

# User Manual



# Relevant Product

Product Name	Version
PCAN-OBDonUDS API	1.0.0

## Imprint

PCAN and PLIN are registered trademarks of PEAK-System Technik GmbH.

All other product names in this document may be the trademarks or registered trademarks of their respective companies. They are not explicitly marked by ™ or ®.

© 2024 PEAK-System Technik GmbH

Duplication (copying, printing, or other forms) and the electronic distribution of this document is only allowed with explicit permission of PEAK-System Technik GmbH. PEAK-System Technik GmbH reserves the right to change technical data without prior announcement. The general business conditions and the regulations of the license agreement apply. All rights are reserved.

PEAK-System Technik GmbH  
Darmstadt, Germany

Phone: +49 6151 8173-20  
Fax: +49 6151 8173-29

[www.peak-system.com](http://www.peak-system.com)  
[info@peak-system.com](mailto:info@peak-system.com)

Document version 1.0.0 (2024-07-31)

# Contents

<b>Imprint</b>	<b>2</b>
<b>Relevant Product</b>	<b>2</b>
<b>Contents</b>	<b>3</b>
<b>1 Disclaimer</b>	<b>7</b>
<b>2 Introduction</b>	<b>8</b>
2.1 Understanding PCAN-OBDonUDS API	8
2.2 Using PCAN-OBDonUDS API	10
2.3 Features	11
2.4 System Requirements	11
2.5 Scope of Supply	11
<b>3 DLL API Reference</b>	<b>12</b>
3.1 Namespaces	12
3.1.1 Peak.Can.OBDonUDS	12
3.2 Units	13
3.3 Classes	14
3.3.1 OBDonUDSApi	14
3.4 Structures	15
3.4.1 obd_netaddrinfo	16
3.4.2 obd_msgaccess	17
3.4.3 obd_msg	18
3.4.4 obd_response_generic	18
3.4.5 obd_did_object	19
3.4.6 obd_request_current_data_response	20
3.4.7 obd_request_freeze_frame_data_response	21
3.4.8 obd_severity_trouble_code	22
3.4.9 obd_request_trouble_codes_response	23
3.4.10 obd_request_clear_trouble_codes_response	24
3.4.11 obd_test_data_object	25
3.4.12 obd_request_test_results_response	26
3.4.13 obd_request_control_operation_response	27
3.4.14 obd_request_vehicle_information_response	28
3.4.15 obd_trouble_code	29
3.4.16 obd_request_permanent_trouble_codes_response	29
3.4.17 obd_request_supported_dtc_extended_response	31
3.4.18 obd_request_dtc_extended_response	32
3.4.19 obd_request_dtc_for_a_readiness_group_response	33
3.5 Types	34
3.5.1 obd_ecu	34
3.5.2 obd_service	35
3.5.3 obd_sub_function	36
3.5.4 obd_errstatus	36
3.5.5 obd_status	37
3.5.6 obd_parameter	39
3.5.7 obd_baudrate	45
3.5.8 obd_msgprotocol	45

3.5.9	obd_addressing .....	46
3.5.10	obd_datatype .....	47
3.5.11	obd_DID_t .....	48
3.5.12	obd_DTC_t .....	48
3.5.13	obd_RID_t .....	49
3.6	PCAN-UDS and PCAN-ISO-TP Dependencies .....	50
3.7	Methods .....	51
3.7.1	Initialize .....	52
3.7.2	Initialize(cantp_handle channel, cantp_baudrate baudrate, cantp_hwtype hw_type, UInt32 io_ port, UInt16 interrupt) .....	53
3.7.3	Initialize(cantp_handle channel, cantp_baudrate baudrate) .....	54
3.7.4	Initialize(cantp_handle channel) .....	56
3.7.5	Uninitialize .....	57
3.7.6	SetValue .....	58
3.7.7	SetValue(cantp_handle channel, obd_parameter parameter, IntPtr buffer, UInt32 buffer_size) .....	59
3.7.8	SetValue(cantp_handle channel, obd_parameter parameter, ref UInt32 buffer, UInt32 buffer_size) .....	60
3.7.9	SetValue(cantp_handle channel, obd_parameter parameter, String buffer, UInt32 buffer_size) .....	61
3.7.10	SetValue(cantp_handle channel, obd_parameter parameter, byte[] buffer, UInt32 buffer_size) .....	62
3.7.11	GetValue .....	63
3.7.12	GetValue( cantp_handle channel, obd_parameter parameter, IntPtr buffer, UInt32 buffer_size) .....	64
3.7.13	GetValue( cantp_handle channel, obd_parameter parameter, StringBuilder buffer, UInt32 buffer_size) .....	65
3.7.14	GetValue( cantp_handle channel, obd_parameter parameter, out UInt32 buffer, UInt32 buffer_size) .....	66
3.7.15	GetValue( cantp_handle channel, obd_parameter parameter, obd_baudrate buffer, UInt32 buffer_size) .....	67
3.7.16	GetValue( cantp_handle channel, obd_parameter parameter, out obd_msgprotocol buffer, UInt32 buffer_size) .....	68
3.7.17	GetValue( cantp_handle channel, obd_parameter parameter, [Out] Byte[] buffer, UInt32 buffer_size) .....	69
3.7.18	GetStatus .....	70
3.7.19	StatusIsOk .....	71
3.7.20	StatusIsOk( obd_status status, obd_status status_expected, bool strict_mode) .....	72
3.7.21	StatusIsOk( obd_status status, obd_status status_expected) .....	73
3.7.22	StatusIsOk( obd_status status) .....	74
3.7.23	GetErrorText .....	75
3.7.24	ParseResponse_RequestCurrentData .....	76
3.7.25	ParseResponse_RequestFreezeFrameData .....	78
3.7.26	ParseResponse_RequestConfirmedTroubleCodes .....	80
3.7.27	ParseResponse_ClearTroubleCodes .....	82
3.7.28	ParseResponse_RequestTestResults .....	83
3.7.29	ParseResponse_RequestPendingTroubleCodes .....	85
3.7.30	ParseResponse_RequestControlOperation .....	87
3.7.31	ParseResponse_RequestVehicleInformation .....	89
3.7.32	ParseResponse_RequestPermanentTroubleCodes .....	90
3.7.33	ParseResponse_RequestSupportedDTCExtended .....	92
3.7.34	ParseResponse_RequestDTCExtended .....	93
3.7.35	ParseResponse_RequestDTCForAReadinessGroup .....	95
3.7.36	Reset .....	96
3.7.37	FindOBDonEDS .....	97
3.7.38	FindOBDonEDS(cantp_handle channel, cantp_baudrate baudrate, cantp_hwtype hw_type, UInt32 io_ port, UInt16 interrupt) .....	98
3.7.39	FindOBDonEDS(cantp_handle channel, cantp_baudrate baudrate) .....	99

