RobotFramework_UDS

v. 0.1.12

Mai Minh Tri

10.02.2024

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.

The **RobotFramework_UDS** provides a versatile "Send UDS Request By Name" keyword that can send and receive UDS messages without requiring the command type specification (e.g., Read, Write, RoutineControl). This allows users to send requests without needing to know the specific UDS command type associated with a service, as the keyword automatically detects and handles it.

Examples

Send UDS Request By Name

```
Test user can send UDS request without needing to specify the command type
    Log    Test user can send UDS request without needing to specify the command type

    Log    readCPUClockFrequencies_Read
    ${service_name_list}=    Create List    readCPUClockFrequencies_Read
    ${responses}=    Send UDS Request 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

    Log    CAM1PowerSupply_Set
    ${param_dict_input_output_control}=    Create Dictionary    mode=passiv
    ${response}=    Send UDS Request By Name    CAM1PowerSupply_Set    ↩
    ↪ ${param_dict_input_output_control}

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

    Log    RealTimeClock_Write
    ${param_dict_write_data_by_name}=    Create Dictionary    Day=26    Month=September    ↩
    ↪    Year=2024    Hour=10    Second=45    Minute=0
    ${res}=    Send UDS Request By Name    RealTimeClock_Write    ↩
    ↪ ${param_dict_write_data_by_name}
    Log    ${res}    console=True

    Log    Routine Control By Name service: StartIperfServer_Start

    ${param_dict_routine_control_by_name}=    Create Dictionary    port=5101    ↩
    ↪ argument=-i 0.5 -B 192.168.1.
    ${response}=    Send UDS Request By Name    StartIperfServer_Start    ↩
    ↪ ${param_dict_routine_control_by_name}

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

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
```

Input Output Control By Name

```
Test user can use Input Output Control By Name service on ECU
    Log    Input Output Control By Name service: CAM1PowerSupply_Set

    ${param_dict}=    Create Dictionary    mode=passiv
    ${response}=    Input Output Control By Name    CAM1PowerSupply_Set    ${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
```

