UDS

Release 0.1.1

Maciej Dąbrowski

Oct 02, 2021

CONTENTS

CHAPTER

ONE

INSTALLATION

To install the package, run the following command in your command line interface:

```
pip install py-uds
```

If you have already installed the package, you can update it using the following command:

```
pip install -U py-uds
```

UDS package is distributed via PyPI. You can visit distribution page of UDS package using the following hyperlink: https://pypi.org/project/py-uds/.

CHAPTER

TWO

DIAGNOSTIC MESSAGES

Implementation related to diagnostic messages and packets is located in `uds.messages` sub-package.

2.1 UDS Message Implementation

Diagnostic messages **implementation is divided into two parts:**

- *UDS Message* - storage for a temporary diagnostic message definition on the user side
- *UDS Message Record* - storage for historic information of a diagnostic message that was either received or transmitted

2.1.1 UDS Message

`UdsMessage` class is meant to provide containers for *diagnostic messages* information. Once a diagnostic message object is created, it stores diagnostic message data that were provided by a user. One can **use these objects to execute complex operations** (provided in other subpackages) such as diagnostic messages transmission or *segmentation*.

All `UdsMessage` **attributes are validated on each value change**, therefore a user will face an exception if one tries to set an invalid (incompatible with the annotation) value to of these attributes.

Attributes implemented in `UdsMessage` **class:**

- `payload` - settable
- `addressing` - settable

Example code:

```
from uds.messages import UdsMessage, AddressingType

# example how to create an object
uds_message = UdsMessage(payload=[0x10, 0x03],
                         addressing=AddressingType.PHYSICAL)

# raw message attribute
print(uds_message.payload)
uds_message.payload = (0x62, 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF)
print(uds_message.payload)
uds_message.payload = [0x3E, 0x80]
print(uds_message.payload)
```

(continues on next page)

(continued from previous page)

```
# addressing attribute
print(uds_message.addressing)
uds_message.addressing = AddressingType.FUNCTIONAL
print(uds_message.addressing)
uds_message.addressing = AddressingType.PHYSICAL.value
print(uds_message.addressing)
```

2.1.2 UDS Message Record

UdsMessageRecord class is meant to provide container for historic information of *diagnostic messages* that were either transmitted or received. A **user shall not create objects of this class** in normal cases, but one would probably use them quite often as they are returned by other layers of *uds* package.

All *UdsMessageRecord* **attributes are read only** (they are set only once upon an object creation) as they store historic data and history cannot be changed (*can't it, right?*). A user will face an exception if one tries to modify any attribute.

Attributes implemented in *UdsMessageRecord* class:

- *payload* - readable
- *addressing* - readable
- *direction* - readable
- *packets_records* - readable
- *transmission_start* - readable
- *transmission_end* - readable

2.2 UDS Packet Implementation

***UDS packets* implementation is divided into three parts:**

- *UDS Packet Type* - enums with *Network Protocol Control Information (N_PCI)* values definitions
- *UDS Packet* - storages for a temporary *Network Protocol Data Unit (N_PDU)* definition on the user side
- *UDS Packet Record* - storages for historic information of a *Network Protocol Data Unit (N_PDU)* that was either received or transmitted

2.2.1 UDS Packet Type

UDS packet types are supposed to be understood as values of *Network Protocol Control Information (N_PCI)*. Supported values of UDS packet types are defined in specially designed for this purpose enum classes.

Enum classes that implements UDS packet types:

- *AbstractUdsPacketType*

AbstractUdsPacketType

AbstractUdsPacketType class is an empty enum that is a parent class for all concrete UDS packet types enum classes. It **provides common API and values restriction** (UDS packet type values must be 4-bit integer) **for all children classes**.

A **user shall not use** *AbstractUdsPacketType* **directly**, but one is able (and encouraged) to use *AbstractUdsPacketType* implementation with any of its children classes.

Methods implemented in *AbstractUdsPacketType* class:

- *is_member()*
- *validate_member()*
- *add_member()*

2.2.2 UDS Packet

UDS packets **differs for each communication bus**, therefore **multiple classes implementing them are defined**. Each UDS packet class provides containers for *Network Protocol Data Unit (N_PDU)* information that are specific for a communication bus for which this class is relevant. **Objects of UDS packet classes might be used to execute complex operations** (provided in other subpackages) such as packets transmission or *desegmentation*.

Implemented UDS packet classes:

- *AbstractUdsPacket*

AbstractUdsPacket

AbstractUdsPacket class **contains common implementation and provides common API** for all UDS Packet classes as they are inheriting after *AbstractUdsPacket* class.

A **user shall not use** *AbstractUdsPacket* **directly**, but one is able (and encouraged) to use *AbstractUdsPacket* implementation with any of its children classes.

Properties implemented in *AbstractUdsPacket* class:

- *raw_data* - settable
- *addressing* - settable
- *packet_type* - readable

2.2.3 UDS Packet Record

UDS packet record is a container that stores historic information of *UDS packet (N_PDU)* that was either received or transmitted. UDS packets **differs for each communication bus**, therefore **multiple classes implementing UDS packet records are defined**.

A **user shall not create objects of UDS packet record classes** in normal cases, but one would probably use them quite often as they are returned by other layers of *uds* package.

Implemented UDS packet record classes:

- *AbstractUdsPacketRecord*

AbstractUdsPacketRecord

AbstractUdsPacketRecord class **contains common implementation and provides common API** for all UDS Packet classes as they are inheriting after *AbstractUdsPacketRecord* class.

A **user shall not use** *AbstractUdsPacketRecord* **directly**, but one is able (and encouraged) to use *AbstractUdsPacketRecord* implementation with any of its children classes.

Properties implemented in *AbstractUdsPacketRecord* class:

- *frame* - readable
- *direction* - readable
- *packet_type* - readable
- *raw_data* - readable and abstract (bus specific)
- *addressing* - readable and abstract (bus specific)
- *transmission_time* - readable and abstract (bus specific)

2.3 UDS Messages Data

Implementation of data parameters that are defined by UDS specification.

UDS data parameters:

- *Service Identifiers* - are implemented by:
  - *POSSIBLE_REQUEST_SIDS*
  - *RequestSID*
  - *POSSIBLE_RESPONSE_SIDS*
  - *ResponseSID*
- *Negative Response Codes*

2.3.1 Service Identifiers

POSSIBLE_REQUEST_SIDS

POSSIBLE_REQUEST_SIDS is a set with all possible values of *Service Identifier* data parameter in a *request message*.

RequestSID

Enum *RequestSID* contains definitions of request *Service Identifiers* values.

Methods implemented in *RequestSID* class:

- *is_request_sid()*
- *is_member()*
- *validate_member()*
- *add_member()*

POSSIBLE_RESPONSE_SIDS

POSSIBLE_RESPONSE_SIDS is a set with all possible values of *Service Identifier* data parameter in a *response message*.

ResponseSID

Enum *ResponseSID* contains definitions of response *Service Identifiers* values.

Methods implemented in *ResponseSID* class:

- *is_response_sid()*
- *is_member()*
- *validate_member()*
- *add_member()*

2.3.2 Negative Response Codes

Enum *NRC* contains definitions of all common (defined by ISO 14229) *Negative Response Codes* values.

Methods implemented in *NRC* class:

- *is_member()*
- *validate_member()*
- *add_member()*

2.4 Transmission Attributes

Attributes that describes UDS communication:

- *Addressing* - enum with UDS communication models
- *Transmission Direction* - enum with communication directions

2.4.1 Addressing

Enum *AddressingType* contains definitions of *addressing* values that determines UDS communication model:

- *PHYSICAL* - direct one to one communication (*physical addressing*)
- *FUNCTIONAL* - one to many communication (*functional addressing*)

Methods implemented in *AddressingType* class:

- *is_member()*
- *validate_member()*

2.4.2 Transmission Direction

Enum *TransmissionDirection* contains definitions of communication directions:

- *RECEIVED* - incoming
- *TRANSMITTED* - outcoming

Methods implemented in *TransmissionDirection* class:

- *is_member()*
- *validate_member()*

CHAPTER

THREE

SEGMENTATION

Implementation related to *segmentation* is located in *uds.segmentation* sub-package.

3.1 AbstractSegmenter

AbstractSegmenter defines common API and contains common code for all segmenter classes. Each concrete segmenter class implements segmentation strategy for a specific bus.

A **user shall not use** *AbstractSegmenter* **directly**, but one is able (and encouraged) to use *AbstractSegmenter* implementation with any of its children classes.

Attributes defined in *AbstractUdsPacketType* class:

- *supported_packet_classes* - readable and abstract (bus specific)
- *initial_packet_types* - readable and abstract (bus specific)

Methods defined in *AbstractUdsPacketType* class:

- *is_supported_packet()*
- *is_supported_packets_sequence()*
- is_initial_packet()
- *get_consecutive_packets_number()*
- *is_following_packets_sequence()*
- *is_complete_packets_sequence()*
- *segmentation()*
- *desegmentation()*

CHAPTER

FOUR

TRANSPORT INTERFACES

Transport interfaces are meant to handle Physical (layer 1), Data (layer 2), Network (layer 3) and Transport (layer 4) layers of UDS OSI model which are unique for every communication bus. First two layers (Physical and Data Link) are usually handled by external packages (e.g. python-can handles first two layers for CAN bus).

4.1 CAN Transport Interface

CAN FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

4.2 Ethernet Transport Interface

Ethernet FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

4.3 LIN Transport Interface

LIN FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

4.4 FlexRay Transport Interface

FlexRay FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

4.5 K-Line Transport Interface

K-Line FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

4.6 Custom Transport Interface

THIS FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

CHAPTER

FIVE

CLIENT SIMULATION

This chapter describes how to simulate client (diagnostic tester or any other node which sends its request to other ECUs) in UDS communication. Client simulation enables sending diagnostic requests and receiving diagnostic responses from connected nodes.

THIS FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

CHAPTER

SIX

SERVER SIMULATION

This chapter describes how to simulate server (any ECU that is recipient of diagnostic requests) in UDS communication. Server simulation supports defining diagnostic responses to incoming requests and then python program send them automatically according to the configuration provided by the user.

THIS FEATURE IS PLANNED BUT NOT IMPLEMENTED YET, THEREFORE THERE ARE NO MORE INFORMATION TO DISPLAY.

CHAPTER

SEVEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

7.1 uds

Package for handling Unified Diagnostic Services (UDS) protocol defined by ISO-14229.

The package is meant to provide tools that enables:

- monitoring UDS communication
- simulation of any UDS node (either a client or a server)
- testing of a device that supports UDS
- injection of communication faults on any layers 3-7 of UDS OSI Model

The package is created with an idea to support any communication bus:

- CAN
- LIN
- Ethernet
- FlexRay
- K-Line

7.1.1 Subpackages

uds.messages

A subpackage with tools for handling diagnostic messages.

It provides tools for:

- creating new diagnostic messages
- storing historic information about diagnostic messages that were either received or transmitted
- creating new packets
- storing historic information about packets that were either received or transmitted
- Service Identifiers (SID) definition

[1] Created with sphinx-autoapi

- Negative Response Codes (NRC) definition
- addressing types definition

Submodules

uds.messages.nrc

Module with entire *Negative Response Codes (NRC)* implementation.

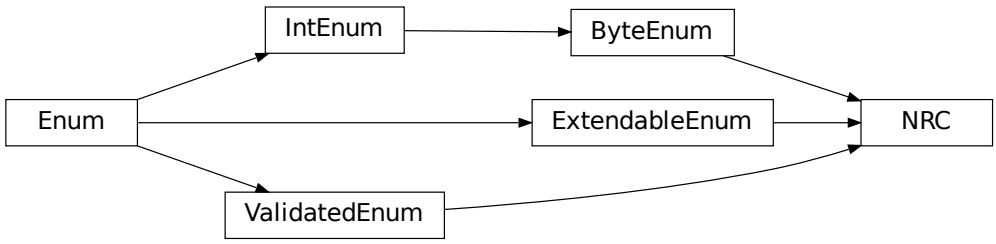
Module Contents

Classes

NRC	Negative Response Codes (NRC) values.

class uds.messages.nrc.**NRC**

Bases: uds.utilities.ByteEnum, uds.utilities.ValidatedEnum, uds.utilities.ExtendableEnum



Negative Response Codes (NRC) values.