3.7.40	FindOBDonEDS(cantp_handle channel)	100
3.7.41	RequestCurrentData	102
3.7.42	RequestFreezeFrameData	104
3.7.43	RequestConfirmedTroubleCodes	105
3.7.44	ClearTroubleCodes	107
3.7.45	RequestTestResults	108
3.7.46	RequestPendingTroubleCodes	110
3.7.47	RequestControlOperation	112
3.7.48	RequestVehicleInformation	113
3.7.49	RequestPermanentTroubleCodes	115
3.7.50	RequestSupportedDTCEXtended	116
3.7.51	RequestDTCEXtended	118
3.7.52	RequestDTCForAReadinessGroup	119
3.7.53	WaitForService	121
3.7.54	WaitForServiceFunctional	123
3.7.55	MsgFree	124
3.7.56	ParsedResponseFree	125
3.7.57	GetData	126
3.7.58	GetData(ref obd_request_vehicle_information_response parsed_response, byte[] vals, Int32 nb)	127
3.7.59	GetData(ref obd_request_current_data_response parsed_response, obd_did_object[] vals, Int32 nb)	127
3.7.60	GetData(ref obd_did_object didObject, byte[] vals, Int32 nb)	128
3.7.61	GetData(ref obd_request_trouble_codes_response parsed_response, obd_severity_trouble_code[] vals, Int32 nb)	129
3.7.62	GetData(ref obd_request_freeze_frame_data_response parsed_response, obd_did_object[] vals, Int32 nb)	129
3.7.63	GetData(ref obd_request_test_results_response parsed_response, obd_test_data_object[] vals, Int32 nb)	130
3.7.64	GetData(ref obd_request_control_operation_response parsed_response, byte[] vals, Int32 nb)	130
3.7.65	GetData(ref obd_request_permanent_trouble_codes_response parsed_response, obd_trouble_code[] vals, Int32 nb)	131
3.7.66	GetData(ref obd_request_supported_dtc_extended_response parsed_response, obd_trouble_code[] vals, Int32 nb)	131
3.7.67	GetData(ref obd_request_dtc_extended_response parsed_response, byte[] vals, Int32 nb)	132
3.7.68	GetData(ref obd_request_dtc_for_a_readiness_group_response parsed_response, obd_trouble_code[] vals, Int32 nb)	133
3.7.69	GetByte	134
3.8	Functions	135
3.8.1	OBDonUDS_Initialize	137
3.8.2	OBDonUDS_Uninitialize	138
3.8.3	OBDonUDS_SetValue	140
3.8.4	OBDonUDS_GetValue	141
3.8.5	OBDonUDS_GetStatus	142
3.8.6	OBDonUDS_StatusIsOk	144
3.8.7	OBDonUDS_GetErrorText	145
3.8.8	OBDonUDS_ParseResponse_RequestCurrentData	146
3.8.9	OBDonUDS_ParseResponse_RequestFreezeFrameData	148
3.8.10	OBDonUDS_ParseResponse_RequestConfirmedTroubleCodes	149
3.8.11	OBDonUDS_ParseResponse_ClearTroubleCodes	151
3.8.12	OBDonUDS_ParseResponse_RequestTestResults	152

3.8.13	OBDonUDS_ParseResponse_RequestPendingTroubleCodes	154
3.8.14	OBDonUDS_ParseResponse_RequestControlOperation	155
3.8.15	OBDonUDS_ParseResponse_RequestVehicleInformation	157
3.8.16	OBDonUDS_ParseResponse_RequestPermanentTroubleCodes	158
3.8.17	OBDonUDS_ParseResponse_RequestSupportedDTCExtended	160
3.8.18	OBDonUDS_ParseResponse_RequestDTCExtended	161
3.8.19	OBDonUDS_ParseResponse_RequestDTCForAReadinessGroup	163
3.8.20	OBDonUDS_Reset	165
3.8.21	OBDonUDS_FindOBDonEDS	166
3.8.22	OBDonUDS_RequestCurrentData	167
3.8.23	OBDonUDS_RequestFreezeFrameData	169
3.8.24	OBDonUDS_RequestConfirmedTroubleCodes	171
3.8.25	OBDonUDS_ClearTroubleCodes	172
3.8.26	OBDonUDS_RequestTestResults	174
3.8.27	OBDonUDS_RequestPendingTroubleCodes	176
3.8.28	OBDonUDS_RequestControlOperation	178
3.8.29	OBDonUDS_RequestVehicleInformation	180
3.8.30	OBDonUDS_RequestPermanentTroubleCodes	181
3.8.31	OBDonUDS_RequestSupportedDTCExtended	183
3.8.32	OBDonUDS_RequestDTCExtended	185
3.8.33	OBDonUDS_RequestDTCForAReadinessGroup	186
3.8.34	OBDonUDS_WaitForService	188
3.8.35	OBDonUDS_WaitForServiceFunctional	190
3.8.36	OBDonUDS_MsgFree	192
3.8.37	OBDonUDS_ParsedResponseFree	192
3.9	Definitions	194
3.9.1	Miscellaneous	194
3.9.2	POBDONUDS parameter value definitions	194
<b>4</b>	<b>Additional information</b>	<b>195</b>
4.1	PCAN Fundamentals	195
4.2	PCAN-Basic	196
4.3	OBDonUDS, UDS, and ISO-TP Network Addressing Information	197
4.3.1	UDS and ISO-TP Network Addressing Information	198

# 1 Disclaimer

The interface DLLs of this API, PCAN-Basic, device drivers, and further files needed for linking are property of the PEAK-System Technik GmbH and may be used only in connection with a hardware component purchased from PEAK-System or one of its partners.

If a CAN hardware component of third party suppliers should be compatible to one of PEAK-System, then you are not allowed to use or to pass on the APIs and driver software of PEAK-System.



**Important note:** If a third-party supplier develops software based on the PCAN-OBDonUDS API and problems occur during the use of this software, consult the software provider.

## 2 Introduction

The standard OBDOnUDS (SAE J1979-2) describes the communication between on-board diagnostic systems of vehicles and corresponding test equipment. OBDOnUDS was designed as the successor to the OBD-2 standard which has reached its limits, as on-board diagnostics have become increasingly complex.

Based on the established UDS standard (ISO 14229-1), OBDOnUDS creates structures and functionality for improved vehicle diagnostics and monitoring. For example, more DTC error codes are possible and more detailed data is captured for each error.

The free PCAN-OBDOnUDS API provides the services of the OBDOnUDS standard with its functions. Diagnostic data is exchanged via CAN by utilizing the underlying PCAN-UDS API.

### 2.1 Understanding PCAN-OBDOnUDS API

OBD stands for On-Board Diagnostics. It is a standard that specifies the type of diagnostic connector and its pinout, the electrical signaling protocols available, the messaging format and a candidate list of vehicle parameters to monitor along with information how to encode the data for each.

The OBDOnUDS communication protocol is defined in SAE J1979-2 together with ISO 14229-1 (UDS) and some specific implementation for the CAN bus is described in ISO 15765-4 (OBD). It is a Client/Server oriented protocol:

- External test equipment has the role of the client
- Electronic Control Units (ECUs) connected inside the vehicle are servers

Client always starts the communication by sending a request. If a server supports this request, it will reply with a positive or negative response, otherwise it can just ignore request.