Send UDS Request By Name

```
Test user can send UDS request without needing to specify the command type
    Log    Test user can send UDS request without needing to specify the command type

    Log    readCPUClockFrequencies_Read
    ${service_name_list}=    Create List    readCPUClockFrequencies_Read
    ${responses}=    Send UDS Request 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

    Log    CAM1PowerSupply_Set
    ${param_dict_input_output_control}=    Create Dictionary    mode=passiv
    ${response}=    Send UDS Request By Name    CAM1PowerSupply_Set    ↩
    ↪ ${param_dict_input_output_control}

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

    Log    RealTimeClock_Write
    ${param_dict_write_data_by_name}=    Create Dictionary    Day=26    Month=September    ↩
    ↪    Year=2024    Hour=10    Second=45    Minute=0
    ${res}=    Send UDS Request By Name    RealTimeClock_Write    ↩
    ↪ ${param_dict_write_data_by_name}
    Log    ${res}    console=True

    Log    Routine Control By Name service: StartIperfServer_Start

    ${param_dict_routine_control_by_name}=    Create Dictionary    port=5101    ↩
    ↪ argument=-i 0.5 -B 192.168.1.
    ${response}=    Send UDS Request By Name    StartIperfServer_Start    ↩
    ↪ ${param_dict_routine_control_by_name}

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

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

Arguments:

- `odx_param`
  / *Condition*: required / *Type*: object /
  The ODX parameters.
- `req_sub_param`
  / *Condition*: required / *Type*: dict /
  The dictionary of request parameter.

Returns:

- `req_sub_param`
  / *Type*: dict /
  The dictionary of request parameters with the correct data types.

3.1.2 Method: convert_request_data_type

Convert given request parameters (dictionary) to correct data type

Arguments:

- `service`
  / *Condition*: required / *Type*: object /
  The diagnostic service.
- `parameter_dict`
  / *Condition*: required / *Type*: dict /
  The dictionary of request parameter.

Returns:

- `parameter_dict`
  / *Type*: dict /
  The dictionary of request parameters with the correct data types.

3.1.3 Method: get_diag_service_by_name

Retrieve the list of diagnostic services from a PDX file using a specified list of service names.

Arguments:

- `service_name_list`
  / *Condition*: required / *Type*: list /
  The list of service names

Returns:

- `diag_service_list`
  / *Type*: list /
  The list of diagnostic services from a PDX file.

3.1.4 Method: get_encoded_request_message

Retrieve the encode request message from parameters dictionary.

Arguments:

- `service_name`
  / *Condition*: required / *Type*: str /
  The service's names
- `parameter_dict`
  / *Condition*: required / *Type*: dict /
  The dictionary of request parameter.

Returns:

- `encode_message`
  / *Type*: bytes /
  The encoded message.

3.1.5 Method: get_decode_response_message

Retrieve the encode request message from parameters dictionary.

Arguments:

- `service_name`
  / *Condition*: required / *Type*: str /
  The service's names
- `raw_message`
  / *Condition*: required / *Type*: bytes /
  The raw message from the response.

Returns:

- `decode_message`
  / *Type*: bytes /
  The decoded message.

3.1.6 Method: get_full_positive_response_data

Retrieve the complete byte data from the response, as the UDS removes the service ID.

Arguments:

- `service_name`
  / *Condition*: required / *Type*: str /
  The service's names
- `data`
  / *Condition*: required / *Type*: bytes /
  The raw message from the response.

Returns:

- `positive_response_data`
  / *Type*: bytes /
  The complete byte data from the response.

3.1.7 Method: get_did_codec

Retrieves a dictionary of DID codecs for a given diagnostic service ID.

Arguments:

- `service_id`
  / *Condition*: required / *Type*: int /
  The service's did

Returns:

- `did_codec`
  / *Type*: dict /
  A dictionary where the keys are DIDs

3.2 Class: PDXCodec

Imported by:

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

3.2.1 Method: decode

3.2.2 Method: encode

3.3 Class: ServiceID

Imported by:

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

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

Connects a UDS connector for the specified device.
Arguments:

- `device_name`
  / *Condition*: optional / *Type*: str / *Default*: "default" /
  Name of the device to connect to. If the device does not exist, a ValueError will be raised.
- `config`
  / *Condition*: optional / *Type*: dict / *Default*: default_client_config /
  Configuration settings for the UDS client, applied if the device is not already available.
- `close_connection`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Indicates whether to close the connection automatically when done.

Raises:

- `ValueError`

  Raised if the specified device does not exist, suggesting the use of "Create UDS Connector" to create a new device.

