RobotFramework_UDS

v. 0.1.2

Mai Minh Tri

05.09.2024

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.

2.6 Examples

```
*** Settings ***
Library           RobotFramework_TestsuitesManagement    WITH NAME    testsuites
Library           RobotFramework_UDS
Suite Setup       Connect
Suite Teardown    Disconnect

*** Variables ***
${SUT_IP_ADDRESS}=         ecu_ip_address
${SUT_LOGICAL_ADDRESS}=    ecu_logical_address
${TB_IP_ADDRESS}=          client_ip_address
${TB_LOGICAL_ADDRESS}=     client_logical_address
${ACTIVATION_TYPE}=        0

${FILE}=       path/file.pdx
${VARIANT}=    variant

${NAME}=    UDS Connector

*** Keywords ***
Connect
    Log    Create a uds Connector
    ${uds}=    Create UDS Connector    ecu_ip_address= ${SUT_IP_ADDRESS}
    ...                                ecu_logical_address= ${SUT_LOGICAL_ADDRESS}
    ...                                client_ip_address= ${TB_IP_ADDRESS}
    ...                                client_logical_address= ${TB_LOGICAL_ADDRESS}
    ...                                activation_type= ${ACTIVATION_TYPE}
    Log    Using UDS Connector
    Connect UDS Connector    name=${Name}
    Log    Open uds connection
    Open UDS Connection
    Using pdx

Disconnect
    Log    Close uds connection
    Close UDS Connection

Using pdx
    Load PDX    ${FILE}    ${VARIANT}

*** Test Cases ***
Test user can use Tester Present service on ECU
    Log    Use Tester Present service
    ${response}=    Tester Present

Test user can use ECU Reset service on ECU
    Log    Use ECU Reset service

    Log    Hard reset
    ${response_hr}=    ECU Reset    1

    Log    Key off on reset
    ${response_k}=     ECU Reset    2

    Log    Soft reset
    ${response_sr}=    ECU Reset    3

    Log    Enable rapid power shut down
    ${response_e}=     ECU Reset    4

    Log    disable rapid power shut down
    ${response_d}=     ECU Reset    5

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

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

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

    Log    PingTest_Start_NoResponse
    Routine Control By Name    PingTest_Start_NoResponse

Test user can use Diagnostic Session Control service on ECU
    Log    Diagnostic Session Control service

    Diagnostic Session Control    1
```

Chapter 3

DiagnosticServices.py

3.1 Class: DiagnosticServices

Imported by:

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

3.1.1 Method: get_data_by_name

3.1.2 Method: get_encoded_request_message

3.1.3 Method: get_did_codec

3.2 Class: PDXCodec

Imported by:

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

3.2.1 Method: decode

Chapter 4

UDSKeywords.py

4.1 Class: UDSKeywords

Imported by:

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

4.1.1 Method: connect_uds_connector

4.1.2 Method: create_uds_connector

Description: Create a connection to establish

Parameters:

- param `name`: Name of connection
  - doip: Establish a doip connection to an (ECU)
- type `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

4.1.3 Method: load_pdx

Description: Load PDX

Parameters:
- param `pdx_file`: pdx file path
- type `pdx_file`: st
- param `variant`:
- type `variant`: str

4.1.4 Method: build_payload

4.1.5 Method: validate_content_response

Description: Validates the content of a UDS response

Parameters:
- param ‘response‘: The UDS response object to validate.
- type ‘response‘: udsoncan.Response<Response>
- param ‘expected_data‘: The expected data (optional) to be matched within the response.
- type ‘expected_data‘: byte

4.1.6 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

4.1.7 Method: set_config

Description: Set UDS config Using create_configure to create a new config for UDS connector. If not, the default config will be use.

4.1.8 Method: connect

Description: Open uds connection

4.1.9 Method: disconnect

Description: Close uds connection

4.1.10 Method: access_timing_parameter

Description: Sends a generic request for AccessTimingParameter service

Parameters:
- param `access_type` (required): The service subfunction
  - readExtendedTimingParameterSet = 1
  - setTimingParametersToDefaultValues = 2
  - readCurrentlyActiveTimingParameters = 3
  - setTimingParametersToGivenValues = 4
- type `access_type` in
- param `timing_param_record` (optional): The parameters data. Specific to each ECU.
- type `timing_param_record` bytes

4.1.11 Method: clear_dianostic_infomation

Description: Requests the server to clear its active Diagnostic Trouble Codes.

Parameters:
- param `group`: 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 value

  - `0x000000` : Emissions-related systems
  - `0xFFFFFF` : All DTCs

- type `group`: in
- **param memory_selection: MemorySelection byte (0-0xFF). This value is user defined and introduced**
  Only added to the request payload when different from None. Default : None
- type `memory_selection`: int

4.1.12 Method: communication_control

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

Parameter:
- param `control_type` (required): The action to request such as enabling or disabling some messages. This value can also be ECU manufacturer-specifi
  - enableRxAndTx = 0
  - enableRxAndDisableTx = 1
  - disableRxAndEnableTx = 2
  - disableRxAndTx = 3
  - enableRxAndDisableTxWithEnhancedAddressInformation = 4
  - enableRxAndTxWithEnhancedAddressInformation = 5
- type `control_type`: int
- param `communication_type` (required): 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 a bytes is given, the value will be decoded to create the required CommunicationType<CommunicationType object
- type communication_type: CommunicationType<CommunicationType>, bytes, int.
- param `node_id` (optional):

DTC memory identifier (nodeIdentificationNumber). This value is user defined and introduced in 2013 version of ISO-14229-1. Possible only when control type is `enableRxAndDisableTxWithEnhancedAddressInformation` or `enableRxAndTxWithEnhancedAddressInformation` Only added to the request payload when different from None.