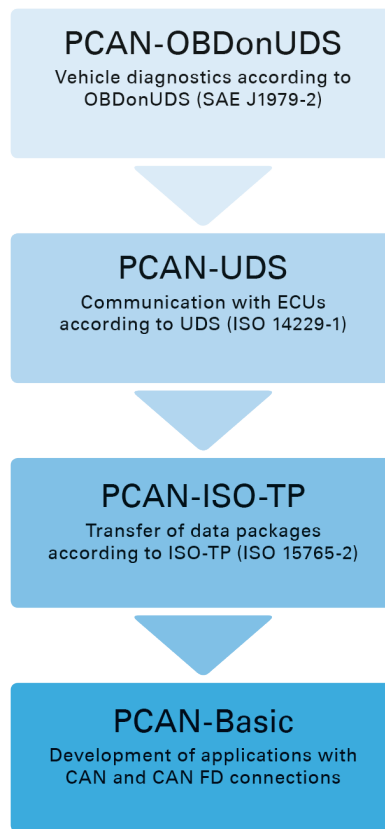
Two addressing modes are supported:

- Physical: The client sends a request to one server.
- Functional: The client sends a request without specifying the server. The client should be aware of receiving more than one response to request.

OBDOnUDS is not just a translation of the classic "OBD II" SAE J1979 \$01-\$0A services to UDS services; it provides additional features.

PCAN-OBDOnUDS API is an implementation of the OBDOnUDS on CAN standard.





The physical communication is carried out by PCAN-Hardware (PCAN-USB, PCAN-PCI etc.) through the PCAN-UDS, PCAN-ISO-TP, and PCAN-Basic APIs. Because of this it is necessary to have also the PCAN-UDS, PCAN-ISO-TP, and PCAN-Basic APIs (PCAN-UDS.dll, PCAN-ISO-TP.dll, and PCANBasic.dll) present on the working computer where the PCAN-OBDonUDS API is intended to be used.

PCAN-OBDonUDS, PCAN-UDS, PCAN-ISO-TP, and PCAN-Basic APIs are free and available for all people that acquire a PCAN-Hardware.



**Note:** Diagnostic data is exchanged via CAN by utilizing the underlying PCAN-UDS API. DoIP is not supported.

## 2.2 Using PCAN-OBDonUDS API

Since PCAN-OBDonUDS API is built on top of the PCAN-UDS API, PCAN-ISO-TP API, and PCAN-Basic API, it shares similar functions. It offers the possibility to use several PCAN-OBDonUDS channels within the same application in an easy way. The communication process is divided in 3 phases: initialization, interaction, and finalization of a POBDonUDS channel.

- **Initialization:** In order to do OBDonUDS communication using a channel, it is necessary to initialize it first. This is done by making a call to the function `OBDonUDS_Initialize` (class-method: `Initialize`).
- **Interaction:** After a successful initialization, a channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions starting with `OBDonUDS_Request` (class-methods: starting with `Request`) can be used to transmit legislated-OBDonUDS requests. The utility functions starting with `OBDonUDS_WaitFor` (class-methods: starting with `WaitFor`) are used to retrieve the results of a previous request (raw responses of ECUs). The functions starting with `OBDonUDS_Parse` (class methods starting with `Parse`) can be used to parse the raw responses.
- **Finalization:** When the communication is finished, the function `OBDonUDS_Uninitialize` (class-method: `Uninitialize`) should be called in order to release the POBDonUDS channel and the resources allocated for it. In this way the channel is marked as "Free" and can be used by other applications.

## 2.3 Features

- Implementation of the OBDOnUDS protocol (SAE J1979-2, ISO 14229-1, ISO 15765-4:2021) as on-board diagnostics standard
- Windows DLLs for the development of applications for the platforms Windows 11 (x64/ARM64), 10 (x86/x64)
- Thread-safe API
- Physical communication via CAN using a CAN interface of the PCAN series
- Uses the PCAN-Basic programming interface to access the CAN hardware in the computer
- Uses the PCAN-ISO-TP programming interface (ISO 15765-2) for the transfer of data packages up to 4095 bytes via the CAN bus
- Uses the PCAN-UDS programming interface (ISO 14229-1) for the communication with control units

## 2.4 System Requirements

- Windows 11 (x64/ARM64), Windows 10 (x64)
- For the CAN bus connection: PC CAN interface from PEAK-System
- PCAN-Basic API
- PCAN-ISO-TP API
- PCAN-UDS API

## 2.5 Scope of Supply

### Download

- Interface DLLs (x86/x64/ARM64)
- Examples and header files for all common programming languages
- Documentation in PDF format

## 3 DLL API Reference

This section contains information about the data types (classes, structures, types, defines, enumerations) and API functions which are contained in the PCAN-OBDonUDS API.

### 3.1 Namespaces

PEAK-System offers the implementation of some specific programming interfaces as namespaces for the .NET Framework programming environment. The following namespaces are to notice:

Name	Description
Peak	Contains all namespaces that are part of the managed programming environment from PEAK-System.
Peak.Can	Contains types and classes for using the PCAN-API from PEAK-System.
Peak.Can.Basic	Contains types and classes for using the PCAN-Basic API from PEAK-System.
Peak.Can.IsoTp	Contains types and classes for using the PCAN-ISO-TP API implementation from PEAK-System.
Peak.Can.Uds	Contains types and classes for using the PCAN-UDS API implementation from PEAK-System.
Peak.Can.ObdII	Contains types and classes for using the PCAN-OBDonUDS API implementation from PEAK-System.
Peak.Can.OBDonUDS	Contains types and classes for using the PCAN-OBDonUDS API implementation from PEAK-System.

#### 3.1.1 Peak.Can.OBDonUDS

The Peak.Can.OBDonUDS namespace contains types and classes to use the PCAN-OBDonUDS API within the .NET Framework programming environment and handle PCAN devices from PEAK-System.

##### Remarks

Under the future Delphi environment, these elements are enclosed in the POBDonUDS-Unit. The functionality of all elements included here is just the same. The difference between this namespace and the Delphi unit consists in the fact that Delphi accesses the Windows API directly (it is not Managed Code).

##### Classes

Class	Description
OBDonUDSApi	Defines a class which represents the PCAN-OBDonUDS API

##### Structures

Structure	Description
obd_netaddrinfo	Represents a OBDonUDS Network Addressing Information
obd_msgaccess	Provides accessors to the corresponding data in the cantp_msg
obd_msg	Represents the OBDonUDS message
obd_response_generic	Internal reserved structure for parsed responses
obd_did_object	Represents a part of a parsed response to service 22-DID
obd_request_current_data_response	Represents a parsed response to service 22-DID
obd_request_freeze_frame_data_response	Represents a parsed response to service 19-04
obd_severity_trouble_code	Represents a part of a parsed response to service 19-42

Structure	Description
obd_request_trouble_codes_response	Represents a parsed response to service 19-42
obd_request_clear_trouble_codes_response	Represents a parsed response to service 14
obd_test_data_object	Represents a part of a parsed response to service 22-MID
obd_request_test_results_response	Represents a parsed response to service 22-MID
obd_request_control_operation_response	Represents a parsed response to service 31
obd_request_vehicle_information_response	Represents a parsed response to service 22-ITID
obd_trouble_code	Represents a part of a parsed response to service 19-55
obd_request_permanent_trouble_codes_response	Represents a parsed response to service 19-55
obd_request_supported_dtc_extended_response	Represents a parsed response to service 19-1A
obd_request_dtc_extended_response	Represents a parsed response to service 19-06
obd_request_dtc_for_a_readiness_group_response	Represents a parsed response to service 19-56

## Enumerations

Enumeration	Definition
obd_ecu	Addresses of ECU
obd_service	Service IDs
obd_sub_function	Sub-functions of services
obd_errstatus	POBDonUDS error codes (used in obd_status)
obd_status	POBDonUDS status codes
obd_parameter	POBDonUDS parameter to be read or set
obd_baudrate	POBDonUDS baud rates
obd_msgprotocol	POBDonUDS message protocols
obd_addressing	POBDonUDS addressing modes
obd_datatype	Reserved enumeration of internal parsed responses types

## Aliases

Enumeration	Definition
obd_DID_t	A DID value on 2 bytes
obd_DTC_t	A DTC value on 4 bytes
obd_RID_t	A RID value on 2 bytes

## 3.2 Units

NA

## 3.3 Classes

The following classes are offered to make use of the PCAN-OBDonUDS API in a managed or unmanaged way.