Explanation of *NRC* values meaning is located in appendix A1 of ISO 14229-1 standard.

Initialize self. See help(type(self)) for accurate signature.

GeneralReject = 16

GeneralReject (0x10) NRC indicates that the requested action has been rejected by the server.

ServiceNotSupported = 17

ServiceNotSupported (0x11) NRC indicates that the requested action will not be taken because the server does not support the requested service.

SubFunctionNotSupported = 18

SubFunctionNotSupported (0x12) NRC indicates that the requested action will not be taken because the server does not support the service specific parameters of the request message.

IncorrectMessageLengthOrInvalidFormat = 19

IncorrectMessageLengthOrInvalidFormat (0x13) NRC indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified service or the format of the parameters do not match the prescribed format for the specified service.

ResponseTooLong = 20
ResponseTooLong (0x14) NRC shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer. This could occur if the response message exceeds the maximum size allowed by the underlying transport protocol or if the response message exceeds the server buffer size allocated for that purpose.

BusyRepeatRequest = 33
BusyRepeatRequest (0x21) NRC indicates that the server is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents.

ConditionsNotCorrect = 34
ConditionsNotCorrect (0x22) NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met.

RequestSequenceError = 36
RequestSequenceError (0x24) NRC indicates that the requested action will not be taken because the server expects a different sequence of request messages or message as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order.

NoResponseFromSubnetComponent = 37
NoResponseFromSubnetComponent (0x25) NRC indicates that the server has received the request but the requested action could not be performed by the server as a subnet component which is necessary to supply the requested information did not respond within the specified time.

FailurePreventsExecutionOfRequestedAction = 38
FailurePreventsExecutionOfRequestedAction (0x26) NRC indicates that the requested action will not be taken because a failure condition, identified by a DTC (with at least one DTC status bit for TestFailed, Pending, Confirmed or TestFailedSinceLastClear set to 1), has occurred and that this failure condition prevents the server from performing the requested action.

RequestOutOfRange = 49
RequestOutOfRange (0x31) NRC indicates that the requested action will not be taken because the server has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a DataIdentifier/RoutineIdentifer that is not supported or not supported in active session.

SecurityAccessDenied = 51
SecurityAccessDenied (0x33) NRC indicates that the requested action will not be taken because the server's security strategy has not been satisfied by the client.

AuthenticationRequired = 52
AuthenticationRequired (0x34) NRC indicates that the requested service will not be taken because the client has insufficient rights based on its Authentication state.

InvalidKey = 53
InvalidKey (0x35) NRC indicates that the server has not given security access because the key sent by the client did not match with the key in the server's memory. This counts as an attempt to gain security.

ExceedNumberOfAttempts = 54
ExceedNumberOfAttempts (0x36) NRC indicates that the requested action will not be taken because the client has unsuccessfully attempted to gain security access more times than the server's security strategy will allow.

RequiredTimeDelayNotExpired = 55
RequiredTimeDelayNotExpired (0x37) NRC indicates that the requested action will not be taken because

the client's latest attempt to gain security access was initiated before the server's required timeout period had elapsed.

SecureDataTransmissionRequired = 56
SecureDataTransmissionRequired (0x38) NRC indicates that the requested service will not be taken because the requested action is required to be sent using a secured communication channel.

SecureDataTransmissionNotAllowed = 57
SecureDataTransmissionNotAllowed (0x39) NRC indicates that this message was received using the SecuredDataTransmission (SID 0x84) service. However, the requested action is not allowed to be sent using the SecuredDataTransmission (0x84) service.

SecureDataVerificationFailed = 58
SecureDataVerificationFailed (0x3A) NRC indicates that the message failed in the security sub-layer.

CertificateVerificationFailed_InvalidTimePeriod = 80
CertificateVerificationFailed_InvalidTimePeriod (0x50) NRC indicates that date and time of the server does not match the validity period of the Certificate.

CertificateVerificationFailed_InvalidSignature = 81
CertificateVerificationFailed_InvalidSignature (0x51) NRC indicates that signature of the Certificate could not be verified.

CertificateVerificationFailed_InvalidChainOfTrust = 82
CertificateVerificationFailed_InvalidChainOfTrust (0x52) NRC indicates that The Certificate could not be verified against stored information about the issuing authority.

CertificateVerificationFailed_InvalidType = 83
CertificateVerificationFailed_InvalidType (0x53) NRC indicates that the Certificate does not match the current requested use case.

CertificateVerificationFailed_InvalidFormat = 84
CertificateVerificationFailed_InvalidFormat (0x54) NRC indicates that the Certificate could not be evaluated because the format requirement has not been met.

CertificateVerificationFailed_InvalidContent = 85
CertificateVerificationFailed_InvalidContent (0x55) NRC indicates that the Certificate could not be verified because the content does not match.

CertificateVerificationFailed_InvalidScope = 86
CertificateVerificationFailed_InvalidScope (0x56) NRC indicates that the scope of the Certificate does not match the contents of the server.

CertificateVerificationFailed_InvalidCertificate = 87
CertificateVerificationFailed_InvalidCertificate (0x57) NRC indicates that the Certificate received from client is invalid, because the server has revoked access for some reason.

OwnershipVerificationFailed = 88
OwnershipVerificationFailed (0x58) NRC indicates that delivered Ownership does not match the provided challenge or could not verified with the own private key.

ChallengeCalculationFailed = 89
ChallengeCalculationFailed (0x59) NRC indicates that the challenge could not be calculated on the server side.

SettingAccessRightsFailed = 90
SettingAccessRightsFailed (0x5A) NRC indicates that the server could not set the access rights.

SessionKeyCreationOrDerivationFailed = 91
SessionKeyCreationOrDerivationFailed (0x5B) NRC indicates that the server could not create or derive a session key.

ConfigurationDataUsageFailed = 92
ConfigurationDataUsageFailed (0x5C) NRC indicates that the server could not work with the provided configuration data.

DeAuthenticationFailed = 93
DeAuthenticationFailed (0x5D) NRC indicates that DeAuthentication was not successful, server could still be unprotected.

UploadDownloadNotAccepted = 112
UploadDownloadNotAccepted (0x70) NRC indicates that an attempt to upload/download to a server's memory cannot be accomplished due to some fault conditions.

TransferDataSuspended = 113
TransferDataSuspended (0x71) NRC indicates that a data transfer operation was halted due to some fault. The active transferData sequence shall be aborted.

GeneralProgrammingFailure = 114
GeneralProgrammingFailure (0x72) NRC indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory).

WrongBlockSequenceCounter = 115
WrongBlockSequenceCounter (0x73) NRC indicates that the server detected an error in the sequence of blockSequenceCounter values. Note that the repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server.

RequestCorrectlyReceived_ResponsePending = 120
RequestCorrectlyReceived_ResponsePending (0x78) NRC ndicates that the request message was received correctly, and that all parameters in the request message were valid (these checks can be delayed until after sending this NRC if executing the boot software), but the action to be performed is not yet completed and the server is not yet ready to receive another request. As soon as the requested service has been completed, the server shall send a positive response message or negative response message with a response code different from this.

SubFunctionNotSupportedInActiveSession = 126
SubFunctionNotSupportedInActiveSession (0x7E) NRC indicates that the requested action will not be taken because the server does not support the requested SubFunction in the session currently active. This NRC shall only be used when the requested SubFunction is known to be supported in another session, otherwise response code SubFunctionNotSupported shall be used.

ServiceNotSupportedInActiveSession = 127
ServiceNotSupportedInActiveSession (0x7F) NRC indicates that the requested action will not be taken because the server does not support the requested service in the session currently active. This NRC shall only be used when the requested service is known to be supported in another session, otherwise response code serviceNotSupported shall be used.

RpmTooHigh = 129
RpmTooHigh (0x81) NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is above a preprogrammed maximum threshold).

RpmTooLow = 130
RpmTooLow (0x82) NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is below a preprogrammed minimum threshold).

EngineIsRunning = 131
EngineIsRunning (0x83) NRC is required for those actuator tests which cannot be actuated while the Engine is running. This is different from RPM too high negative response, and shall be allowed.

EngineIsNotRunning = 132
EngineIsNotRunning (0x84) NRC is required for those actuator tests which cannot be actuated unless the

Engine is running. This is different from RPM too low negative response, and shall be allowed.

EngineRunTimeTooLow = 133
EngineRunTimeTooLow (0x85) NRC indicates that the requested action will not be taken because the server prerequisite condition for engine run time is not met (current engine run time is below a preprogrammed limit).

TemperatureTooHigh = 134
TemperatureTooHigh (0x86) NRC indicates that the requested action will not be taken because the serve prerequisite condition for temperature is not met (current temperature is above a preprogrammed maximum threshold).

TemperatureTooLow = 135
TemperatureTooLow (0x87) NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is below a preprogrammed minimum threshold).

VehicleSpeedTooHigh = 136
VehicleSpeedTooHigh (0x88) NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is above a preprogrammed maximum threshold).

VehicleSpeedTooLow = 137
VehicleSpeedTooLow (0x89) NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is below a preprogrammed minimum threshold).

ThrottleOrPedalTooHigh = 138
ThrottleOrPedalTooHigh (0x8A) NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current throttle/pedal position is above a preprogrammed maximum threshold).

ThrottleOrPedalTooLow = 139
ThrottleOrPedalTooLow (0x8B) NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current throttle/pedal position is below a preprogrammed minimum threshold).

TransmissionRangeNotInNeutral = 140
TransmissionRangeNotInNeutral (0x8C) NRC indicates that the requested action will not be taken because the server prerequisite condition for being in neutral is not met (current transmission range is not in neutral).

TransmissionRangeNotInGear = 141
TransmissionRangeNotInGear (0x8D) NRC indicates that the requested action will not be taken because the server prerequisite condition for being in gear is not met (current transmission range is not in gear).

BrakeSwitchOrSwitchesNotClosed = 143
BrakeSwitchOrSwitchesNotClosed (0x8F) NRC indicates that for safety reasons, this is required for certain tests before it begins, and shall be maintained for the entire duration of the test.

ShifterLeverNotInPark = 144
ShifterLeverNotInPark (0x90) NRC indicates that for safety reasons, this is required for certain tests before it begins, and shall be maintained for the entire duration of the test.

TorqueConvertClutchLocked = 145
TorqueConvertClutchLocked (0x91) RC indicates that the requested action will not be taken because the server prerequisite condition for torque converter clutch is not met (current torque converter clutch status above a preprogrammed limit or locked).

VoltageTooHigh = 146
VoltageTooHigh (0x92) NRC indicates that the requested action will not be taken because the server pre-

requisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is above a preprogrammed maximum threshold).

VoltageTooLow = 147
VoltageTooLow (0x93) NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is below a preprogrammed minimum threshold).

ResourceTemporarilyNotAvailable = 148
ResourceTemporarilyNotAvailable (0x94) NRC indicates that the server has received the request but the requested action could not be performed by the server because an application which is necessary to supply the requested information is temporality not available. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.

uds.messages.service_identifiers

Service Identifiers (SID) implementation.

Module Contents

Classes

RequestSID	Request Service Identifier values for all services that are defined in ISO 14229-1:2020.
ResponseSID	Response Service Identifier values for all services that are defined in ISO 14229-1:2020.

