P2 and P2* timing values with a DiagnosticSessionControl request. By setting this parameter to True, the value received from the server will be used. When False, these timing values will be ignored and local configuration timing will be used. Note that no timeout value can exceed the config['request_timeout'] as it is meant to avoid the client from hanging for too long. This parameter has no effect when config['standard_version'] is set to 2006. Default value is True
- type 'use_server_timing': bool

4.3.5 Method: set_config

This method sets the UDS config.

Arguments:

- No specific arguments for this method.

Returns:

- `config`
  / *Type*: Configuration /
  Returns the new UDS configuration created by create_configure or the default config if none is provided.

4.3.6 Method: connect

Opens a UDS connection.

Arguments:

- No specific arguments for this method.

4.3.7 Method: disconnect

Closes a UDS connection.

Arguments:

- No specific arguments for this method.

4.3.8 Method: access_timing_parameter

Sends a generic request for AccessTimingParameter service.

Arguments:

- `access_type`
  / *Condition*: required / *Type*: int /
  The service subfunction:
  - readExtendedTimingParameterSet = 1
  - setTimingParametersToDefaultValues = 2
  - readCurrentlyActiveTimingParameters = 3
  - setTimingParametersToGivenValues = 4
- `timing_param_record`
  / *Condition*: optional / *Type*: bytes /
  The parameters data. Specific to each ECU.

Returns:

- `response`
  / *Type*: Response /
  The response from the AccessTimingParameter service request.

4.3.9 Method: clear_dianostic_infomation

Requests the server to clear its active Diagnostic Trouble Codes.

Arguments:

- `group`
  / *Type*: int /
  The group of DTCs to clear. It may refer to Powertrain DTCs, Chassis DTCs, etc. Values are defined by the ECU manufacturer except for two specific values:
  - `0x000000` : Emissions-related systems
  - `0xFFFFFF` : All DTCs
- `memory_selection`
  / *Condition*: optional / *Type*: int /
  MemorySelection byte (0-0xFF). This value is user-defined and introduced in the 2020 version of ISO-14229-1. Only added to the request payload when different from None. Default: None.

Returns:

- `response`
  / *Type*: Response /
  The response from the server after attempting to clear the active Diagnostic Trouble Codes.

4.3.10 Method: communication_control

Switches the transmission or reception of certain messages on/off with CommunicationControl service.

Arguments:

- `control_type`
  / *Condition*: required / *Type*: int /
  The action to request such as enabling or disabling some messages. This value can also be ECU manufacturer-specific:
  - enableRxAndTx = 0
  - enableRxAndDisableTx = 1
  - disableRxAndEnableTx = 2
  - disableRxAndTx = 3
  - enableRxAndDisableTxWithEnhancedAddressInformation = 4
  - enableRxAndTxWithEnhancedAddressInformation = 5
- `communication_type`
  / *Condition*: required / *Type*: CommunicationType<CommunicationType>, bytes, int /
  Indicates what section of the network and the type of message that should be affected by the command. Refer to CommunicationType<CommunicationType> for more details. If an integer or bytes is given, the value will be decoded to create the required CommunicationType<CommunicationType> object.
- `node_id`
  / *Condition*: optional / *Type*: int /
  DTC memory identifier (nodeIdentificationNumber). This value is user-defined and introduced in the 2013 version of ISO-14229-1. Possible only when control type is `enableRxAndDisableTxWithEnhancedAddressInformatic` or `enableRxAndTxWithEnhancedAddressInformation`. Only added to the request payload when different from None. Default: None.

Returns:

- `response`
  / *Type*: Response /
  The response from the CommunicationControl service request.

4.3.11 Method: control_dtc_setting

Controls some settings related to the Diagnostic Trouble Codes by sending a ControlDTCSetting service request. It can enable/disable some DTCs or perform some ECU-specific configuration.

Arguments:

- `setting_type`
  / *Condition*: required / *Type*: int /
  Allowed values are from 0 to 0x7F:
  - on = 1
  - off = 2
  - vehicleManufacturerSpecific = (0x40, 0x5F) # For logging purposes only.
  - systemSupplierSpecific = (0x60, 0x7E) # For logging purposes only.
- `data`
  / *Condition*: optional / *Type*: bytes /
  Optional additional data sent with the request called DTCSettingControlOptionRecord.