Class	Description
OBDonUDSApi	Defines a class which represents the PCAN-OBDonUDS API within the Microsoft's .NET Framework programming environment.

### 3.3.1 OBDonUDSApi

Defines a class which represents the PCAN-OBDonUDS API within the Microsoft's .NET Framework programming environment.

#### Syntax

**C#**

**C/C++**

```
public static class OBDonUDSApi
```

#### Remarks

The OBDonUDSApi class collects and implements the PCAN-OBDonUDS API methods. Each method is called just like the API function with the exception that the prefix OBDonUDS\_ is not used. The structure and functionality of the methods and API functions are the same.

Within the .NET Framework from Microsoft, the OBDonUDSApi class is a static, not inheritable, class. It must directly be used, without any instance of it, e.g.

```
obd_status res;  
// Static use, without any instance  
//  
res = OBDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_USBBUS1,  
                             (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_AUTODETECT);
```

#### See also

"Methods" on page 51, "Definitions" on page 194

## 3.4 Structures

The PCAN-OBDonUDS API defines the following structures:

Name	Description
obd_netaddrinfo	Represents a OBDonUDS Network Addressing Information
obd_msgaccess	Provides accessors to the corresponding data in the cantp_msg
obd_msg	Represents the OBDonUDS message
obd_response_generic	Internal reserved structure for parsed responses
obd_did_object	Represents a part of a parsed response to service 22-DID
obd_request_current_data_response	Represents a parsed response to service 22-DID
obd_request_freeze_frame_data_response	Represents a parsed response to service 19-04
obd_severity_trouble_code	Represents a part of a parsed response to service 19-42
obd_request_trouble_codes_response	Represents a parsed response to service 19-42
obd_request_clear_trouble_codes_response	Represents a parsed response to service 14
obd_test_data_object	Represents a part of a parsed response to service 22-MID
obd_request_test_results_response	Represents a parsed response to service 22-MID
obd_request_control_operation_response	Represents a parsed response to service 31
obd_request_vehicle_information_response	Represents a parsed response to service 22-ITID
obd_trouble_code	Represents a part of a parsed response to service 19-55
obd_request_permanent_trouble_codes_response	Represents a parsed response to service 19-55
obd_request_supported_dtc_extended_response	Represents a parsed response to service 19-1A
obd_request_dtc_extended_response	Represents a parsed response to service 19-06
obd_request_dtc_for_a_readiness_group_response	Represents a parsed response to service 19-56

### 3.4.1 obd\_netaddrinfo

Represents a OBDOnUDS Network Addressing Information.

#### Syntax

##### C/C++

```
typedef struct _obd_netaddrinfo {
    obd_msgprotocol protocol;
    obd_addressing target_type;
    uint16_t source_addr;
    uint16_t target_addr;
} obd_netaddrinfo;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_netaddrinfo
{
    [MarshalAs(UnmanagedType.U4)]
    public obd_msgprotocol protocol;
    [MarshalAs(UnmanagedType.U4)]
    public obd_addressing target_type;
    public UInt16 source_addr;
    public UInt16 target_addr;
}
```

#### Fields

Name	Description
protocol	Communication protocol (see "obd_msgprotocol" on page 45)
target_type	ISO-TP target type (physical or functional) (see "obd_addressing" on page 46)
source_addr	Source address (by default should be PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT for requests)
target_addr	Target address (see "obd_ecu" on page 34)



### 3.4.2 obd\_msgaccess

Provides accessors to the corresponding data in the underlying `cantp_msg` of `obd_msg`.

#### Syntax

##### C/C++

```
typedef struct _obd_msgaccess {
    uint8_t* service_id;
    uint8_t* nrc;
    cantp_netstatus* network_result;
} obd_msgaccess;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_msgaccess
{
    public IntPtr service_id;
    public IntPtr nrc;
    public IntPtr network_result;
}
```

#### Fields

Name	Description
Name	Description
service_id	Pointer to the Service ID in message's data.
nrc	Pointer to the Negative Response Code (see <code>uds_nrc</code> enumeration of PCAN-UDS API at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50) in message's data (NULL on positive response).
network_result	Pointer to the ISOTP network result (see <code>cantp_netstatus</code> enumeration of PCAN-ISOTP API at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50).

#### .NET Remarks:

Pointers can be used in safe mode with `Marshal.ReadByte` or in unsafe mode with `IntPtr.toPointer`. See also: `Helpers` (C# specific methods).

### 3.4.3 obd\_msg

Represents the OBDOnUDS message.

**Syntax**

**C/C++**

```
typedef struct _obd_msg {
    obd_netaddrinfo nai;
    obd_msgaccess links;
    uds_msg msg;
} obd_msg;
```

**C#**

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_msg
{
    public obd_netaddrinfo nai;
    public obd_msgaccess links;
    public uds_msg msg;
}
```

**Fields**

Name	Description
nai	Network address information (see "obd_netaddrinfo" on page 16)
links	Links to information in underlying messages (see "obd_msgaccess" on the previous page)
msg	The underlying PCAN-UDS message data (see uds_msg of the PCAN-UDS API at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

### 3.4.4 obd\_response\_generic

Internal reserved structure for parsed responses.

### 3.4.5 obd\_did\_object

Represents a part of a parsed response to service 22-DID.

#### Syntax

##### C/C++

```
typedef struct _obd_did_object {
    obd_DID_t data_identifier;
    uint32_t size;
    uint8_t* data;
} obd_did_object;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_did_object
{
    [MarshalAs(UnmanagedType.U2)]
    public obd_DID_t data_identifier;
    public UInt32 size;
    public IntPtr data;
}
```

#### Fields

Name	Description
data_identifier	DID (see "obd_DID_t" on page 48)
size	Size (number of bytes) of "data"
data	Array of bytes representing data dedicated to "data_identifier"
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.Copy or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

### 3.4.6 obd\_request\_current\_data\_response

Represents a parsed response to service 22-DID.

#### Syntax

##### C/C++

```
typedef struct _obd_request_current_data_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint32_t nb_elements;
    obd_did_object* elements;
} obd_request_current_data_response;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_current_data_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
nb_elements	Number of objects in "elements"
elements	Array of objects (see "obd_did_object" on the previous page)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

### 3.4.7 obd\_request\_freeze\_frame\_data\_response

Represents a parsed response to service 19-04.