Attributes

POSSIBLE_REQUEST_SIDS	Set with all possible values of Request SID data parameter according to SAE J1979 and ISO 14229 standards.
POSSIBLE_RESPONSE_SIDS	Set with all possible values of Response SID data parameter according to SAE J1979 and ISO 14229 standards.

uds.messages.service_identifiers.**POSSIBLE_REQUEST_SIDS :uds.utilities.RawBytesSet**
Set with all possible values of Request SID data parameter according to SAE J1979 and ISO 14229 standards.

uds.messages.service_identifiers.**POSSIBLE_RESPONSE_SIDS :uds.utilities.RawBytesSet**
Set with all possible values of Response SID data parameter according to SAE J1979 and ISO 14229 standards.

exception uds.messages.service_identifiers.**UnrecognizedSIDWarning**
Bases: Warning

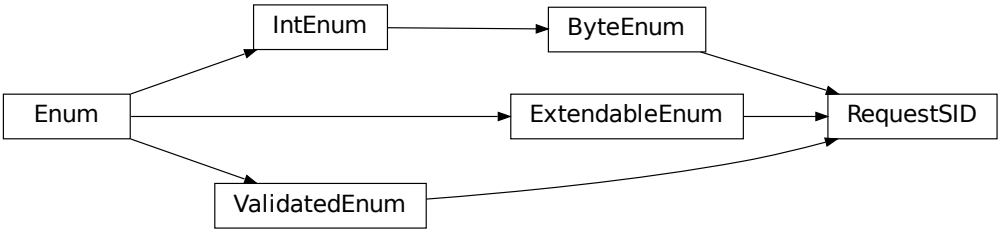
Warning about SID value that is legit but not recognized by the package.

If you want to register a SID value, you need to define members (for this SID) manually using *add_member()* method (on *RequestSID* and *ResponseSID* classes). You can also create feature request in the UDS project issues management system to register the SID value (for which this warning was raised).

Initialize self. See help(type(self)) for accurate signature.

class uds.messages.service_identifiers.**RequestSID**

Bases: uds.utilities.ByteEnum, uds.utilities.ValidatedEnum, uds.utilities.ExtendableEnum



Request Service Identifier values for all services that are defined in ISO 14229-1:2020.

Note: Request *SID* is always the first payload byte of all request messages.

Initialize self. See help(type(self)) for accurate signature.

DiagnosticSessionControl = 16

ECUReset = 17

SecurityAccess = 39

CommunicationControl = 40

Authentication = 41

TesterPresent = 62

ControlDTCSetting = 133

ResponseOnEvent = 134

LinkControl = 135

ReadDataByIdentifier = 34

ReadMemoryByAddress = 35

ReadScalingDataByIdentifier = 36

ReadDataByPeriodicIdentifier = 42

DynamicallyDefineDataIdentifier = 44

WriteDataByIdentifier = 46

WriteMemoryByAddress = 61

ClearDiagnosticInformation = 20

ReadDTCInformation = 25

InputOutputControlByIdentifier = 47

RoutineControl = 49

RequestDownload = 52

RequestUpload = 53

TransferData = 54

RequestTransferExit = 55

RequestFileTransfer = 56

SecuredDataTransmission = 132

classmethod is_request_sid(*cls*, *value*)

Check whether given value is Service Identifier (SID).

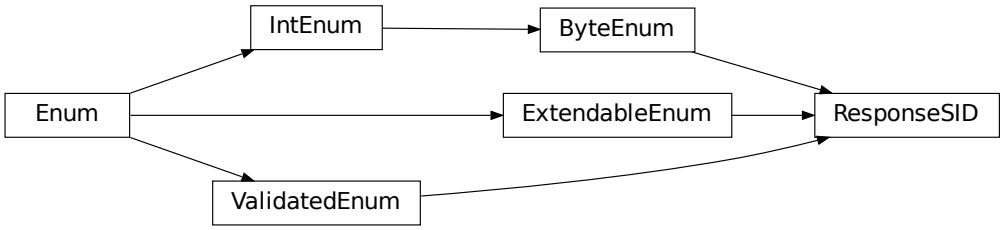
Parameters **value** (*uds.utilities.RawByte*) – Value to check.

Returns True if value is valid SID, else False.

Return type bool

class uds.messages.service_identifiers.**ResponseSID**

Bases: uds.utilities.ByteEnum, uds.utilities.ValidatedEnum, uds.utilities.ExtendableEnum



Response Service Identifier values for all services that are defined in ISO 14229-1:2020.

Note: Response *SID* is always the first payload byte of all request messages.

Note: This Enum contains multiple members (for all the services as RequestSID), but most of them are dynamically (implicitly) added and invisible in the documentation.

Initialize self. See help(type(self)) for accurate signature.

NegativeResponse = 127

classmethod is_response_sid(*cls*, *value*)
Check whether given value is Response Service Identifier (RSID).

Parameters **value** (*uds.utilities.RawByte*) – Value to check.

Returns True if value is valid RSID, else False.

Return type bool

uds.messages.transmission_attributes

Attributes that describes UDS communication.

Module Contents

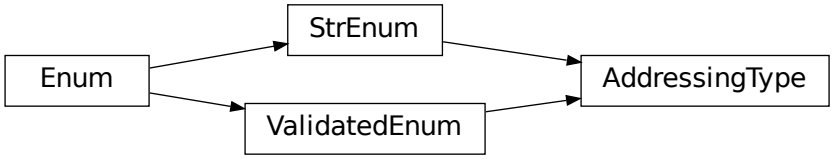
Classes

AddressingType	Model of UDS communication.
TransmissionDirection	Direction of a communication.

Attributes

AddressingMemberTyping	Typing alias that describes AddressingType member.
DirectionMemberTyping	Typing alias that describes TransmissionDirection member.

class uds.messages.transmission_attributes.**AddressingType**
Bases: aenum.StrEnum, uds.utilities.ValidatedEnum



Model of UDS communication.

Initialize self. See help(type(self)) for accurate signature.

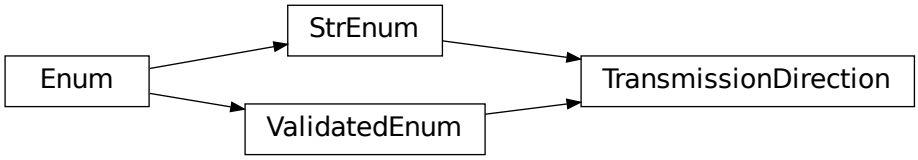
PHYSICAL = Physical
Physical addressing - 1 (client) to 1 (server) communication.

FUNCTIONAL = Functional
Functional addressing - 1 (client) to many (servers) communication.

uds.messages.transmission_attributes.**AddressingMemberTyping**

Typing alias that describes AddressingType member.

class uds.messages.transmission_attributes.**TransmissionDirection**
Bases: aenum.StrEnum, uds.utilities.ValidatedEnum



Direction of a communication.

Initialize self. See help(type(self)) for accurate signature.

RECEIVED = Rx
Incoming transmission from the perspective of the code.

TRANSMITTED = Tx
Outcoming transmission from the perspective of the code.

uds.messages.transmission_attributes.**DirectionMemberTyping**
Typing alias that describes TransmissionDirection member.

uds.messages.uds_message

Module with common implementation of all diagnostic messages (requests and responses).

Diagnostic messages are defined on higher layers of UDS OSI Model.

Module Contents

Classes

UdsMessage	Definition of a diagnostic message.
UdsMessageRecord	Storage for historic information of a diagnostic message that was either received or transmitted.

class uds.messages.uds_message.**UdsMessage**(*payload*, *addressing*)
Definition of a diagnostic message.

Objects of this class act as a storage for all relevant attributes of a diagnostic message. Later on, such object might be used in segmentation process or to transmit the message. Once a message is transmitted, its historic data would be stored in *UdsMessageRecord*.

Create a storage for a single diagnostic message.

Parameters

- **payload** (*uds.utilities.RawBytes*) – Raw bytes of payload that this diagnostic message

carries.

- **addressing** (*uds.messages.transmission_attributes.AddressingMemberTyping*) – Addressing type for which this message is relevant.

property payload(*self*)
Raw bytes of payload that this diagnostic message carries.

Return type uds.utilities.RawBytesTuple

property addressing(*self*)
Addressing type for which this message is relevant.

Return type *uds.messages.transmission_attributes.AddressingType*

class uds.messages.uds_message.**UdsMessageRecord**(*payload*, *packets_records*)
Storage for historic information of a diagnostic message that was either received or transmitted.

Create a record of a historic information about a diagnostic message that was either received or transmitted.

Parameters

- **packets_records** (*uds.messages.uds_packet.PacketsRecordsSequence*) – Sequence (in transmission order) of UDS packets records that carried this diagnostic message.
- **payload** (*uds.utilities.RawBytes*) –

static __validate_packets_records(*value*)
Validate whether the argument contains UDS Packets records.

Parameters **value** (*Any*) – Value to validate.

Raises

- **TypeError** – UDS Packet Records sequence is not list or tuple type.
- **ValueError** – At least one of UDS Packet Records sequence elements is not an object of *AbstractUdsPacketRecord* class.

Return type None

property payload(*self*)
Raw bytes of payload that this diagnostic message carried.

Return type uds.utilities.RawBytesTuple

property packets_records(*self*)
Sequence (in transmission order) of UDS packets records that carried this diagnostic message.

Return type uds.messages.uds_packet.PacketsRecordsTuple

property addressing(*self*)
Addressing type which was used to transmit this message.

Return type *uds.messages.transmission_attributes.AddressingType*

property direction(*self*)
Information whether this message was received or sent by the code.

Return type *uds.messages.transmission_attributes.TransmissionDirection*

property transmission_start(*self*)
Time stamp when transmission of this messages was initiated.

It is determined by a moment of time when the first packet (that carried this message) was published to a bus (either received or transmitted).

Returns Time stamp when transmission of this message was initiated.

Return type uds.utilities.TimeStamp

property transmission_end(*self*)

Time stamp when transmission of this messages was completed.

It is determined by a moment of time when the last packet (that carried this message) was published to a bus (either received or transmitted).

Returns Time stamp when transmission of this message was completed.

Return type uds.utilities.TimeStamp

uds.messages.uds_packet

Module with common implementation of UDS packets for all bus types.

UDS Packets are defined on middle layers of UDS OSI Model.

Module Contents

Classes

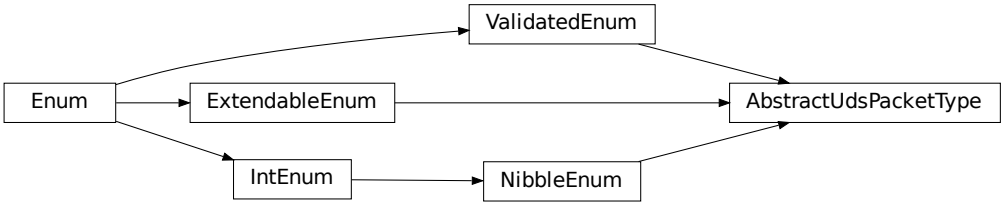
AbstractUdsPacketType	Abstract definition of UDS packet type.
AbstractUdsPacket	Abstract definition of UDS Packet (Network Protocol Data Unit - N_PDU).
AbstractUdsPacketRecord	Abstract definition of a storage for historic information about transmitted or received UDS Packet.

Attributes

PacketTypesTuple	Typing alias of a tuple filled with *AbstractUdsPacketType* members.
PacketsDefinitionTuple	Typing alias of a tuple filled with *AbstractUdsPacket* instances.
PacketsDefinitionSequence	Typing alias of a sequence filled with *AbstractUdsPacket* instances.
PacketsRecordsTuple	Typing alias of a tuple filled with *AbstractUdsPacketRecord* instances.
PacketsRecordsSequence	Typing alias of a sequence filled with *AbstractUdsPacketRecord* instances.
PacketTyping	Typing alias of UDS packet.
PacketsTuple	Typing alias of a tuple filled with UDS packets.
PacketsSequence	Typing alias of a sequence filled with UDS packets.

class uds.messages.uds_packet.**AbstractUdsPacketType**

Bases: uds.utilities.NibbleEnum, uds.utilities.ValidatedEnum, uds.utilities.ExtendableEnum



Abstract definition of UDS packet type.