Returns:

- `response`
  / *Type*: Response /
  The response from the ControlDTCSetting service request.

4.3.12 Method: diagnostic_session_control

Requests the server to change the diagnostic session with a DiagnosticSessionControl service request.

Arguments:

- `newsession`
  / *Condition*: required / *Type*: int /
  The session to try to switch:
  - defaultSession = 1
  - programmingSession = 2
  - extendedDiagnosticSession = 3
  - safetySystemDiagnosticSession = 4

Returns:

- `response`
  / *Type*: Response /
  The response from the DiagnosticSessionControl service request.

4.3.13 Method: dynamically_define_did

Defines a dynamically defined DID.

Arguments:

- `did`
  / *Type*: int /
  The data identifier to define.
- `did_definition`
  / *Type*: DynamicDidDefinition<DynamicDidDefinition> or MemoryLocation<MemoryLocation> /
  The definition of the DID. Can be defined by source DID or memory address. If a `MemoryLocation<MemoryLocation>` object is given, the definition will automatically be by memory address.

Returns:

- `response`
  / *Type*: Response /
  The response from the request to define the dynamically defined DID.

4.3.14 Method: ecu_reset

Requests the server to execute a reset sequence through the ECUReset service.

Arguments:

- `reset_type`
  / *Condition*: required / *Type*: int /
  The type of reset to perform:
  - hardReset = 1
  - keyOffOnReset = 2
  - softReset = 3
  - enableRapidPowerShutDown = 4
  - disableRapidPowerShutDown = 5

Returns:

- `response`
  / *Type*: Response /
  The response from the ECUReset service request.

4.3.15 Method: io_control

Substitutes the value of an input signal or overrides the state of an output by sending an InputOutputControlByIdentifier service request.

Arguments:

- `did`
  / *Condition*: required / *Type*: int /
  Data identifier to represent the IO.
- `control_param`
  / *Condition*: optional / *Type*: int /
  Control parameters:
  - returnControlToECU = 0
  - resetToDefault = 1
  - freezeCurrentState = 2
  - shortTermAdjustment = 3
- `values`
  / *Condition*: optional / *Type*: list, dict, IOValues<IOValues> /
  Optional values to send to the server. This parameter will be given to DidCodec<DidCodec>.encode() method. It can be:
  - A list for positional arguments
  - A dict for named arguments
  - An instance of IOValues<IOValues> for mixed arguments
- `masks`
  / *Condition*: optional / *Type*: list, dict, IOMask<IOMask>, bool /
  Optional mask record for composite values. The mask definition must be included in `config['input_output']`. It can be:
  - A list naming the bit mask to set
  - A dict with the mask name as a key and a boolean setting or clearing the mask as the value
  - An instance of IOMask<IOMask>
  - A boolean value to set all masks to the same value.

Returns:

- `response`
  / *Type*: Response /
  The response from the InputOutputControlByIdentifier service request.

4.3.16 Method: link_control

Controls the communication baudrate by sending a LinkControl service request.

Arguments:

- `control_type`
  / *Condition*: required / *Type*: int /
  Allowed values are from 0 to 0xFF:
  - verifyBaudrateTransitionWithFixedBaudrate = 1
  - verifyBaudrateTransitionWithSpecificBaudrate = 2
  - transitionBaudrate = 3
- `baudrate`
  / *Condition*: required / *Type*: Baudrate<Baudrate> /
  Required baudrate value when `control_type` is either `verifyBaudrateTransitionWithFixedBaudrate` (1) or `verifyBaudrateTransitionWithSpecificBaudrate` (2).

Returns:

- `response`
  / *Type*: Response /
  The response from the LinkControl service request.

4.3.17 Method: read_data_by_identifier

Requests a value associated with a data identifier (DID) through the ReadDataByIdentifier service.

Arguments:

- `data_id_list`
  / *Type*: int | list[int] /
  The list of DIDs to be read.