#### Syntax

##### C/C++

```
typedef struct _obd_request_freeze_frame_data_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    obd_DTC_t dtc_number;
    uint8_t status_of_dtc;
    uint8_t record_number;
    uint8_t nb_identifiers;
    obd_did_object* identifiers;
} obd_request_freeze_frame_data_response;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_freeze_frame_data_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    [MarshalAs(UnmanagedType.U4)]
    public obd_DTC_t dtc_number;
    public byte status_of_dtc;
    public byte record_number;
    public byte nb_identifiers;
    public IntPtr identifiers;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
report_type	report Type
dtc_number	DTC (see " obd_DTC_t" on page 48)
status_of_dtc	Status Of DTC
record_number	DTC Snapshot Record Number
nb_identifiers	DTC Snapshot Record Number Of Identifiers, number of objects in "identifiers"
identifiers	Array of objects (DTC Snapshot Record Data Record) (see "obd_did_object" on page 19)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

### 3.4.8 obd\_severity\_trouble\_code

Represents a part of a parsed response to service 19-42.

#### Syntax

##### C/C++

```
typedef struct _obd_severity_trouble_code {
    uint8_t DTC_severity;
    obd_DTC_t DTC_identifier;
    uint8_t status_of_dtc;
} obd_severity_trouble_code;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_severity_trouble_code
{
    public byte DTC_severity;
    [MarshalAs(UnmanagedType.U4)]
    public obd_DTC_t DTC_identifier;
    public byte status_of_dtc;
}
```

#### Fields

Name	Description
DTC_severity	DTC Severity
DTC_identifier	DTC (see " obd_DTC_t" on page 48)
status_of_dtc	Status of DTC

### 3.4.9 obd\_request\_trouble\_codes\_response

Represents a parsed response to service 19-42.

#### Syntax

#### C/C++

```
typedef struct _obd_request_trouble_codes_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    uint8_t functional_group_identifier;
    uint8_t DTC_status_availability_mask;
    uint8_t DTC_severity_mask;
    uint8_t DTC_format_identifier;
    uint32_t nb_elements;
    obd_severity_trouble_code* elements;
} obd_request_trouble_codes_response;
```

#### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_trouble_codes_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    public byte functional_group_identifier;
    public byte DTC_status_availability_mask;
    public byte DTC_severity_mask;
    public byte DTC_format_identifier;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
report_type	DTC Report Type
functional_group_identifier	Functional Group Identifier
DTC_status_availability_mask	DTC Status Availability Mask
DTC_severity_mask	DTC Severity Availability Mask
DTC_format_identifier	DTC Format Identifier
nb_elements	Number of objects in "elements"
elements	Array of objects (DTC And Severity Record) (see "obd_severity_trouble_code" on the previous page)

**.NET Remarks:**

Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).

### 3.4.10 obd\_request\_clear\_trouble\_codes\_response

Represents a parsed response to service 14.

#### Syntax

##### C/C++

```
typedef struct _obd_request_clear_trouble_codes_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
} obd_request_clear_trouble_codes_response;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_clear_trouble_codes_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)



### 3.4.11 obd\_test\_data\_object

Represents a part of a parsed response to service 22-MID.

#### Syntax

##### C/C++

```
typedef struct _obd_test_data_object {
    uint8_t test_identifier;
    uint8_t unit_and_scaling;
    uint8_t test_value[2];
    uint8_t min_test_limit[2];
    uint8_t max_test_limit[2];
} obd_test_data_object;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_test_data_object
{
    public byte test_identifier;
    public byte unit_and_scaling;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
    public byte[] test_value;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
    public byte[] min_test_limit;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
    public byte[] max_test_limit;
}
```

#### Fields

Name	Description
test_identifier	TID
unit_and_scaling	Unit and scaling ID
test_value[2]	Test value
min_test_limit[2]	Min. test limit
max_test_limit[2]	Max. test limit

### 3.4.12 obd\_request\_test\_results\_response

Represents a parsed response to service 22-MID.

#### Syntax

##### C/C++

```
typedef struct _obd_request_test_results_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    obd_DID_t data_identifier;
    uint32_t nb_elements;
    obd_test_data_object* elements;
} obd_request_test_results_response;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_test_results_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    [MarshalAs(UnmanagedType.U2)]
    public obd_DID_t data_identifier;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
data_identifier	DID (see "obd_DID_t" on page 48)
nb_elements	Number of objects in "elements"
elements	Array of objects (data record of supported MID) (see "obd_test_data_object" on the previous page)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

### 3.4.13 obd\_request\_control\_operation\_response

Represents a parsed response to service 31.

#### Syntax

#### C/C++

```
typedef struct _obd_request_control_operation_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t routine_control_type;
    obd_RID_t routine_identifier;
    uint8_t routine_info;
    uint8_t nb_elements;
    uint8_t* elements;
} obd_request_control_operation_response;
```

#### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_control_operation_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte routine_control_type;
    [MarshalAs(UnmanagedType.U2)]
    public obd_RID_t routine_identifier;
    public byte routine_info;
    public byte nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
routine_control_type	Routine Control Type
routine_identifier	RID (see " obd_RID_t" on page 49)
routine_info	Routine Info
nb_elements	Number of bytes in "elements"
elements	Array of bytes (Routine Status Record)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.Copy or unsafe mode with IntPtr.ToPointer. See also: Helpers (C# specific methods).	

#### .NET Remarks :

Pointers can be used in safe mode with Marshal.ReadByte or in unsafe mode with IntPtr.ToPointer. See also: Helpers (C# specific methods).

3.4.14 obd\_request\_vehicle\_information\_response

Represents a parsed response to service service 22-ITID.

Syntax

C/C++

```
typedef struct _obd_request_vehicle_information_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    obd_DID_t data_identifier;
    uint32_t nb_elements;
    uint8_t* elements;
} obd_request_vehicle_information_response;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_vehicle_information_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    [MarshalAs(UnmanagedType.U2)]
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
data_identifier	DID (ITID) (DID in data record of InfoType) (see "obd_DID_t" on page 48)
nb_elements	Number of bytes in "elements"
elements	Array of bytes (data in data record of InfoType)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.Copy or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

### 3.4.15 obd\_trouble\_code

Represents a part of a parsed response to service 19-55.