Packet type information is carried by *Network Protocol Control Information (N_PCI)*. Enums with packet types (N_PCI) values for certain buses (e.g. CAN, LIN, FlexRay) must inherit after this class.

Note: There are some differences in values for each bus (e.g. LIN does not use Flow Control).

Initialize self. See help(type(self)) for accurate signature.

abstract classmethod is_initial_packet_type(*cls*, *value*)

Check whether given argument is a member or a value of packet type that initiates a diagnostic message.

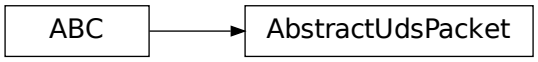
Parameters **value** (*Any*) – Value to check.

Returns True if given argument is a packet type that initiates a diagnostic message, else False.

Return type bool

class uds.messages.uds_packet.**AbstractUdsPacket**(*raw_data*, *addressing*)

Bases: abc.ABC



Abstract definition of UDS Packet (Network Protocol Data Unit - N_PDU).

Create a storage for a single UDS packet.

Parameters

- **raw_data** (*uds.utilities.RawBytes*) – Raw bytes of UDS packet data.
- **addressing** (*uds.messages.transmission_attributes.AddressingMemberTyping*) – Addressing type for which this packet is relevant.

property addressing(*self*)

Addressing type for which this packet is relevant.

Return type *uds.messages.transmission_attributes.AddressingType*

property raw_data(*self*)

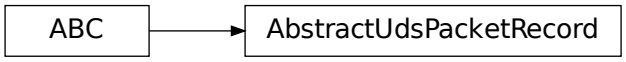
Raw bytes of data that this packet carries.

Return type uds.utilities.RawBytesTuple

property packet_type(*self*)
UDS packet type value - N_PCI value of this N_PDU.

Return type *AbstractUdsPacketType*

class uds.messages.uds_packet.**AbstractUdsPacketRecord**(*frame*, *direction*)
Bases: abc.ABC



Abstract definition of a storage for historic information about transmitted or received UDS Packet.

Create a record of a historic information about a packet that was either received or transmitted.

Parameters

- **frame** (*object*) – Frame that carried this UDS packet.
- **direction** (*uds.messages.transmission_attributes.DirectionMemberTyping*) – Information whether this packet was transmitted or received.

abstract __validate_frame(*self*, *value*)
Validate whether the argument contains value with a frame object.

Parameters **value** (*Any*) – Value to validate.

Raises

- **TypeError** – The frame argument has other type than expected.
- **ValueError** – Some attribute of the frame argument is missing or its value is unexpected.

Return type None

property frame(*self*)
Frame that carried this packet.

Return type object

property direction(*self*)
Information whether this packet was transmitted or received.

Return type *uds.messages.transmission_attributes.TransmissionDirection*

property raw_data(*self*)
Raw bytes of data that this packet carried.

Return type uds.utilities.RawBytesTuple

property addressing(*self*)
Addressing type over which this packet was transmitted.

Return type *uds.messages.transmission_attributes.AddressingType*

property transmission_time(*self*)
Time stamp when this packet was fully transmitted on a bus.

Return type uds.utilities.TimeStamp

property packet_type(*self*)
UDS packet type value - N_PCI value of this N_PDU.

Return type *AbstractUdsPacketType*

uds.messages.uds_packet.**PacketTypesTuple**
Typing alias of a tuple filled with *AbstractUdsPacketType* members.

uds.messages.uds_packet.**PacketsDefinitionTuple**
Typing alias of a tuple filled with *AbstractUdsPacket* instances.

uds.messages.uds_packet.**PacketsDefinitionSequence**
Typing alias of a sequence filled with *AbstractUdsPacket* instances.

uds.messages.uds_packet.**PacketsRecordsTuple**
Typing alias of a tuple filled with *AbstractUdsPacketRecord* instances.

uds.messages.uds_packet.**PacketsRecordsSequence**
Typing alias of a sequence filled with *AbstractUdsPacketRecord* instances.

uds.messages.uds_packet.**PacketTyping**
Typing alias of UDS packet.

uds.messages.uds_packet.**PacketsTuple**
Typing alias of a tuple filled with UDS packets.

uds.messages.uds_packet.**PacketsSequence**
Typing alias of a sequence filled with UDS packets.

uds.segmentation

A subpackage with tools for executing *segmentation*.

It defines:

- common API interface for all segmentation duties

Submodules

uds.segmentation.abstract_segmenter

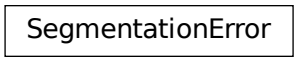
Definition of API for segmentation and desegmentation strategies.

Module Contents

Classes

AbstractSegmenter	Abstract definition of a segmenter class.

exception uds.segmentation.abstract_segmenter.**SegmentationError**
Bases: ValueError

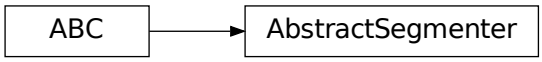


UDS segmentation or desegmentation process cannot be completed due to input data inconsistency.

Initialize self. See help(type(self)) for accurate signature.

class uds.segmentation.abstract_segmenter.**AbstractSegmenter**

Bases: abc.ABC



Abstract definition of a segmenter class.

Segmenter classes defines UDS segmentation and desegmentation strategies. They contain helper methods that are essential for successful *segmentation* and *desegmentation* execution. Each concrete segmenter class handles a single bus.

property supported_packet_classes(*self*)

Classes that define packet objects supported by this segmenter.

Return type Tuple[type]

property initial_packet_types(*self*)

Types of packets that initiates a diagnostic message transmission for the managed bus.

Return type uds.messages.PacketTypesTuple

is_supported_packet(*self*, *value*)

Check if the argument value is a packet object of a supported type.

Parameters **value** (*Any*) – Value to check.

Returns True if provided value is an object of a supported packet type, False otherwise.

Return type bool

is_supported_packets_sequence(*self*, *value*)

Check if the argument value is a packet sequence of a supported type.

Parameters **value** (*Any*) – Value to check.

Returns True if provided value is a packet sequence of a supported type, False otherwise.

Return type bool

abstract is_following_packets_sequence(*self*, *packets*)

Check whether provided packets are a sequence of following packets.

Note: This function will return True under following conditions:

- a sequence of packets was provided
- the first packet in the sequence is an initial packet
- no other packet in the sequence is an initial packet
- each packet (except the first one) is a consecutive packet for the previous packet in the sequence

Parameters **packets** (*uds.messages.PacketsSequence*) – Packets sequence to check.

Raises **ValueError** – Provided value is not a packets sequence of a supported type.

Returns True if the provided packets are a sequence of following packets, otherwise False.

Return type bool

is_complete_packets_sequence(*self*, *packets*)

Check whether provided packets are full sequence of packets that form exactly one diagnostic message.

Parameters **packets** (*uds.messages.PacketsSequence*) – Packets sequence to check.

Returns True if the packets form exactly one diagnostic message. False if there are missing, additional or inconsistent (e.g. two packets that initiate a message) packets.

Return type bool

abstract get_consecutive_packets_number(*self*, *first_packet*)

Get number of consecutive packets that must follow this packet to fully store a diagnostic message.

Parameters **first_packet** (*uds.messages.PacketTyping*) – The first packet of a segmented diagnostic message.

Raises **ValueError** – Provided value is not an an initial packet.

Returns Number of following packets that together carry a diagnostic message.

Return type int

abstract segmentation(*self*, *message*)

Perform segmentation of a diagnostic message.

Parameters **message** (*uds.messages.UdsMessage*) – UDS message to divide into UDS packets.

Raises **TypeError** – Provided 'message' argument is not *UdsMessage* type.

Returns UDS packets that are an outcome of UDS message segmentation.

Return type uds.messages.PacketsDefinitionTuple

abstract desegmentation(*self*, *packets*)

Perform desegmentation of UDS packets.

Parameters **packets** (*uds.messages.PacketsSequence*) – UDS packets to desegment into UDS message.

Raises ***SegmentationError*** – Provided packets are not a complete packet sequence that form a diagnostic message.

Returns A diagnostic message that is an outcome of UDS packets desegmentation.

Return type Union[uds.messages.UdsMessage, uds.messages.UdsMessageRecord]

uds.transport_interface

A subpackage with UDS middle layers (Transport and Network) implementation.

TODO: provide more information when implementing tasks that expands capabilities of this package

Submodules

uds.transport_interface.packet_queue

Module with common implementation of UDS Packets queues.

Module Contents

Classes

ReceivedPacketsQueue	Queue for storing received packets.

class uds.transport_interface.packet_queue.**ReceivedPacketsQueue**(*packet_class=AbstractUdsPacketRecord*)

Queue for storing received packets.

Create a queue as a storage for received packets.

Parameters **packet_class** (*type*) – A class that defines UDS packets type that is accepted by this queue. One can use this parameter to restrict packets managed by this queue.

Raises **TypeError** – Provided packet_class argument is not a class that inherits after :class:"uds.messages.uds_packet.AbstractUdsPacketRecord".

abstract __del__(*self*)

Delete the object safely.

To satisfy safe closure or tasks using the queue:

- prevent new tasks creations
- close or await already started tasks

Return type NoReturn

abstract __len__(*self*)

Get number of packets that are currently stored by the queue.

Return type int

abstract is_empty(*self*)

Check if queue is empty.

Returns True if queue is empty (does not contain any packets), False otherwise.

Return type bool

abstract packet_task_done(*self*)

Inform that a task related to one packet was completed.

This method is used during closing all tasks safely and quietly.

Return type None

abstract async get_packet(*self*)
Get the next received packet from the queue.

Note: **If called, when there are no packets in the queue, then execution would await until another packet** is received.

Returns The next received packet.

Return type uds.messages.AbstractUdsPacketRecord

abstract async put_packet(*self*, *packet*)
Add a packet (that was just received) to the end of the queue.

Parameters **packet** (*uds.messages.AbstractUdsPacketRecord*) – A packet that was just received.

Return type None

uds.utilities

Various helper functions, classes and variables that are shared and reused within the project.

Submodules

uds.utilities.common_types

Module with all common types (and its aliases) used in the package and helper functions for these types.

Module Contents

Functions

validate_raw_bytes(value)	Validate whether provided value stores raw bytes.

Attributes

RawByte	Typing alias of byte value - integer in range 0x00-0xFF - that is used by the package.
RawBytesTuple	Typing alias of a tuple filled with byte values.
RawBytesSet	Typing alias of a set filled with byte values.
RawBytes	Typing alias of a sequence filled with byte values.
TimeMilliseconds	Typing alias of an amount of time in milliseconds that is used by the package.
TimeStamp	Typing alias of a timestamp that is used by the package.

uds.utilities.common_types.**RawByte**
Typing alias of byte value - integer in range 0x00-0xFF - that is used by the package.

uds.utilities.common_types.**RawBytesTuple**
Typing alias of a tuple filled with byte values.

uds.utilities.common_types.**RawBytesSet**
Typing alias of a set filled with byte values.

uds.utilities.common_types.**RawBytes**
Typing alias of a sequence filled with byte values.

uds.utilities.common_types.**TimeMilliseconds**
Typing alias of an amount of time in milliseconds that is used by the package.

uds.utilities.common_types.**TimeStamp**
Typing alias of a timestamp that is used by the package.

uds.utilities.common_types.**validate_raw_bytes**(*value*)
Validate whether provided value stores raw bytes.

Parameters **value** (*Any*) – Value to validate.

Raises

- **TypeError** – Value is not tuple or list type.
- **ValueError** – Value does not contain raw bytes (int value between 0x00-0xFF) only.

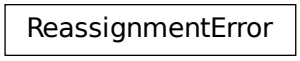
Return type None

uds.utilities.custom_exceptions

Custom exception used within the project.

Module Contents

exception uds.utilities.custom_exceptions.**ReassignmentError**
Bases: Exception



Attempt to set a new value of an attribute that cannot be changed after the initial value was already set.