Returns:

- `response`
  / *Type*: Response /
  The response from the ReadDataByIdentifier service request.

4.3.18 Method: read_dtc_information

Performs a ReadDiagnosticInformation service request.

Arguments:

- `subfunction`
  / *Condition*: required / *Type*: int /
  The subfunction for the ReadDiagnosticInformation service.
- `status_mask`
  / *Condition*: optional / *Type*: int /
  Status mask to filter the diagnostic information.
- `severity_mask`
  / *Condition*: optional / *Type*: int /
  Severity mask to filter the diagnostic information.

- `dtc`
  / *Condition*: optional / *Type*: int | Dtc /
  The Diagnostic Trouble Code to query. Can be an integer or a Dtc object.
- `snapshot_record_number`
  / *Condition*: optional / *Type*: int /
  Snapshot record number to specify the snapshot to read.
- `extended_data_record_number`
  / *Condition*: optional / *Type*: int /
  Extended data record number to specify the extended data to read.
- `extended_data_size`
  / *Condition*: optional / *Type*: int /
  Size of the extended data to read.
- `memory_selection`
  / *Condition*: optional / *Type*: int /
  Memory selection to specify the memory to be accessed.

Returns:

- `response`
  / *Type*: Response /
  The response from the ReadDiagnosticInformation service request.

4.3.19 Method: read_memory_by_address

Reads a block of memory from the server by sending a ReadMemoryByAddress service request.

Arguments:

- `memory_location`
  / *Condition*: required / *Type*: MemoryLocation<MemoryLocation> /
  The address and the size of the memory block to read.

Returns:

- `response`
  / *Type*: Response /
  The response from the ReadMemoryByAddress service request.

4.3.20 Method: request_download

Informs the server that the client wants to initiate a download from the client to the server by sending a RequestDownload service request.

Arguments:

- `memory_location`
  / *Condition*: required / *Type*: MemoryLocation<MemoryLocation> /
  The address and size of the memory block to be written.
- `dfi`
  / *Condition*: optional / *Type*: DataFormatIdentifier<DataFormatIdentifier> /
  Optional defining the compression and encryption scheme of the data. If not specified, the default value of 00 will be used, specifying no encryption and no compression.

Returns:

- `response`
  / *Type*: Response /
  The response from the RequestDownload service request.

4.3.21 Method: request_transfer_exit

Informs the server that the client wants to stop the data transfer by sending a RequestTransferExit service request.

Arguments:

- `data`
  / *Condition*: optional / *Type*: bytes /
  Optional additional data to send to the server.

Returns:

- `response`
  / *Type*: Response /
  The response from the RequestTransferExit service request.

4.3.22 Method: request_upload

Informs the server that the client wants to initiate an upload from the server to the client by sending a RequestUpload service request.

Arguments:

- `memory_location`
  / *Condition*: required / *Type*: MemoryLocation<MemoryLocation> /
  The address and size of the memory block to be written.
- `dfi`
  / *Condition*: optional / *Type*: DataFormatIdentifier<DataFormatIdentifier> /
  Optional defining the compression and encryption scheme of the data. If not specified, the default value of 00 will be used, specifying no encryption and no compression.

Returns:

- `response`
  / *Type*: Response /
  The response from the RequestUpload service request.

4.3.23 Method: routine_control

Sends a generic request for the RoutineControl service.

Arguments:

- `routine_id`
  / *Condition*: required / *Type*: int /
  The 16-bit numerical ID of the routine.
- `control_type`
  / *Condition*: required / *Type*: int /
  The service subfunction. Valid values are:

  - startRoutine = 1
  - stopRoutine = 2
  - requestRoutineResults = 3

- `data`
  / *Condition*: optional / *Type*: bytes /
  Optional additional data to give to the server.

Returns:

- `response`
  / *Type*: Response /
  The response from the RoutineControl service request.

4.3.24 Method: security_access

Successively calls request_seed and send_key to unlock a security level with the SecurityAccess service. The key computation is done by calling config['security_algo'].