#### Syntax

##### C/C++

```
typedef struct _obd_trouble_code {
    obd_DTC_t DTC_identifier;
    uint8_t status_of_dtc;
} obd_trouble_code;
```

##### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_trouble_code
{
    [MarshalAs(UnmanagedType.U4)]
    public obd_DTC_t DTC_identifier;
    public byte status_of_dtc;
}
```

#### Fields

Name	Description
DTC_identifier	DTC in DTCAndStatusRecord (see " obd_DTC_t" on page 48)
status_of_dtc	Status of DTC

### 3.4.16 obd\_request\_permanent\_trouble\_codes\_response

Represents a parsed response to service 19-55.

#### Syntax

##### C/C++

```
typedef struct _obd_request_permanent_trouble_codes_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    uint8_t functional_group_identifier;
    uint8_t DTC_status_availability_mask;
    uint8_t DTC_format_identifier;
    uint32_t nb_elements;
    obd_trouble_code* elements;
} obd_request_permanent_trouble_codes_response;
```

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_permanent_trouble_codes_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    public byte functional_group_identifier;
    public byte DTC_status_availability_mask;
    public byte DTC_format_identifier;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
report_type	DTC Report Type
functional_group_identifier	Functional Group Identifier
DTC_status_availability_mask	DTC Status Availability Mask
DTC_format_identifier	DTC Format Identifier
nb_elements	Number of objects in "elements"
elements	Array of objects (DTCAndStatusRecord) (see "obd_trouble_code" on the previous page)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

.NET Remarks:

Pointers can be used in safe mode with Marshal.ReadByte or in unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).

### 3.4.17 obd\_request\_supported\_dtc\_extended\_response

Represents a parsed response to service 19-1A.

#### Syntax

#### C/C++

```
typedef struct _obd_request_supported_dtc_extended_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    uint8_t DTC_status_availability_mask;
    uint8_t DTC_extended_data_record_number;
    uint32_t nb_elements;
    obd_trouble_code* elements;
} obd_request_supported_dtc_extended_response;
```

#### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_supported_dtc_extended_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    public byte DTC_status_availability_mask;
    public byte DTC_extended_data_record_number;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
report_type	DTC Report Type
DTC_status_availability_mask	DTC Status Availability Mask
DTC_extended_data_record_number	DTC Extended Data Record Number
nb_elements	Number of objects in "elements"
elements	Array of objects (DTCAndStatusRecord) (see "obd_trouble_code" on page 29)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	

#### .NET Remarks:

Pointers can be used in safe mode with Marshal.ReadByte or in unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).

### 3.4.18 obd\_request\_dtc\_extended\_response

Represents a parsed response to service 19-06.

#### Syntax

#### C/C++

```
typedef struct _obd_request_dtc_extended_response {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    obd_DTC_t DTC_identifier;
    uint8_t status_of_dtc;
    uint8_t DTC_extended_data_record_number;
    uint32_t nb_elements;
    uint8_t* elements;
} obd_request_dtc_extended_response;
```

#### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_dtc_extended_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    [MarshalAs(UnmanagedType.U4)]
    public obd_DTC_t DTC_identifier;
    public byte status_of_dtc;
    public byte DTC_extended_data_record_number;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on page 34)
nrc	Negative Response Code (0 if no NRC)
report_type	DTC Report Type
DTC_identifier	DTC in DTCAndStatusRecord (see " obd_DTC_t" on page 48)
status_of_dtc	Status of DTC
DTC_extended_data_record_number	DTC Extended Data Record Number
nb_elements	Number of bytes in "elements"
elements	Array of bytes (DTCExtendedDataRecord)
<b>.NET Remarks:</b>	
Pointers can be used in safe mode with Marshal.Copy or unsafe mode with IntPtr.toPointer. See also: Helpers (C# specific methods).	



### 3.4.19 obd\_request\_dtc\_for\_a\_readiness\_group\_response

Represents a parsed response to service 19-56.

#### Syntax

#### C/C++

```
typedef struct _obd_request_dtc_for_a_readiness_group {
    obd_datatype reserved_object_type;
    uint32_t reserved_object_size;
    uint16_t ecu_address;
    uint8_t nrc;
    uint8_t report_type;
    uint8_t functional_group_identifier;
    uint8_t DTC_status_availability_mask;
    uint8_t DTC_format_identifier;
    uint8_t readiness_group_identifier;
    uint32_t nb_elements;
    obd_trouble_code* elements;
} obd_request_dtc_for_a_readiness_group_response;
```

#### C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct obd_request_dtc_for_a_readiness_group_response
{
    [MarshalAs(UnmanagedType.U1)]
    public obd_datatype reserved_object_type;
    public UInt32 reserved_object_size;
    public UInt16 ecu_address;
    public byte nrc;
    public byte report_type;
    public byte functional_group_identifier;
    public byte DTC_status_availability_mask;
    public byte DTC_format_identifier;
    public byte readiness_group_identifier;
    public UInt32 nb_elements;
    public IntPtr elements;
}
```

#### Fields

Name	Description
reserved_object_type	reserved
reserved_object_size	reserved
ecu_address	ECU address (see "obd_ecu" on the next page)
nrc	Negative Response Code (0 if no NRC)
report_type	DTC Report Type
functional_group_identifier	Functional Group Identifier
DTC_status_availability_mask	DTC Status Availability Mask
DTC_format_identifier	DTC Format Identifier
readiness_group_identifier	DTC Readiness Group Identifier
nb_elements	Number of objects in "elements"
elements	Array of objects (DTCAndStatusRecord) (see "obd_trouble_code" on page 29)

**.NET Remarks:**  
Pointers can be used in safe mode with Marshal.PtrToStructure or unsafe mode with IntPtr.toPointer.  
See also: Helpers (C# specific methods).

## 3.5 Types

The PCAN-OBDonUDS API defines the following types:

Name	Definition
obd_ecu	Addresses of ECU
obd_service	Service IDs
obd_sub_function	Sub-functions of services
obd_errstatus	POBDonUDS error codes (used in obd_status)
obd_status	POBDonUDS status codes
obd_parameter	POBDonUDS parameter to be read or set
obd_baudrate	POBDonUDS baud rates
obd_msgprotocol	POBDonUDS message protocols
obd_addressing	POBDonUDS addressing modes
obd_datatype	Reserved enumeration of internal parsed responses types
obd_DID_t	A DID value on 2 bytes
obd_DTC_t	A DTC value on 4 bytes
obd_RID_t	A RID value on 2 bytes

### 3.5.1 obd\_ecu

Represents the addresses of ECUs.

#### Syntax

#### C/C++

```
typedef enum _obd_ecu {
    POBD_ECU_1 = PUDS_ISO_15765_4_ADDR_ECU_1,
    POBD_ECU_2 = PUDS_ISO_15765_4_ADDR_ECU_2,
    POBD_ECU_3 = PUDS_ISO_15765_4_ADDR_ECU_3,
    POBD_ECU_4 = PUDS_ISO_15765_4_ADDR_ECU_4,
    POBD_ECU_5 = PUDS_ISO_15765_4_ADDR_ECU_5,
    POBD_ECU_6 = PUDS_ISO_15765_4_ADDR_ECU_6,
    POBD_ECU_7 = PUDS_ISO_15765_4_ADDR_ECU_7,
    POBD_ECU_8 = PUDS_ISO_15765_4_ADDR_ECU_8,
} obd_ecu;
```

#### C#

```
public enum obd_ecu : UInt16
{
    POBD_ECU_1 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_1,
    POBD_ECU_2 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_2,
    POBD_ECU_3 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_3,
    POBD_ECU_4 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_4,
    POBD_ECU_5 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_5,
    POBD_ECU_6 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_6,
    POBD_ECU_7 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_7,
    POBD_ECU_8 = uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_ECU_8,
}
```

## Values

Name	Value	Description
POBD_ECU_1	PUDS_ISO_15765_4_ADDR_ECU_1 (see PCAN-UDS API)	ECU #1
POBD_ECU_2	PUDS_ISO_15765_4_ADDR_ECU_2 (see PCAN-UDS API)	ECU #2
POBD_ECU_3	PUDS_ISO_15765_4_ADDR_ECU_3 (see PCAN-UDS API)	ECU #3
POBD_ECU_4	PUDS_ISO_15765_4_ADDR_ECU_4 (see PCAN-UDS API)	ECU #4
POBD_ECU_5	PUDS_ISO_15765_4_ADDR_ECU_5 (see PCAN-UDS API)	ECU #5
POBD_ECU_6	PUDS_ISO_15765_4_ADDR_ECU_6 (see PCAN-UDS API)	ECU #6
POBD_ECU_7	PUDS_ISO_15765_4_ADDR_ECU_7 (see PCAN-UDS API)	ECU #7
POBD_ECU_8	PUDS_ISO_15765_4_ADDR_ECU_8 (see PCAN-UDS API)	ECU #8

### 3.5.2 obd\_service

Represents the service IDs.