Initialize self. See help(type(self)) for accurate signature.

uds.utilities.enums

Module with special Enums implementations.

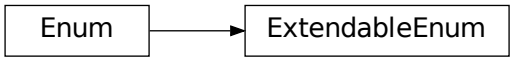
Module Contents

Classes

ExtendableEnum	Enum that supports new members adding.
ValidatedEnum	Enum that supports members validation.
ByteEnum	Enum which members are one byte integers (0x00-0xFF) only.
NibbleEnum	Enum which members are one nibble (4 bits) integers (0x0-0xF) only.

class uds.utilities.enums.**ExtendableEnum**

Bases: aenum.Enum



Enum that supports new members adding.

classmethod add_member(*cls*, *name*, *value*)

Register a new member.

Parameters

- **name** (*str*) – Name of a new member.
- **value** (*Any*) – Value of a new member.

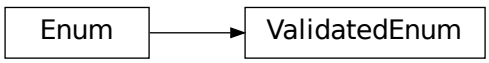
Raises **ValueError** – Such name or value is already in use.

Returns The new member that was just created.

Return type aenum.Enum

class uds.utilities.enums.**ValidatedEnum**

Bases: aenum.Enum



Enum that supports members validation.

classmethod is_member(*cls*, *value*)
Check whether given argument is a member or a value stored by this Enum.

Parameters **value** (*Any*) – Value to check.

Returns True if given argument is a member or a value of this Enum, else False.

Return type bool

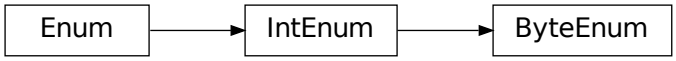
classmethod validate_member(*cls*, *value*)
Validate whether given argument is a member or a value stored by this Enum.

Parameters **value** (*Any*) – Value to validate.

Raises **TypeError** – Provided value is not a member neither a value of this Enum.

Return type None

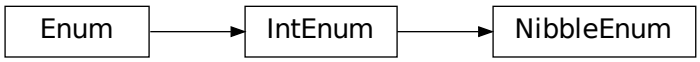
class uds.utilities.enums.**ByteEnum**
Bases: aenum.IntEnum



Enum which members are one byte integers (0x00-0xFF) only.

Initialize self. See help(type(self)) for accurate signature.

class uds.utilities.enums.**NibbleEnum**
Bases: aenum.IntEnum



Enum which members are one nibble (4 bits) integers (0x0-0xF) only.

Initialize self. See help(type(self)) for accurate signature.

CHAPTER

EIGHT

UDS KNOWLEDGE BASE

If you are not an UDS expert, this part of documentation is created for you. It is meant to provide a technical support for every user of UDS package so you can better understand the code, but also UDS protocol itself.

8.1 UDS OSI Model

Overview of UDS OSI model.

8.1.1 UDS Standards

UDS is defined by multiple standards which are the main source of information and requirements about this protocol. Full list of standards is included in the table below:

OSI Layer	Common	CAN	FlexRay	Ethernet	K-Line	LIN
Layer 7 Application	ISO 14229-1 ISO 27145-3	ISO 14229-3	ISO 14229-4	ISO 14229-5	ISO 14229-6	ISO 14229-7
Layer 6 Presentation	ISO 27145-2					
Layer 5 Session	ISO 14229-2					
Layer 4 Transport	ISO 27145-4	ISO 15765-2	ISO 10681-2	ISO 13400-2	Not applicable	ISO 17987-2
Layer 3 Network						
Layer 2 Data		ISO 11898-1	ISO 17458-2	ISO 13400-3	ISO 14230-2	ISO 17987-3
Layer 1 Physical		ISO 11898-2 ISO 11898-3	ISO 17458-4		ISO 14230-1	ISO 17987-4

Where:

- OSI Layer - OSI Model Layer for which standards are relevant
- Common - standards mentioned in this column are always relevant for UDS communication regardless of bus used
- CAN - standards which are specific for UDS on CAN implementation
- FlexRay - standards which are specific for UDS on FlexRay implementation
- Ethernet - standards which are specific for UDS on IP implementation
- K-Line - standards which are specific for UDS on K-Line implementation

- LIN - standards which are specific for UDS on LIN implementation

8.1.2 UDS Functionalities

An overview of features that are required to fully implement UDS protocol is presented in the table below:

OSI Layer	Functionalities	Implementation
Layer 7 Application	• diagnostic messages support	• `uds.messages.uds_message`
Layer 6 Presentation	• diagnostic messages data interpretation • messaging database import from a file • messaging database export to a file	*To be provided with Database feature.*
Layer 5 Session	• Client simulation • Server simulation	*To be provided with Client feature.* *To be provided with Server feature.*
Layer 4 Transport Layer 3 Network	• UDS packet support • bus specific segmentation • bus specific packets transmission	• `uds.messages.uds_packet` *To be extended with features:* *- Segmentation* *- Transport Interface* *- support for each bus*
Layer 2 Data Layer 1 Physical	• frames transmission • frames receiving	External python packages for bus handling: • CAN *More driver packages to be decided.*

Where:

- OSI Layer - considered OSI Model Layer
- Functionalities - functionalities required in the implementation to handle considered UDS OSI layer
- Implementation - UDS package implementation that provides mentioned functionalities

8.1.3 Protocol Data Units

Each layer of OSI Model defines their own Protocol Data Unit (PDU). To make things simpler for the users and our developers, in the implementation we distinguish following PDUs:

- Application Protocol Data Unit (A_PDU) - called *diagnostic message* or *UDS Message* in the implementation and documentation. More information about A_PDU can be found in:
  - *knowledge base section - diagnostic message*
  - *implementation - diagnostic message*
- Network Protocol Data Unit (N_PDU) - called *UDS packet* in the implementation and documentation. More information about N_PDU can be found in:
  - *knowledge base section - UDS packet*

  - *implementation section - UDS packet*

- Data Protocol Data Unit (D_PDU) - called *frame* in the implementation and documentation. We do not have any internal frames documentation. Implementation of frames is usually provided by external packages.

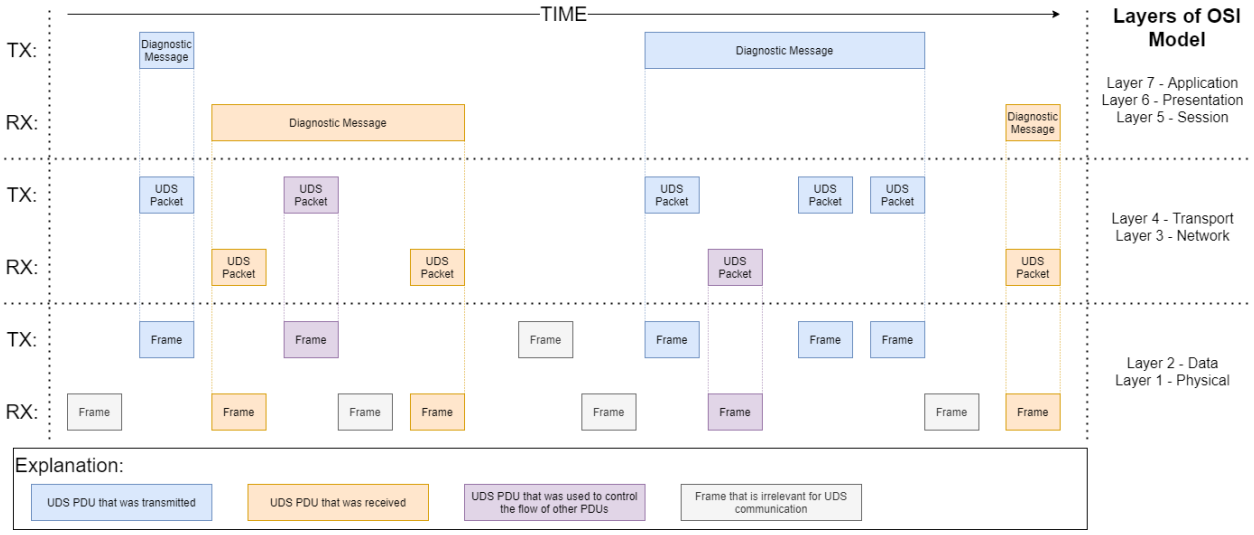


Fig. 1: UDS Protocol Data Units on different layers of OSI Model.

8.2 Diagnostic Message

Messages that are exchanged by clients and servers during UDS communications are usually called *diagnostic messages*. In the documentation and the implementation, *UDS message* name is also in use.

We distinguish two types of diagnostic messages depending on who is a transmitter:

- *diagnostic request*
- *diagnostic response*

UDS communication is always initiated by a client who sends a *diagnostic request* to a network that it has direct connection with. The client might not be directly connected to a desired recipient(s) of the request, therefore some servers might be forced to act as gateways and transmit the request to another sub-network(s). Servers' decision (whether to redirect a message to another sub-network) depends on a target(s) of the request i.e. server shall transmit the request to the sub-network if this is a route (not necessarily a direct one) to at least one recipient of the message.

Each server which was the recipient of the request, might decide to send a response back to the nearest client (the one which previously transmitted the request in this sub-network). Then, the client shall act as a gateway again and redirect the response back until it reaches the request message originator (Diagnostic Tester).

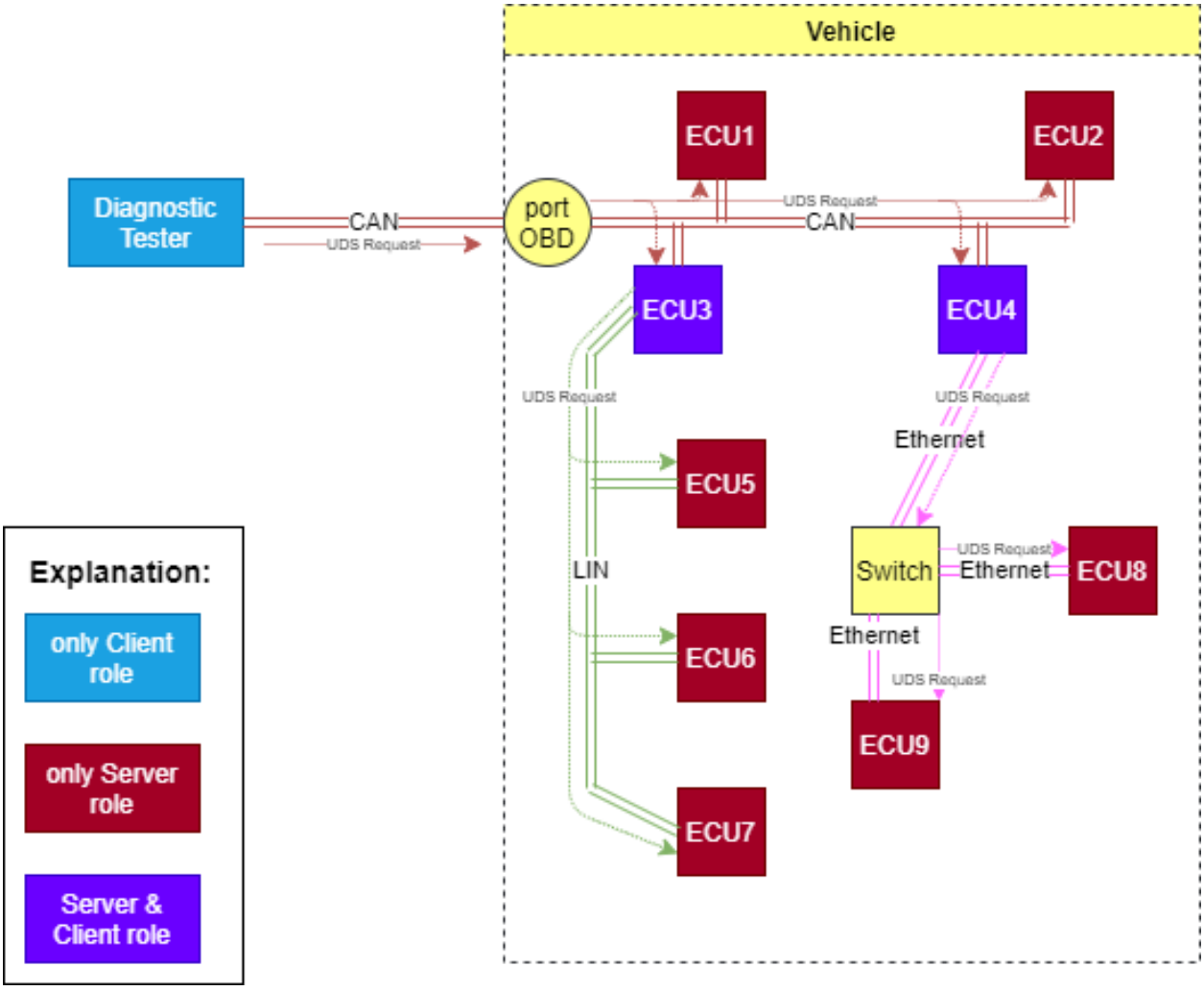


Fig. 2: Diagnostic request routing in example vehicle networks.
In this example all ECUs in the vehicle are the targets of the request - functionally addressed request was sent.