Arguments:

- `level`
  / *Condition*: required / *Type*: int /
  The level to unlock. Can be the odd or even variant of it.
- `seed_params`
  / *Condition*: optional / *Type*: bytes /
  Optional data to attach to the RequestSeed request (securityAccessDataRecord).

Returns:

- `response`
  / *Type*: Response /
  The response from the SecurityAccess service request.

4.3.25 Method: tester_present

Sends a TesterPresent request to keep the session active.

Arguments:

- No specific arguments for this method.

Returns:

- `response`
  / *Type*: Response /
  The response from the TesterPresent request.

4.3.26 Method: transfer_data

Transfers a block of data to/from the client to/from the server by sending a TransferData service request and returning the server response.

Arguments:

- `sequence_number`
  / *Condition*: required / *Type*: int /
  Corresponds to an 8-bit counter that should increment for each new block transferred. Allowed values are from 0 to 0xFF.
- `data`
  / *Condition*: optional / *Type*: bytes /
  Optional additional data to send to the server.

Returns:

- `response`
  / *Type*: Response /
  The response from the TransferData service request.

4.3.27 Method: write_data_by_identifier

Requests to write a value associated with a data identifier (DID) through the WriteDataByIdentifier service.

Arguments:

- `did`
  / *Condition*: required / *Type*: int /
  The DID to write its value.
- `value`
  / *Condition*: required / *Type*: dict /
  Value given to the DidCodec.encode method. The payload returned by the codec will be sent to the server.

Returns:

- `response`
  / *Type*: Response /
  The response from the WriteDataByIdentifier service request.

4.3.28 Method: write_memory_by_address

Writes a block of memory in the server by sending a WriteMemoryByAddress service request.

Arguments:

- `memory_location`
  / *Condition*: required / *Type*: MemoryLocation<MemoryLocation> /
  The address and the size of the memory block to write.
- `data`
  / *Condition*: required / *Type*: bytes /
  The data to write into memory.

Returns:

- `response`
  / *Type*: Response /
  The response from the WriteMemoryByAddress service request.

4.3.29 Method: request_file_transfer

Sends a RequestFileTransfer request **Arguments:**

- `moop`
  / *Condition*: required / *Type*: int /
  Mode of operation:
  - AddFile = 1
  - DeleteFile = 2
  - ReplaceFile = 3
  - ReadFile = 4
  - ReadDir = 5
  - ResumeFile = 6
- `path`
  / *Condition*: required / *Type*: str /
  The path of the file or directory.
- `dfi`
  / *Condition*: optional / *Type*: DataFormatIdentifier /
  DataFormatIdentifier defining the compression and encryption scheme of the data. Defaults to no compression and no encryption. Use for:
  - AddFile = 1
  - ReplaceFile = 3
  - ReadFile = 4
  - ResumeFile = 6
- `filesize`
  / *Condition*: optional / *Type*: int | Filesize /
  The filesize of the file to write. If Filesize, uncompressed and compressed sizes will be encoded as needed. Use for:
  - AddFile = 1
  - ReplaceFile = 3
  - ResumeFile = 6

Returns:

- `response`
  / *Type*: Response /
  The response from the file operation.

4.3.30 Method: authentication

Sends an Authentication request introduced in 2020 version of ISO-14229-1. **Arguments:**

- `authentication_task`
  / *Condition*: required / *Type*: int /
  The authentication task (subfunction) to use:
  - deAuthenticate = 0
  - verifyCertificateUnidirectional = 1
  - verifyCertificateBidirectional = 2

  - proofOfOwnership = 3
  - transmitCertificate = 4
  - requestChallengeForAuthentication = 5
  - verifyProofOfOwnershipUnidirectional = 6
  - verifyProofOfOwnershipBidirectional = 7
  - authenticationConfiguration = 8

- `communication_configuration`

  / *Condition*: optional / *Type*: int /

  Configuration about security in future diagnostic communication (vehicle manufacturer specific). Allowed values are from 0 to 255.

- `certificate_client`

  / *Condition*: optional / *Type*: bytes /

  The certificate to verify.

- `challenge_client`

  / *Condition*: optional / *Type*: bytes /

  Client challenge containing vehicle manufacturer-specific data or a random number.