#### Syntax

##### C/C++

```
typedef enum _obd_service {
    POBD_SERVICE_14 = 0x14,
    POBD_SERVICE_19 = 0x19,
    POBD_SERVICE_22 = 0x22,
    POBD_SERVICE_31 = 0x31,
} obd_service;
```

##### C#

```
public enum obd_service : byte
{
    POBD_SERVICE_14 = 0x14,
    POBD_SERVICE_19 = 0x19,
    POBD_SERVICE_22 = 0x22,
    POBD_SERVICE_31 = 0x31,
}
```

## Values

Name	Value	Description
POBD_SERVICE_14	0x14	Service \$14
POBD_SERVICE_19	0x19	Service \$19
POBD_SERVICE_22	0x22	Service \$22
POBD_SERVICE_31	0x31	Service \$31

### 3.5.3 obd\_sub\_function

Represents the sub-functions of services.

#### Syntax

##### C/C++

```
typedef enum _obd_sub_function {
    POBD_SUB_FUNCTION_04 = 0x04,
    POBD_SUB_FUNCTION_42 = 0x42,
    POBD_SUB_FUNCTION_55 = 0x55,
    POBD_SUB_FUNCTION_1A = 0x1A,
    POBD_SUB_FUNCTION_06 = 0x06,
    POBD_SUB_FUNCTION_56 = 0x56,
} obd_sub_function;
```

##### C#

```
public enum obd_sub_function : byte
{
    POBD_SUB_FUNCTION_04 = 0x04,
    POBD_SUB_FUNCTION_42 = 0x42,
    POBD_SUB_FUNCTION_55 = 0x55,
    POBD_SUB_FUNCTION_1A = 0x1A,
    POBD_SUB_FUNCTION_06 = 0x06,
    POBD_SUB_FUNCTION_56 = 0x56,
}
```

#### Values

Name	Value	Description
POBD_SUB_FUNCTION_04	0x04	Sub-function \$04
POBD_SUB_FUNCTION_42	0x42	Sub-function \$42
POBD_SUB_FUNCTION_55	0x55	Sub-function \$55
POBD_SUB_FUNCTION_1A	0x1A	Sub-function \$1A
POBD_SUB_FUNCTION_06	0x06	Sub-function \$06
POBD_SUB_FUNCTION_56	0x56	Sub-function \$56

### 3.5.4 obd\_errstatus

Represents the POBDOnUDS error codes used in obd\_status (see " obd\_status" on the next page).

#### Syntax

##### C/C++

```
typedef enum _obd_errstatus {
    POBDONUDS_ERRSTATUS_UNSUPPORTED_ECUS = 1,
    POBDONUDS_ERRSTATUS_INVALID_REQUEST_ADDRESSING,
    POBDONUDS_ERRSTATUS_INVALID_SI,
    POBDONUDS_ERRSTATUS_INVALID_DATA_LENGTH,
    POBDONUDS_ERRSTATUS_NETWORK_ERROR,
    POBDONUDS_ERRSTATUS_NOT_PARSABLE,
    POBDONUDS_CAUTION_NRC_NOT_NULL,
} obd_errstatus;
```

## C#

```
public enum obd_errstatus : byte
{
    POBDONUDS_ERRSTATUS_UNSUPPORTED_ECUS = 1,
    POBDONUDS_ERRSTATUS_INVALID_REQUEST_ADDRESSING,
    POBDONUDS_ERRSTATUS_INVALID_SI,
    POBDONUDS_ERRSTATUS_INVALID_DATA_LENGTH,
    POBDONUDS_ERRSTATUS_NETWORK_ERROR,
    POBDONUDS_ERRSTATUS_NOT_PARSABLE,
    POBDONUDS_CAUTION_NRC_NOT_NULL,
}
```

## Values

Name	Value	Description
POBDONUDS_ERRSTATUS_UNSUPPORTED_ECUS	1	No OBDOnUDS ECU found
POBDONUDS_ERRSTATUS_INVALID_REQUEST_ADDRESSING	2	Invalid request addressing mode
POBDONUDS_ERRSTATUS_INVALID_SI	3	Invalid Service ID found while parsing (null pointer or bad value)
POBDONUDS_ERRSTATUS_INVALID_DATA_LENGTH	4	Invalid length found while parsing
POBDONUDS_ERRSTATUS_NETWORK_ERROR	5	Invalid Network result found while parsing (null pointer or value indicating an error)
POBDONUDS_ERRSTATUS_NOT_PARSABLE	6	The requested DID/MID/RID/ITID corresponds to a request of type "get supported identifiers", the response is not parsable
POBDONUDS_CAUTION_NRC_NOT_NULL	7	NRC not null found while parsing. This is a caution indicating that the response is a negative response.

### 3.5.5 obd\_status

Represents the POBDOnUDS status codes.

## Syntax

## C/C++

```
typedef enum _obd_status {
    POBDONUDS_STATUS_OK,
    POBDONUDS_STATUS_NOT_INITIALIZED,
    POBDONUDS_STATUS_ALREADY_INITIALIZED,
    POBDONUDS_STATUS_NO_MEMORY,
    POBDONUDS_STATUS_PARAM_INVALID_TYPE,
    POBDONUDS_STATUS_PARAM_INVALID_VALUE,
    POBDONUDS_STATUS_UNKNOWN,
    POBDONUDS_STATUS_HANDLE_INVALID,
    POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR,
    POBDONUDS_STATUS_MASK_OBDONUDS_ERROR,
    POBDONUDS_STATUS_UNSUPPORTED_ECUS,
    POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING,
    POBDONUDS_STATUS_INVALID_SI,
    POBDONUDS_STATUS_INVALID_DATA_LENGTH,
    POBDONUDS_STATUS_NETWORK_ERROR,
    POBDONUDS_STATUS_NOT_PARSABLE,
    POBDONUDS_STATUS_CAUTION_NRC_NOT_NULL
} obd_status;
```

```

public enum obd_status : UInt32
{
    POBDONUDS_STATUS_OK,
    POBDONUDS_STATUS_NOT_INITIALIZED,
    POBDONUDS_STATUS_ALREADY_INITIALIZED,
    POBDONUDS_STATUS_NO_MEMORY,
    POBDONUDS_STATUS_PARAM_INVALID_TYPE,
    POBDONUDS_STATUS_PARAM_INVALID_VALUE,
    POBDONUDS_STATUS_UNKNOWN,
    POBDONUDS_STATUS_HANDLE_INVALID,
    POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR,
    POBDONUDS_STATUS_MASK_OBDONUDS_ERROR,
    POBDONUDS_STATUS_UNSUPPORTED_ECUS,
    POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING,
    POBDONUDS_STATUS_INVALID_DATA_LENGTH,
    POBDONUDS_STATUS_NETWORK_ERROR,
    POBDONUDS_STATUS_NOT_PARSABLE,
    POBDONUDS_STATUS_CAUTION_NRC_NOT_NULL,
}