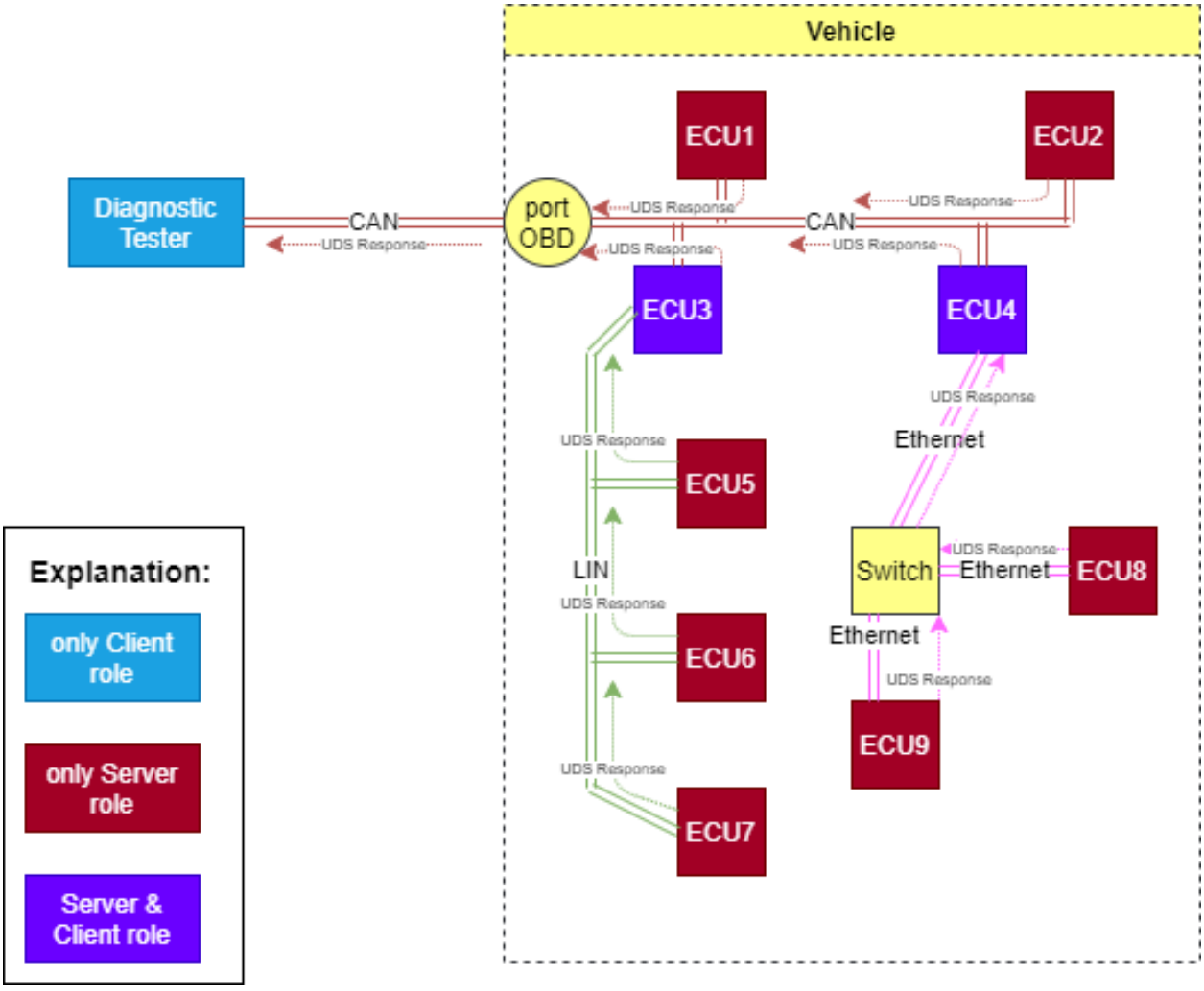


Fig. 3: Diagnostic responses routing in example vehicle networks.
In this example all ECUs in the vehicle responds to the request.

8.2.1 Diagnostic Request

Diagnostic request is a *diagnostic message* that was transmitted by a client and targets a server or group of servers. Diagnostic request can be identified by its *Service Identifier* (SID) value.

8.2.2 Diagnostic Response

Diagnostic response is a *diagnostic message* that was transmitted by a server and targets a client. Diagnostic response can be identified by its *Service Identifier* (SID) value.

UDS defines two formats of diagnostic responses:

- *positive response message*
- *negative response message*

Positive Response Message

If a server responds with a positive response message, it means that the server received the corresponding request message and executed actions requested by a client.

Format of positive response messages:

Byte	Description	Value
1	Response SID	SID + 0x40
2	data-parameter#1	XX
…	…	…
n	data-parameter#n	XX

Where:

- SID - *Service Identifier* value that was received in the request message to which the server responded
- XX - any byte value

Note: When positive diagnostic message is received, this equation is always true:

```
RSID = SID + 0x40
```

Negative Response Message

If a server responds with a negative response message, it means that (for some reason) the server could not execute actions requested by a client.

Format of negative response messages:

Byte	Description	Value
1	Negative Response SID	0x7F
2	Request SID	SID
3	NRC	XX

Where:

- SID - *Service Identifier* value that was received in the request message to which the server responded
- NRC - *Negative Response Code* value that identified the reason for negative response

8.2.3 Service Identifier

Service Identifier (SID) is data parameter that is always located in the first Application Data (A_Data) byte of each *diagnostic message* . SID value determines whether the message is *diagnostic request* or *diagnostic response*. General purpose (application) and format of *diagnostic message* is also by determined by SID value.

List of all Service Identifier (SID) values and their application:

- 0x00 - not applicable, reserved by ISO 14229-1
- 0x01-0x0F - ISO 15031-5/SAE J1979 specific services
- 0x10 - *DiagnosticSessionControl* service request
- 0x11 - *ECUReset* service request
- 0x12-0x13 - reserved by ISO 14229-1
- 0x14 - *ClearDiagnosticInformation* service request
- 0x15-0x18 - reserved by ISO 14229-1
- 0x19 - *ReadDTCInformation* service request
- 0x1A-0x21 - reserved by ISO 14229-1
- 0x22 - *ReadDataByIdentifier* service request
- 0x23 - *ReadMemoryByAddress* service request
- 0x24 - *ReadScalingDataByIdentifier* service request
- 0x25-0x26 - reserved by ISO 14229-1
- 0x27 - *SecurityAccess* service request
- 0x28 - *CommunicationControl* service request
- 0x29 - *Authentication* service request
- 0x2A - *ReadDataByPeriodicIdentifier* service request
- 0x2B - reserved by ISO 14229-1
- 0x2C - *DynamicallyDefineDataIdentifier* service request
- 0x2D - reserved by ISO 14229-1
- 0x2E - *WriteDataByIdentifier* service request
- 0x2F - *InputOutputControlByIdentifier* service request
- 0x30 - reserved by ISO 14229-1
- 0x31 - *RoutineControl* service request
- 0x32-0x33 - reserved by ISO 14229-1
- 0x34 - *RequestDownload* service request
- 0x35 - *RequestUpload* service request
- 0x36 - *TransferData* service request
- 0x37 - *RequestTransferExit* service request
- 0x38 - *RequestFileTransfer* service request
- 0x39-0x3C - reserved by ISO 14229-1

- 0x3D - *WriteMemoryByAddress* service request
- 0x3E - *TesterPresent* service request
- 0x3F - not applicable, reserved by ISO 14229-1
- 0x40 - not applicable, reserved by ISO 14229-1
- 0x41-0x4F - ISO 15031-5/SAE J1979 specific services
- 0x50 - positive response to *DiagnosticSessionControl* service
- 0x51 - positive response to *ECUReset* service
- 0x52-0x53 - reserved by ISO 14229-1
- 0x54 - positive response to *ClearDiagnosticInformation* service
- 0x55-0x58 - reserved by ISO 14229-1
- 0x59 - positive response to *ReadDTCInformation* service
- 0x5A-0x61 - reserved by ISO 14229-1
- 0x62 - positive response to *ReadDataByIdentifier* service
- 0x63 - positive response to *ReadMemoryByAddress* service
- 0x64 - positive response to *ReadScalingDataByIdentifier* service
- 0x65-0x66 - reserved by ISO 14229-1
- 0x67 - positive response to *SecurityAccess* service
- 0x68 - positive response to *CommunicationControl* service
- 0x69 - positive response to *Authentication* service
- 0x6A - positive response to *ReadDataByPeriodicIdentifier* service
- 0x6B - reserved by ISO 14229-1
- 0x6C - positive response to *DynamicallyDefineDataIdentifier* service
- 0x6D - reserved by ISO 14229-1
- 0x6E - positive response to *WriteDataByIdentifier* service
- 0x6F - positive response to *InputOutputControlByIdentifier* service
- 0x70 - reserved by ISO 14229-1
- 0x71 - positive response to *RoutineControl* service
- 0x72-0x73 - reserved by ISO 14229-1
- 0x74 - positive response to *RequestDownload* service
- 0x75 - positive response to *RequestUpload* service
- 0x76 - positive response to *TransferData* service
- 0x77 - positive response to *RequestTransferExit* service
- 0x78 - positive response to *RequestFileTransfer* service
- 0x79-0x7C - reserved by ISO 14229-1
- 0x7D - positive response to *WriteMemoryByAddress* service
- 0x7E - positive response to *TesterPresent* service

- 0x7F - negative response service identifier
- 0x80-0x82 - not applicable, reserved by ISO 14229-1
- 0x83 - reserved by ISO 14229-1
- 0x84 - *SecuredDataTransmission* service request
- 0x85 - *ControlDTCSetting* service request
- 0x86 - *ResponseOnEvent* service request
- 0x87 - *LinkControl* service request
- 0x88 - reserved by ISO 14229-1
- 0x89-0xB9 - not applicable, reserved by ISO 14229-1
- 0xBA-0xBE - system supplier specific service requests
- 0xBF-0xC2 - not applicable, reserved by ISO 14229-1
- 0xC3 - reserved by ISO 14229-1
- 0xC4 - positive response to *SecuredDataTransmission* service
- 0xC5 - positive response to *ControlDTCSetting* service
- 0xC6 - positive response to *ResponseOnEvent* service
- 0xC7 - positive response to *LinkControl* service
- 0xC8 - reserved by ISO 14229-1
- 0xC9-0xF9 - not applicable, reserved by ISO 14229-1
- 0xFA-0xFE - positive responses to system supplier specific requests
- 0xFF - not applicable, reserved by ISO 14229-1

DiagnosticSessionControl

DiagnosticSessionControl service is used to change diagnostic sessions in the server(s). In each diagnostic session a different set of diagnostic services (and/or functionalities) is enabled in the server. Server shall always be in exactly one diagnostic session.