- `algorithm_indicator`

  / *Condition*: optional / *Type*: bytes /

  Algorithm used in Proof of Ownership (POWN). This is a 16-byte value containing the BER-encoded OID of the algorithm.

- `certificate_evaluation_id`

  / *Condition*: optional / *Type*: int /

  Unique ID for evaluating the transmitted certificate. Allowed values are from 0 to 0xFFFF.

- `certificate_data`

  / *Condition*: optional / *Type*: bytes /

  Certificate data for verification.

- `proof_of_ownership_client`

  / *Condition*: optional / *Type*: bytes /

  Proof of Ownership of the challenge to be verified by the server.

- `ephemeral_public_key_client`

  / *Condition*: optional / *Type*: bytes /

  Client's ephemeral public key for Diffie-Hellman key agreement.

- `additional_parameter`

  / *Condition*: optional / *Type*: bytes /

  Additional parameter provided if required by the server.

Returns:

- `response`

  / *Type*: Response /

  The server's response to the authentication request.

4.3.31 Method: routine_control_by_name

Sends a request for the RoutineControl service by routine name.

Arguments:

- param `routine_name` (required): Name of the routine
  - type `routine_name`: str
- param `data` (optional): Optional additional data to give to the server
  - type `data`: bytes

Returns:

- `response` / *Type*: Response / The server's response to the RoutineControl request.

4.3.32 Method: read_data_by_name

Get diagnostic service list by a list of service names.

Arguments:

- param `service_name_list`: List of service names
  - type `service_name_list`: list[str]
- param `parameters`: Parameter list
  - type `parameters`: list[]

Returns:

- `response` / *Type*: Response / The server's response containing the diagnostic service list.

4.3.33 Method: get_encoded_request_message

Get diagnostic service encoded request (bytes value).

Arguments:

- param `service_name`: Diagnostic service's name
  - type `service_name`: string
- param `parameters_dict`: Parameter dictionary
  - type `parameters_dict`: dict

Returns:

- `encoded_message` / *Type*: bytes / The encoded message in bytes value.

4.3.34 Method: get_decoded_positive_response_message

Get diagnostic service decoded positive response message.

Arguments:

- param `service_name`: Diagnostic service's name
  - type `service_name`: string

- param `response_data`: Bytes data from the response
  - type `parameters_dict`: bytes
- param `device_name`: Name of the device
  - type `device_name`: string

Returns:

- `decode_message` / *Type*: dict / The decode message in dictionary.

4.3.35 Method: write_data_by_name

Requests to write a value associated with a name of service through the WriteDataByName service.

Arguments:

- `did`
  / *Condition*: required / *Type*: int /
  The DID to write its value.
- `value`
  / *Condition*: required / *Type*: dict /
  Value given to the DidCodec.encode method. The payload returned by the codec will be sent to the server.

Returns:

- `response`
  / *Type*: Response /
  The response from the WriteDataByIdentifier service request.

Chapter 5

__init__.py

5.1 Class: RobotFramework_UDS

Imported by:

```
from RobotFramework_UDS.__init__ import RobotFramework_UDS
```

RobotFramework_UDS is a Robot Framework library aimed to provide UDP client to handle request/response.

Chapter 6

Appendix

About this package:

Table 6.1: Package setup

Setup parameter	Value
Name	RobotFramework_UDS
Version	0.1.5
Date	10.10.2024
Description	Robot Framework keywords for UDS (Unified Diagnostic Services) communication
Package URL	robotframework-uds
Author	Mai Minh Tri
Email	tri.maiMinh@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 7

History

0.1.0	09/2024
Initial version	
0.1.4	09/2024
- Update `Write Data By Name` *keyword to return response with given service name instead of did.*	
0.1.5	10/2024
- Update `Read Data By Name` *keyword to return response with given service name instead of did.* *- Update* `Get Encoded Request Message` *keyword to support convert string to proper data type for request parameters.* *- Update* `Get Decoded Response Message` *keyword to support decoded response data.*	

RobotFramework_UDS.pdf

Created at 11.10.2024 - 11:41:38

by GenPackageDoc v. 0.41.1