Default : None

- type `node_id`: int

4.1.13 Method: control_dtc_setting

Description: 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.

Paramters:
- param `setting_type` (required): Allowed values are from 0 to 0x7F.
  - on = 1
  - off = 2
  - vehicleManufacturerSpecific = (0x40, 0x5F) # To be able to print textual name for logging only.
  - systemSupplierSpecific = (0x60, 0x7E) # To be able to print textual name for logging only
- type `setting_type`: in
- param `data` (optional): Optional additional data sent with the request called DTCSettingControlOptionRecord
- type `data`: bytes

4.1.14 Method: diagnostic_session_control

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

Parameters:
- param `newsession` (required): The session to try to switch.
  - defaultSession = 1
  - programmingSession = 2
  - extendedDiagnosticSession = 3
  - safetySystemDiagnosticSession = 4
- type `newsession`: int

4.1.15 Method: dynamically_define_did

Description: Defines a dynamically defined DID.

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

4.1.16 Method: ecu_reset

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

- **param `reset_type` (required): The type of reset to perform.** – hardReset = 1
  - keyOffOnReset = 2
  - softReset = 3
  - enableRapidPowerShutDown = 4
  - disableRapidPowerShutDown = 5
- type reset_type: int

4.1.17 Method: io_control

Description: Substitutes the value of an input signal or overrides the state of an output by sending a InputOutput-ControlByIdentifier service request.

Parameters:
- param `did` (required): Data identifier to represent the IO
- type `did`: int
- param `control_param` (optional):
  - returnControlToECU = 0
  - resetToDefault = 1
  - freezeCurrentState = 2
  - shortTermAdjustment = 3
- type `control_param`: int
- **param `values` (optional): Optional values to send to the server. This parameter will be given to Dic**
  A list for positional arguments
  - A dict for named arguments
  - An instance of IOValues<IOValues> for mixed arguments
- type `values`: list, dict, IOValues<IOValues>
- **param masks: Optional mask record for composite values. The mask definition must be included in**
  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.
- type masks: list, dict, IOMask<IOMask>, bool

4.1.18 Method: link_control

Description: Controls the communication baudrate by sending a LinkControl service request.

Parameters:

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

4.1.19 Method: read_data_by_identifier

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

Parameters: See `an example<reading_a_did>` about how to read a DID

- param data_id_list: The list of DID to be read
- type data_id_list: int | list[int]

4.1.20 Method: read_dtc_information

Update later robotframework-uds-udskeywords-udskeywords-read-memory-by-address --

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

Parameters:

- param `memory_location` (required): The address and the size of the memory block to read.
- type `memory_location`: MemoryLocation <MemoryLocation>

4.1.21 Method: request_download

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

Parameters:

- param `memory_location` (required): The address and size of the memory block to be written.
- type `memory_location`: MemoryLocation <MemoryLocation>
- param `dfi` (optional):

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

- type dfi: DataFormatIdentifier <DataFormatIdentifier>

4.1.22 Method: request_transfer_exit

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

Parameters:

- param `data` (optional): Optional additional data to send to the server
- type `data`: bytes

4.1.23 Method: request_upload

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

Parameters:

- param `memory_location` (required): The address and size of the memory block to be written.
- type `memory_location`: MemoryLocation <MemoryLocation>
- param dfi (optional): 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
- type dfi: DataFormatIdentifier

4.1.24 Method: routine_control

Description: Sends a generic request for the RoutineControl service

Parameters:

- param `routine_id` (required): The 16-bit numerical ID of the routine
- type `routine_id` int
- param `control_type` (required): The service subfunction
- type `control_type` int
- valid `control_type`
  - startRoutine = 1
  - stopRoutine = 2
  - requestRoutineResults = 3
    - param `data` (optional): Optional additional data to give to the server
    - type `data` bytes

4.1.25 Method: security_access

Description: 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']

Parameters:

- param `level` (required): The level to unlock. Can be the odd or even variant of it.
- type `level`: int
- param `seed_params` (optional): Optional data to attach to the RequestSeed request (securityAccess-DataRecord).
- type `seed_params`: bytes

4.1.26 Method: tester_present

Description: Sends a TesterPresent request to keep the session active.