ECUReset

ECUReset service is used by the client to request a server reset.

ClearDiagnosticInformation

ClearDiagnosticInformation service is used by the client to clear all diagnostic information (DTC and related data) in one or multiple servers' memory.

ReadDTCInformation

ReadDTCInformation service allows the client to read from any server or group of servers within a vehicle, current information about all Diagnostic Trouble Codes. This could be a status of reported Diagnostic Trouble Code (DTC), number of currently active DTCs or any other information returned by supported ReadDTCInformation SubFunctions.

ReadDataByIdentifier

ReadDataByIdentifier service allows the client to request data record values from the server identifier by one or more DataIdentifiers (DIDs).

ReadMemoryByAddress

ReadMemoryByAddress service allows the client to request server's memory data stored under provided memory address.

ReadScalingDataByIdentifier

ReadScalingDataByIdentifier service allows the client to request from the server a scaling data record identified by a DataIdentifier (DID). The scaling data contains information such as data record type (e.g. ASCII, signed float), formula and its coefficients used for value calculation, units, etc.

SecurityAccess

SecurityAccess service allows the client to unlock functions/services with restricted access.

CommunicationControl

CommunicationControl service allows the client to switch on/off the transmission and/or the reception of certain messages on a server(s).

Authentication

Authentication service provides a means for the client to prove its identity, allowing it to access data and/or diagnostic services, which have restricted access for, for example security, emissions, or safety reasons.

ReadDataByPeriodicIdentifier

ReadDataByPeriodicIdentifier service allows the client to request the periodic transmission of data record values from the server identified by one or more periodicDataIdentifiers.

DynamicallyDefineDataIdentifier

DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a DataIdentifier (DID) that can be read via the *ReadDataByIdentifier* service at a later time.

WriteDataByIdentifier

WriteDataByIdentifier service allows the client to write information into the server at an internal location specified by the provided DataIdentifier (DID).

InputOutputControlByIdentifier

InputOutputControlByIdentifier service allows the client to substitute a value for an input signal, internal server function and/or force control to a value for an output (actuator) of an electronic system.

RoutineControl

RoutineControl service allows the client to execute a defined sequence of steps to obtain any relevant result. There is a lot of flexibility with this service, but typical usage may include functionality such as erasing memory, resetting or learning adaptive data, running a self-test, overriding the normal server control strategy.

RequestDownload

RequestDownload service allows the client to initiate a data transfer from the client to the server (download).

RequestUpload

RequestUpload service allows the client to initiate a data transfer from the server to the client (upload).

TransferData

TransferData service is used by the client to transfer data either from the client to the server (download) or from the server to the client (upload).

RequestTransferExit

RequestTransferExit service is used by the client to terminate a data transfer between the client and server.

RequestFileTransfer

RequestFileTransfer service allows the client to initiate a file data transfer either from the server to the client (upload) or from the server to the client (upload).

WriteMemoryByAddress

WriteMemoryByAddress service allows the client to write information into server's memory data under provided memory address.

TesterPresent

TesterPresent service is used by the client to indicate to a server(s) that the client is still connected to a vehicle and certain diagnostic services and/or communication that have been previously activated are to remain active.

SecuredDataTransmission

SecuredDataTransmission service is applicable if a client intends to use diagnostic services defined in this document in a secured mode. It may also be used to transmit external data, which conform to some other application protocol, in a secured mode between a client and a server. A secured mode in this context means that the data transmitted is protected by cryptographic methods.

ControlDTCSetting

ControlDTCSetting service allows the client to stop or resume the updating of DTC status bits in the server(s) memory.

ResponseOnEvent

ResponseOnEvent service allows the client to request from the server to start ot stop transmission of responses on a specified event.

LinkControl

LinkControl service allows the client to control the communication between the client and the server(s) in order to gain bus bandwidth for diagnostic purposes (e.g. programming).

8.2.4 Negative Response Code

Negative Response Code (NRC) is one byte value which contains information why a server is not sending a positive response message.

List of NRC values:

- 0x00 - positiveResponse - This NRC shall not be used in a negative response message. This positiveResponse parameter value is reserved for server internal implementation.
- 0x00-0x0F - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x10 - generalReject - This NRC indicates that the requested action has been rejected by the server.
- 0x11 - serviceNotSupported - This NRC indicates that the requested action will not be taken because the server does not support the requested service.
- 0x12 - SubFunctionNotSupported - This NRC indicates that the requested action will not be taken because the server does not support the service specific parameters of the request message.

- 0x13 - incorrectMessageLengthOrInvalidFormat - This NRC indicates that the requested action will not be taken because the length of the received request message does not match the prescribed length for the specified service or the format of the parameters do not match the prescribed format for the specified service.
- 0x14 - responseTooLong - This NRC shall be reported by the server if the response to be generated exceeds the maximum number of bytes available by the underlying network layer. This could occur if the response message exceeds the maximum size allowed by the underlying transport protocol or if the response message exceeds the server buffer size allocated for that purpose.
- 0x15-0x20 - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x21 - busyRepeatRequest - This NRC indicates that the server is temporarily too busy to perform the requested operation. In this circumstance the client shall perform repetition of the "identical request message" or "another request message". The repetition of the request shall be delayed by a time specified in the respective implementation documents.
- 0x22 - conditionsNotCorrect - This NRC indicates that the requested action will not be taken because the server prerequisite conditions are not met.
- 0x23 - ISO Reserved - This value is reserved for future definition by ISO 14229 Standard.
- 0x24 - requestSequenceError - This NRC indicates that the requested action will not be taken because the server expects a different sequence of request messages or message as sent by the client. This may occur when sequence sensitive requests are issued in the wrong order.
- 0x25 - noResponseFromSubnetComponent - This NRC indicates that the server has received the request but the requested action could not be performed by the server as a subnet component which is necessary to supply the requested information did not respond within the specified time.
- 0x26 - FailurePreventsExecutionOfRequestedAction - This NRC indicates that the requested action will not be taken because a failure condition, identified by a DTC (with at least one DTC status bit for TestFailed, Pending, Confirmed or TestFailedSinceLastClear set to 1), has occurred and that this failure condition prevents the server from performing the requested action.
- 0x27-0x30 - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x31 - requestOutOfRange - This NRC indicates that the requested action will not be taken because the server has detected that the request message contains a parameter which attempts to substitute a value beyond its range of authority (e.g. attempting to substitute a data byte of 111 when the data is only defined to 100), or which attempts to access a DataIdentifier/RoutineIdentifer that is not supported or not supported in active session.
- 0x32 - ISO Reserved - This value is reserved for future definition by ISO 14229 Standard.
- 0x33 - securityAccessDenied - This NRC indicates that the requested action will not be taken because the server's security strategy has not been satisfied by the client.
- 0x34 - authenticationRequired - This NRC indicates that the requested service will not be taken because the client has insufficient rights based on its Authentication state.
- 0x35 - invalidKey - This NRC indicates that the server has not given security access because the key sent by the client did not match with the key in the server's memory. This counts as an attempt to gain security.
- 0x36 - exceedNumberOfAttempts - This NRC indicates that the requested action will not be taken because the client has unsuccessfully attempted to gain security access more times than the server's security strategy will allow.
- 0x37 - requiredTimeDelayNotExpired - This NRC indicates that the requested action will not be taken because the client's latest attempt to gain security access was initiated before the server's required timeout period had elapsed.

- 0x38 - secureDataTransmissionRequired - This NRC indicates that the requested service will not be taken because the requested action is required to be sent using a secured communication channel.
- 0x39 - secureDataTransmissionNotAllowed - This NRC indicates that this message was received using the SecuredDataTransmission (SID 0x84) service. However, the requested action is not allowed to be sent using the SecuredDataTransmission (0x84) service.
- 0x3A - secureDataVerificationFailed - This NRC indicates that the message failed in the security sub-layer.
- 0x3B-0x4F - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x50 - Certificate verification failed, Invalid Time Period - Date and time of the server does not match the validity period of the Certificate.
- 0x51 - Certificate verification failed, Invalid Signature - Signature of the Certificate could not be verified.
- 0x52 - Certificate verification failed, Invalid Chain of Trust - Certificate could not be verified against stored information about the issuing authority.
- 0x53 - Certificate verification failed, Invalid Type - Certificate does not match the current requested use case.
- 0x54 - Certificate verification failed, Invalid Format - Certificate could not be evaluated because the format requirement has not been met.
- 0x55 - Certificate verification failed, Invalid Content - Certificate could not be verified because the content does not match.
- 0x56 - Certificate verification failed, Invalid Scope - The scope of the Certificate does not match the contents of the server.
- 0x57 - Certificate verification failed, Invalid Certificate (revoked) - Certificate received from client is invalid, because the server has revoked access for some reason.
- 0x58 - Ownership verification failed - Delivered Ownership does not match the provided challenge or could not verified with the own private key.
- 0x59 - Challenge calculation failed - The challenge could not be calculated on the server side.
- 0x5A - Setting Access Rights failed - The server could not set the access rights.
- 0x5B - Session key creation/derivation failed - The server could not create or derive a session key.
- 0x5C - Configuration data usage failed - The server could not work with the provided configuration data.
- 0x5D - DeAuthentication failed - DeAuthentication was not successful, server could still be unprotected.
- 0x5E-0x6F - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x70 - uploadDownloadNotAccepted - This NRC indicates that an attempt to upload/download to a server's memory cannot be accomplished due to some fault conditions.
- 0x71 - transferDataSuspended - This NRC indicates that a data transfer operation was halted due to some fault. The active transferData sequence shall be aborted.
- 0x72 - generalProgrammingFailure - This NRC indicates that the server detected an error when erasing or programming a memory location in the permanent memory device (e.g. Flash Memory).
- 0x73 - wrongBlockSequenceCounter - This NRC indicates that the server detected an error in the sequence of blockSequenceCounter values. Note that the repetition of a TransferData request message with a blockSequenceCounter equal to the one included in the previous TransferData request message shall be accepted by the server.
- 0x74-0x77 - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.

- 0x78 - requestCorrectlyReceived-ResponsePending - This NRC indicates that the request message was received correctly, and that all parameters in the request message were valid (these checks can be delayed until after sending this NRC if executing the boot software), but the action to be performed is not yet completed and the server is not yet ready to receive another request. As soon as the requested service has been completed, the server shall send a positive response message or negative response message with a response code different from this.
- 0x79-0x7D - ISO Reserved - This range of values is reserved for future definition by ISO 14229 Standard.
- 0x7E - SubFunctionNotSupportedInActiveSession - This NRC indicates that the requested action will not be taken because the server does not support the requested SubFunction in the session currently active. This NRC shall only be used when the requested SubFunction is known to be supported in another session, otherwise response code SubFunctionNotSupported shall be used.
- 0x7F - serviceNotSupportedInActiveSession - This NRC indicates that the requested action will not be taken because the server does not support the requested service in the session currently active. This NRC shall only be used when the requested service is known to be supported in another session, otherwise response code serviceNotSupported shall be used.
- 0x80 - ISO Reserved - This value is reserved for future definition by ISO 14229 Standard.
- 0x81 - rpmTooHigh - This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is above a preprogrammed maximum threshold).
- 0x82 - rpmTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for RPM is not met (current RPM is below a preprogrammed minimum threshold).
- 0x83 - engineIsRunning - This NRC is required for those actuator tests which cannot be actuated while the Engine is running. This is different from RPM too high negative response, and shall be allowed.
- 0x84 - engineIsNotRunning - This NRC is required for those actuator tests which cannot be actuated unless the Engine is running. This is different from RPM too low negative response, and shall be allowed.
- 0x85 - engineRunTimeTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for engine run time is not met (current engine run time is below a preprogrammed limit).
- 0x86 - temperatureTooHigh - This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is above a preprogrammed maximum threshold).
- 0x87 - temperatureTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for temperature is not met (current temperature is below a preprogrammed minimum threshold).
- 0x88 - vehicleSpeedTooHigh - This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is above a preprogrammed maximum threshold).
- 0x89 - vehicleSpeedTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for vehicle speed is not met (current VS is below a preprogrammed minimum threshold).
- 0x8A - throttle/PedalTooHigh - This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current throttle/pedal position is above a preprogrammed maximum threshold).
- 0x8B - throttle/PedalTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for throttle/pedal position is not met (current throttle/pedal position is below a preprogrammed minimum threshold).