Returns:

- `None`

  No return value. The function initializes or updates the UDS connector for the specified device if not already available.

4.3.2 Method: create_uds_connector

Establishes a connection with an ECU.

Arguments:

- `communication_name`

  / *Type*: str / *Condition*: required /

  Specifies the type of communication to establish.

- `ecu_ip_address`

  / *Type*: str / *Condition*: required /

  The IP address of the ECU for establishing the connection. Should be a valid IPv4 (e.g., "192.168.1.1") or IPv6 address (e.g., "2001:db8::").

- `ecu_logical_address`

  / *Type*: any / *Condition*: required /

  The logical address of the ECU.

- `tcp_port`

  / *Type*: int / *Condition*: optional / *Default*: **TCP_DATA_UNSECURED** /

  TCP port used for unsecured data communication.

- `udp_port`

  / *Type*: int / *Condition*: optional / *Default*: **UDP_DISCOVERY** /

  UDP port used for ECU discovery.

- `activation_type`

  / *Type*: RoutingActivationRequest.ActivationType / *Condition*: optional / *Default*: ActivationTypeDefault /

  Specifies the activation type, which can be the default (ActivationTypeDefault) or a value based on application-specific settings.

- `protocol_version`

  / *Type*: int / *Condition*: optional / *Default*: 0x02 /

  The version of the protocol used for the connection.

- `client_logical_address`

  / *Type*: int / *Condition*: optional / *Default*: None /

  The logical address this DoIP client will use to identify itself. Per specification, this should be within the range 0x0E00 to 0x0FFF.

- `client_ip_address`

  / *Type*: str / *Condition*: optional / *Default*: None /

  If specified, binds to this IP as the source for UDP and TCP communication. Can be an IPv4 or IPv6 address, matching the type of `ecu_ip_address`.

- `use_secure`

  / *Type*: Union[bool, ssl.SSLContext] / *Condition*: optional / *Default*: False /

  Enables TLS if set to True. Uses a default SSL context by default; can be set to a preconfigured SSL context for more control. If enabled, consider changing `tcp_port` to 3496.

- `auto_reconnect_tcp`

  / *Type*: bool / *Condition*: optional / *Default*: False /

  Enables automatic reconnection of TCP sockets if closed by the peer.

4.3.3 Method: load_pdx

Load PDX file

Arguments:

- `pdx_file`

  / *Type*: str /

  PDX file path

- `variant`

  / *Type*: str /

4.3.4 Method: create_config

Creates a configuration for the UDS connector.

Arguments:

- `exception_on_negative_response` : bool When set to True, raises a NegativeResponseException if the server responds with a negative response. If False, the Response's positive property will be set to False.
- `exception_on_invalid_response` : bool When set to True, raises an InvalidResponseException if interpret_response encounters an invalid response. If False, the Response's valid property will be set to False.
- `exception_on_unexpected_response` : bool When set to True, raises an UnexpectedResponseException if the server returns an unexpected response, such as an unmatched subfunction echo. If False, the Response's unexpected property will be set to True.
- `security_algo` : Callable[[int, bytes, Any], bytes] Security algorithm function for the SecurityAccess service. Signature: security_algo(level, seed, params) -> bytes
  - `level` : int — The requested security level.
  - `seed` : bytes — The seed provided by the server.
  - `params` : Any — Parameters provided by security_algo_params.
- `security_algo_params` : object or dict Parameters passed to the security algorithm specified in security_algo.
- `data_identifiers` : dict[int, Union[str, DidCodec]] A dictionary mapping data identifiers to a codec (string or DidCodec) for encoding/decoding values in services like ReadDataByIdentifier, WriteDataByIdentifier, etc.
- `input_output` : dict[int, Union[str, DidCodec, dict]] Dictionary mapping IO data identifiers to a codec for InputOutputControlByIdentifier service. Supports composite codecs with sub-dictionaries specifying bitmasks.
- `tolerate_zero_padding` : bool When True, ignores trailing zeros in response data to prevent InvalidResponseException if the protocol uses zero-padding.
- `ignore_all_zero_dtc` : bool For ReadDTCInformation service, skips DTCs with an ID of 0x000000, useful if the protocol uses zero-padding. See online documentation for further details.
- `server_address_format` : int Specifies the MemoryLocation address format to use when not explicitly provided.
- `server_memorysize_format` : int Specifies the MemoryLocation memory size format to use when not explicitly provided.

- `extended_data_size` : dict[int, int] Specifies DTC extended data record sizes. Example: ` { 0x123456: 45, # DTC 0x123456 has an extended data size of 45 bytes. 0x123457: 23 # DTC 0x123457 has an extended data size of 23 bytes. }‘
- `dtc_snapshot_did_size` : int Number of bytes for encoding data identifiers in ReadDTCInformation (default: 2).