4.1.27 Method: transfer_data

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

Parameters:

- **param `sequence_number` (required): Corresponds to an 8bit counter that should increment f**
  Allowed values are from 0 to 0xFF
- type `sequence_number`: int
- param `data` (optional): Optional additional data to send to the server
- type `data`: bytes

4.1.28 Method: write_data_by_identifier

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

Parameters:

- param did: The DID to write its value
- type did: int
- param value: Value given to the DidCodec.encode method. The payload returned by the codec will be sent to the server.
- type value: int

4.1.29 Method: write_memory_by_address

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

Parameters:

- param `memory_location` (required): The address and the size of the memory block to read.
- type `memory_location`: MemoryLocation <MemoryLocation>
- param `data` (required): The data to write into memory.
- type `data`: bytes

4.1.30 Method: request_file_transfer

Parameters:

- param `moop` (required): Mode operate
  - AddFile = 1
  - DeleteFile = 2
  - ReplaceFile = 3
  - ReadFile = 4
  - ReadDir = 5
  - ResumeFile = 6
- type `moop`: int
- param `path` (required):
- type `path`: str
- **param `dfi`: DataFormatIdentifier 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. Use for `moop`:
  - AddFile = 1
  - ReplaceFile = 3
  - ReadFile = 4
  - ResumeFile = 6
- type `dfi`: DataFormatIdentifier
- param `filesize` (optional): The filesize of the file to write.

  If filesize is an object of type Filesize, the uncompressed size and compressed size will be encoded on the minimum amount of bytes necessary, unless a `width` is explicitly defined. If no compressed size is given or filesize is an `int`, then the compressed size will be set equal to the uncompressed size or the integer value given as specified by ISO-14229

  Use for `moop`:

  - AddFile = 1
  - ReplaceFile = 3
  - ResumeFile = 6
- type `filesize`: int | Filesize

4.1.31 Method: authentication

Description: Sends an Authentication request introduced in 2020 version of ISO-14229-1. You can also use the helper functions to send each authentication task (sub function).

Parameters:

- param `authentication_task` (required): The authenticationTask (subfunction) to use.
  - deAuthenticate = 0
  - verifyCertificateUnidirectional = 1
  - verifyCertificateBidirectional = 2
  - proofOfOwnership = 3
  - transmitCertificate = 4
  - requestChallengeForAuthentication = 5
  - verifyProofOfOwnershipUnidirectional = 6
  - verifyProofOfOwnershipBidirectional = 7
  - authenticationConfiguration = 8
- type `authentication_task`: int
- param `communication_configuration` (optional): Optional Configuration information about how to proceed with security in further diagnostic communication after the Authentication (vehicle manufacturer specific).

Allowed values are from 0 to 255.

- type `communication_configuration`: int
- param `certificate_client` (optional): Optional The Certificate to verify.
- type `certificate_client`: bytes
- param `challenge_client` (optional): Optional The challenge contains vehicle manufacturer specific formatted client data (likely containing randomized information) or is a random number.
- type `challenge_client`: bytes
- param `algorithm_indicator` (optional):

Optional Indicates the algorithm used in the generating and verifying Proof of Ownership (POWN), which further determines the parameters used in the algorithm and possibly the session key creation mode. This field is a 16 byte value containing the BER encoded OID value of the algorithm used. The value is left aligned and right padded with zero up to 16 bytes.

- type `algorithm_indicator`: bytes
- param `certificate_evaluation_id`: Optional unique ID to identify the evaluation type of the transmitted certificate.

The value of this parameter is vehicle manufacturer specific. Subsequent diagnostic requests with the same evaluationTypeId will overwrite the certificate data of the previous requests. Allowed values are from 0 to 0xFFFF.

- type certificate_evaluation_id: int
- param `certificate_data` (optional): Optional The Certificate to verify.
- type `certificate_data`: bytes
- param `proof_of_ownership_client` (optional): Optional Proof of Ownership of the previous given challenge to be verified by the server.
- type `proof_of_ownership_client`: bytes
- param `ephemeral_public_key_client` (optional): Optional Ephemeral public key generated by the client for Diffie-Hellman key agreement.
- type `ephemeral_public_key_client`: bytes
- param `additional_parameter` (optional): Optional additional parameter is provided to the server if the server indicates as neededAdditionalParameter.
- type `additional_parameter`: bytes

4.1.32 Method: routine_control_by_name

Description: Sends a request for the RoutineControl service by routine name

Parameters:
- param `routine_name` (required): Name of routine
- type `routine_name`: str
- param `control_type` (required): The service subfunction
- type `control_type` int

Valid `control_type`

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

- param `data` (optional): Optional additional data to give to the server
- type `data` bytes

4.1.33 Method: read_data_by_name

Description: Get diagnostic service list by list of service name

Parameters:
- param `service_name_list`: list of service name
- type `service_name_list`: list[str]
- param `parameters`: parameter list
- type `parameters`: list[]

4.1.34 Method: get_encoded_request_message

Description: Get diagnostic service encoded request list (hex value)

Parameters:
- param `diag_service_list`: Diagnostic service list
- type `diag_service_list`: []
- param `parameters`: parameter list
- type `parameters`: list[]

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	0.1.2
Date	05.09.2024
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	

RobotFramework_UDS.pdf

Created at 05.09.2024 - 06:24:04

by GenPackageDoc v. 0.41.1