- 0x8C - transmissionRangeNotInNeutral - This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in neutral is not met (current transmission range is not in neutral).
- 0x8D - transmissionRangeNotInGear - This NRC indicates that the requested action will not be taken because the server prerequisite condition for being in gear is not met (current transmission range is not in gear).
- 0x8E - ISO Reserved - This value is reserved for future definition by ISO 14229 Standard.
- 0x8F - brakeSwitch(es)NotClosed (Brake Pedal not pressed or not applied) - This NRC indicates that for safety reasons, this is required for certain tests before it begins, and shall be maintained for the entire duration of the test.
- 0x90 - shifterLeverNotInPark - This NRC indicates that for safety reasons, this is required for certain tests before it begins, and shall be maintained for the entire duration of the test.
- 0x91 - torqueConverterClutchLocked - This NRC indicates that the requested action will not be taken because the server prerequisite condition for torque converter clutch is not met (current torque converter clutch status above a preprogrammed limit or locked).
- 0x92 - voltageTooHigh - This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is above a preprogrammed maximum threshold).
- 0x93 - voltageTooLow - This NRC indicates that the requested action will not be taken because the server prerequisite condition for voltage at the primary pin of the server (ECU) is not met (current voltage is below a preprogrammed minimum threshold).
- 0x94 - ResourceTemporarilyNotAvailable - This NRC indicates that the server has received the request but the requested action could not be performed by the server because an application which is necessary to supply the requested information is temporality not available. This NRC is in general supported by each diagnostic service, as not otherwise stated in the data link specific implementation document, therefore it is not listed in the list of applicable response codes of the diagnostic services.
- 0x95-0xEF - reservedForSpecificConditionsNotCorrect - This range of values is reserved for future definition condition not correct scenarios by ISO 14229 Standard.
- 0xF0-0xFE - vehicleManufacturerSpecificConditionsNotCorrect - This range of values is reserved for vehicle manufacturer specific condition not correct scenarios.
- 0xFF - ISO Reserved - This value is reserved for future definition by ISO 14229 Standard.

8.2.5 Addressing

Addressing determines model of UDS communication.

We distinguish following addressing types:

- *Physical*
- *Functional*

Physical

Physical addressing is used to send a dedicated message to a certain server (ECU). When physically addressed messages are sent, the direct (point-to-point) communication between the client and the server takes place. The server shall respond to a physically addressed request unless the request contains an information that a response is not required (further explained in *response behaviour to physically addressed request* chapter).

NOTE: You do not need a direct physical connection between a client and a server to have physically addressed communication as all messages shall be routed to a target of each message.

Response behaviour to physically addressed request

Expected server behaviour in case of receiving physically addressed request message with SubFunction parameter:

Client request		**Server capability**			**Server response**		**Comment**
Addressing	**SPRMIB**	**SID supported**	**SF supported**	**DataParam supported**	**Message**	**NRC**	
physical	False (bit = 0)	YES	YES	At least 1	Positive Response	—	Server supports the requests and sends positive response.
				At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
				None		NRC = ROOR	Servers sends negative response with NRC 0x31.
		NO	—	—		NRC = SNS or SNSIAS	Servers sends negative response with NRC 0x11 or 0x7F.
		YES	NO	—		NRC = SFNS or SFNSIAS	Servers sends negative response with NRC 0x12 or 0x7E.
	True (bit = 1)	YES	YES	At least 1	No Response	—	Server does not send a response.
				At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
				None		NRC = ROOR	Servers sends negative response with NRC 0x31.
		NO	—	—		NRC = SNS or SNSIAS	Servers sends negative response with NRC 0x11 or 0x7F.
		YES	NO	—		NRC = SFNS or SFNSIAS	Servers sends negative response with NRC 0x12 or 0x7E.

Expected server behaviour in case of receiving physically addressed request message without SubFunction parameter:

Client request	Server capability		Server response		Comment
Addressing	**SID supported**	**DataParam supported**	**Message**	**NRC**	
physical	YES	All	Positive Response	—	Server supports the requests and sends positive response.
		At least 1		—	Server supports the requests and sends positive response.
		At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
		None		NRC = ROOR	Servers sends negative response with NRC 0x31.
	NO	—		NRC = SNS or SNSIAS	Servers sends negative response with NRC 0x11 or 0x7F

Where:

- SPRMIB - flag informing whether Suppress Positive Response Message Indication Bit is set in the received request message
- SID supported - flag informing whether Service Identifier in the received request message is supported by the server
- SF supported - flag informing whether SubFunction in the received request message is supported by the server
- DataParam supported - information whether values of data parameters (e.g. DIDs, RIDs, DTCStatusMask) in the received request message are supported by the server
- NRC - Negative Response Code
- ROOR - NRC 0x31 (requestOutOfRange)
- SNS - NRC 0x11 (serviceNotSupported)
- SNSIAS - NRC 0x7F (serviceNotSupportedInActiveSession)
- SFNS - NRC 0x12 (SubFunctionNotSupported)
- SFNSIAS - NRC 0x7E (SubFunctionNotSupportedInActiveSession)
- XX - NRC code that is supported by the server and suitable to the current situation (e.g. NRC 0x21 busyRepeatRequest if server is currently overloaded and cannot process next request message)

Functional

Functional addressing is used to send messages to multiple servers (ECUs) in the network. When functionally addressed messages are sent, a one to many communication between a client and servers (ECUs) takes place. A server shall only respond to certain functionally addressed requests (further explained in *response behaviour to functionally addressed request* chapter.

NOTE: Some types of buses (e.g. LIN) might also support broadcast communication which slightly change expected server behaviour. When broadcast communication is used, then a server response is never expected by a client.

Response behaviour to functionally addressed request

Expected server behaviour in case of receiving functionally addressed request message with SubFunction parameter:

Client request		Server capability			Server response		Comment
Addressing	SPRMIB	SID supported	SF supported	DataParam supported	Message	NRC	
functional	False (bit = 0)	YES	YES	At least 1	Positive Response	—	Server supports the requests and sends positive response.
				At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
				None	No Response	—	Server does not send a response.
		NO	—	—		—	Server does not send a response.
		YES	NO	—		—	Server does not send a response.
	True (bit = 1)	YES	YES	At least 1	No Response	—	Server does not send a response.
				At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
				None	No Response	—	Server does not send a response.
		NO	—	—		—	Server does not send a response.
		YES	NO	—		—	Server does not send a response.

Expected server behaviour in case of receiving functionally addressed request message without SubFunction parameter:

Client request	Server capability		Server response		Comment
Addressing	SID supported	DataParam supported	Message	NRC	
functional	YES	All	Positive Response	—	Server supports the requests and sends positive response.
		At least 1		—	Server supports the requests and sends positive response.
		At least 1	Negative Response	NRC = XX	Server sends negative response because an error occurred processing the data parameters of request message.
		None	No Response	—	Server does not send a response.
	NO	—		—	Server does not send a response.

Where:

- SPRMIB - flag informing whether Suppress Positive Response Message Indication Bit is set in the received request message
- SID supported - flag informing whether Service Identifier in the received request message is supported by the server

- SF supported - flag informing whether SubFunction in the received request message is supported by the server
- DataParam supported - information whether values of data parameters (e.g. DIDs, RIDs, DTCStatusMask) in the received request message are supported by the server
- NRC - Negative Response Code
- XX - NRC code that is supported by the server and suitable to the current situation (e.g. NRC 0x21 busyRepeatRequest if server is currently overloaded and cannot process next request message)

8.3 UDS Packet

UDS packet might also be called Network Protocol Data Unit (N_PDU). The packets are created during *segmentation* of a *diagnostic message*. Each *diagnostic message* consists of at least one UDS Packet (N_PDU). There are some packets which does not carry any diagnostic message data as they are used to manage the flow of other packets.

UDS packet consists of following fields:

- *Network Address Information* (N_AI) - packet addressing
- *Network Protocol Control Information* (N_PCI) - packet type
- *Network Data Field* (N_Data) - packet date

8.3.1 Network Address Information

Network Address Information (N_AI) contains address information which identifies the recipient(s) and the sender between whom data exchange takes place. It also describes communication model (e.g. whether response is required) for the message.

8.3.2 Network Protocol Control Information

Network Protocol Control Information (N_PCI) identifies the type of *UDS packet* (Network Protocol Data Unit). N_PCI values and their interpretation are bus specific.

8.3.3 Network Data Field

Network Data Field (N_Data) carries diagnostic message data. It might be an entire diagnostic message data (if a *diagnostic message* fits into one packet) or just a part of it (if *segmentation* had to be used to divide a *diagnostic message* into smaller parts).

8.4 Segmentation

8.4.1 Message Segmentation

If diagnostic message data to be transmitted does not fit into a single frame, then segmentation process is required to divide *diagnostic message* into smaller pieces called *UDS packets*.

8.4.2 Packets Desegmentation

Desegmentation is a reverse process to a *message segmentation*. It transforms one or more *UDS packets* into a *diagnostic message*.

CHAPTER

NINE

CONTRIBUTION

9.1 How to contribute?

If you want to become a contributor, please visit UDS project page and read CONTRIBUTING.md file.

Contact us to find out what we have got to offer and to get more information.

9.2 Sponsoring

If you consider sponsoring our development team, please visit our wiki page with detailed information for sponsors. With a little help from you, we would be able to improve the quality of our code, speed up the development and provide more features. We also offer special treatment for all our sponsors. Please contact us for more details.

9.3 Reporting issues

To report issues, please use our issues tracking system.

9.4 Our Sponsors

Full list of our sponsors:

- Merit Automotive 

 - sponsoring since September 2021

CHAPTER

TEN

OVERVIEW

The purpose of this project is to provide python tools for simulation (on both sides - client and server) and monitoring of diagnostic communication defined by ISO-14229. It can be used with any bus type (e.g. CAN, Ethernet, LIN).

The most likely use cases of this package are:

- communication with your vehicle (e.g. reading Diagnostic Trouble Codes)
- monitoring and decoding ongoing UDS communication
- performing tests against on-board ECU (server)
- performing tests against OBD Tester (client)

CHAPTER

ELEVEN

IMPLEMENTATION STATUS

The package is currently in the early development phase, therefore only a few features are currently available. If you want to speed up the development, please visit *contribution section* to find out what are your options.

11.1 Features

Current implementation status of package features:

Feature	Implementation Status
UDS Messages and Packets	Implemented in version 0.0.2
UDS Packets Reception	Ongoing
UDS Packets Transmission	Planned
Segmentation	Ongoing
Support for Services with multiple responses	Planned
Client Simulation	Ongoing
Server Simulation	Planned
Support for Messages Databases	Planned

11.2 Buses supported

Current implementation status of support for communication buses:

Bus	Implementation Status
CAN	Ongoing
FlexRay	Planned
Ethernet	Planned
K-Line	Planned
LIN	Planned

CHAPTER

TWELVE

LICENSE

The project is licensed under the MIT license - https://github.com/mdabrowski1990/uds/blob/main/LICENSE

CHAPTER

THIRTEEN

CONTACT

- e-mail: uds-package-development@googlegroups.com
- group: UDS package development
- discord: https://discord.gg/y3waVmR5PZ

Documentation generated

Oct 02, 2021

PYTHON MODULE INDEX

u

INDEX

Symbols

A

B

C

D

O

P

R

S

T

U

V

W