- `standard_version` : int UDS standard version, valid values are 2006, 2013, or 2020 (default: 2020).
- `request_timeout` : float Maximum wait time (in seconds) for a response after sending a request. Defaults to 5 seconds. Set to None to wait indefinitely.
- `p2_timeout` : float Maximum wait time (in seconds) for a first response after sending a request, per ISO 14229-2:2013 (default: 1 second).
- `p2_star_timeout` : float Maximum wait time (in seconds) after receiving a requestCorrectlyReceived-ResponsePending (0x78) response from the server (default: 5 seconds).
- `use_server_timing` : bool When True, uses P2 and P2* timing values provided by the server for sessions with 2013 or later standards. Defaults to True.

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_diagnostic_information

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*: dict /
  The decoded response data.

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.

4.3.36 Method: io_control_by_name

Sends a request for the IOControl service by name of input output control service.

Arguments:

- `io_control_name`
  / *Condition*: required / *Type*: str /
  Name of the input output control service
- `value`
  / *Condition*: optional / *Type*: dict /
  Optional additional data to give to the server
- `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*: dict /
  The decoded response data.

4.3.37 Method: send_uds_request_by_name

Sends a UDS request by the name of the specified diagnostic service.

Arguments:

- `service_name`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Name of the diagnostic service to request.

- `device_name`

  / *Condition*: optional / *Type*: str / *Default*: "default" /

  Name of the device to which the UDS request will be sent.

- `kwargs`

  / *Condition*: optional / *Type*: dict /

  Additional parameters specific to certain services. Possible values include:

  - `reset_type`: (int) Reset type for ECU reset services.
  - `parameters`: (dict) Parameters for WRITE_DATA_BY_IDENTIFIER, INPUT_OUTPUT_CONTROL_BY_IDENTI and ROUTINE_CONTROL services.
  - `mask`: (any) Mask value for the INPUT_OUTPUT_CONTROL_BY_IDENTIFIER service.
  - `groups`: (int) Group identifiers for the CLEAR_DIAGNOSTIC_INFORMATION service (default: 0xFFFFFF).
  - `memory_selection`: (any) Memory selection for the CLEAR_DIAGNOSTIC_INFORMATION service.
  - `control_type`: (int) Control type for COMMUNICATION_CONTROL service.
  - `communication_type`: (int) Communication type for COMMUNICATION_CONTROL service.
  - `node_id`: (any) Node ID for COMMUNICATION_CONTROL service.
  - `setting_type`: (int) Setting type for CONTROL_DTC_SETTING service.
  - `data`: (any) Data for CONTROL_DTC_SETTING and TRANSFER_EXIT services.
  - `session_type`: (int) Session type for DIAGNOSTIC_SESSION_CONTROL service.
  - `subfunction`: (int) Subfunction for READ_DTC_INFORMATION service.
  - `status_mask`: (any) Status mask for READ_DTC_INFORMATION service.
  - `severity_mask`: (any) Severity mask for READ_DTC_INFORMATION service.
  - `dtc`: (any) Diagnostic Trouble Code (DTC) for READ_DTC_INFORMATION service.
  - `snapshot_record_number`: (any) Snapshot record number for READ_DTC_INFORMATION service.
  - `extended_data_record_number`: (any) Extended data record number for READ_DTC_INFORMATION service.
  - `extended_data_size`: (any) Extended data size for READ_DTC_INFORMATION service.
  - `memory_location`: (any) Memory location for READ_MEMORY_BY_ADDRESS, REQUEST_DOWNLOAD, REQUEST_UPLOAD, and WRITE_MEMORY_BY_ADDRESS services.
  - `dfi`: (any) Data Format Identifier (DFI) for REQUEST_DOWNLOAD and REQUEST_UPLOAD services.
  - `level`: (any) Security level for SECURITY_ACCESS service.
  - `seed_params`: (any) Seed parameters for SECURITY_ACCESS service.
  - `sequence_number`: (int) Sequence number for TRANSFER_DATA service.
  - `data`: (any) Data for TRANSFER_DATA and TRANSFER_EXIT services.

Returns:

- `response`

  / *Type*: Response /

  The decoded response data from the 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	0.1.12
Date	10.02.2024
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.*	
0.1.6	10/2024
Temporary solution for encoding byte field param with dynamic length	
0.1.7	10/2024
Improve the encoding function for byte data	
0.1.8	10/2024
- Remove temporary solution for encoding byte field param with dynamic length *- Update package requirements to use* `odxtools` *version greater than* `8.2.1`	
0.1.9	10/2024
Add `Input Output Control By Name` *keyword*	
0.1.10	10/2024
Add `Send UDS Request By Name` *keyword*	
0.1.11	02/2025
Fix the type error that occurs when user utilize string parameters in the `Create UDS Connector` *keyword*	
0.1.12	02/2025
Update `Read/Write data by identifier` *keywords can be used without the need for PDX file*	

RobotFramework_UDS.pdf

Created at 12.02.2025 - 14:31:18

by GenPackageDoc v. 0.16.0