```

## Values

Name	Value	Description
POBDONUDS_STATUS_OK	PUDS_STATUS_OK	No error
POBDONUDS_STATUS_NOT_INITIALIZED	PUDS_STATUS_NOT_INITIALIZED	Not initialized
POBDONUDS_STATUS_ALREADY_INITIALIZED	PUDS_STATUS_ALREADY_INITIALIZED	Already Initialized
POBDONUDS_STATUS_NO_MEMORY	PUDS_STATUS_NO_MEMORY	Could not obtain memory
POBDONUDS_STATUS_PARAM_INVALID_TYPE	PUDS_STATUS_PARAM_INVALID_TYPE	Wrong message parameters
POBDONUDS_STATUS_PARAM_INVALID_VALUE	PUDS_STATUS_PARAM_INVALID_VALUE	Wrong message parameters
POBDONUDS_STATUS_UNKNOWN	PUDS_STATUS_UNKNOWN	Unknown/generic error
POBDONUDS_STATUS_HANDLE_INVALID	PUDS_STATUS_HANDLE_INVALID	Invalid cantp_handle
POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR	0x40 << PCANTP_STATUS_OFFSET_UDS	OBDOnUDS error flag
POBDONUDS_STATUS_MASK_OBDONUDS_ERROR	0x5F << PCANTP_STATUS_OFFSET_UDS	Filter OBDOnUDS error
POBDONUDS_STATUS_UNSUPPORTED_ECUS	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_UNSUPPORTED_ECUS << PCANTP_STATUS_OFFSET_UDS)	No OBDOnUDS ECU found
POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_INVALID_REQUEST_ADDRESSING << PCANTP_STATUS_OFFSET_UDS)	Invalid request addressing mode
POBDONUDS_STATUS_INVALID_SI	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_INVALID_SI << PCANTP_STATUS_OFFSET_UDS)	Invalid Service ID found while parsing (null pointer or bad value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_INVALID_DATA_LENGTH << PCANTP_STATUS_OFFSET_UDS)	Invalid length found while parsing
POBDONUDS_STATUS_NETWORK_ERROR	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_NETWORK_ERROR << PCANTP_STATUS_OFFSET_UDS)	Invalid Network result found while parsing (null pointer or value indicating an error)

Name	Value	Description
POBDONUDS_STATUS_NOT_PARSABLE	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_ERRSTATUS_NOT_PARSABLE << PCANTP_STATUS_OFFSET_UDS)	The requested DID/MID/RID/ITID corresponds to a request of type "get supported identifiers", the response is not parsable
POBDONUDS_STATUS_CAUTION_NRC_NOT_NULL	POBDONUDS_STATUS_FLAG_OBDONUDS_ERROR   (POBDONUDS_CAUTION_NRC_NOT_NULL << PCANTP_STATUS_OFFSET_UDS)	NRC not null found while parsing (Caution indicating that the response is a negative response)

## Remarks

All uds\_status values are compatible with obd\_status.

## See also

uds\_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50

cantp\_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50

"OBDonUDS\_StatusIsOk" on page 144

## 3.5.6 obd\_parameter

Represents a POBDONUDS parameter or value that can be read or set. With some exceptions, a channel must first be initialized before its parameters can be read or set.

## Syntax

### C/C++

```
typedef enum _obd_parameter {
    POBDONUDS_PARAMETER_API_VERSION = 0x301,
    POBDONUDS_PARAMETER_AVAILABLE_ECUS = 0x302,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F4XX = 0x303,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F5XX = 0x304,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F7XX = 0x305,
    POBDONUDS_PARAMETER_SUPPORTMASK_MIDS = 0x306,
    POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E0XX = 0x307,
    POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E1XX = 0x308,
    POBDONUDS_PARAMETER_SUPPORTMASK_ITIDS = 0x309,
    POBDONUDS_PARAMETER_DEBUG = 0x30A,
    POBDONUDS_PARAMETER_BAUDRATE = 0x30B,
    POBDONUDS_PARAMETER_CAN_ID_LENGTH = 0x30C
} obd_parameter;
```

### C#

```
public enum obd_parameter : UInt32
{
    POBDONUDS_PARAMETER_API_VERSION = 0x301,
    POBDONUDS_PARAMETER_AVAILABLE_ECUS = 0x302,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F4XX = 0x303,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F5XX = 0x304,
    POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F7XX = 0x305,
    POBDONUDS_PARAMETER_SUPPORTMASK_MIDS = 0x306,
    POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E0XX = 0x307,
    POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E1XX = 0x308,
    POBDONUDS_PARAMETER_SUPPORTMASK_ITIDS = 0x309,
    POBDONUDS_PARAMETER_DEBUG = 0x30A,
    POBDONUDS_PARAMETER_BAUDRATE = 0x30B,
    POBDONUDS_PARAMETER_CAN_ID_LENGTH = 0x30C
}
```

## Values

Name	Value	Data type	Description
POBDONUDS_PARAMETER_API_VERSION	0x301	byte array	The api version (as a null-terminated string)
POBDONUDS_PARAMETER_AVAILABLE_ECUS	0x302	byte	The number of OBDonUDS ECUs detected
POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F4XX	0x303	256 bytes array	Supported DIDs F4XX for Service 22: Request Current Powertrain Diagnostic Data
POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F5XX	0x304	256 bytes array	Supported DIDs F5XX for Service 22: Request Current Powertrain Diagnostic Data
POBDONUDS_PARAMETER_SUPPORTMASK_DIDS_F7XX	0x305	256 bytes array	Supported DIDs F7XX for Service 22: Request Current Powertrain Diagnostic Data
POBDONUDS_PARAMETER_SUPPORTMASK_MIDS	0x306	256 bytes array	Supported MIDs F6XX for Service 22: Request On-board Monitoring Test Results for Specific Monitored Systems
POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E0XX	0x307	256 bytes array	Supported RIDs E0XX for Service 31: Request Control of On-Board System, Test, or Component
POBDONUDS_PARAMETER_SUPPORTMASK_RIDS_E1XX	0x308	256 bytes array	Supported RIDs E1XX for Service 31: Request Control of On-Board System, Test, or Component
POBDONUDS_PARAMETER_SUPPORTMASK_ITIDS	0x309	256 bytes array	Supported InfoTypes(ITIDs) F8XX for Service 22: Request Vehicle Information
POBDONUDS_PARAMETER_DEBUG	0x30A	byte	The debug mode
POBDONUDS_PARAMETER_BAUDRATE	0x30B	obd_baudrate	The baudrate value(see "obd_baudrate" on page 45)
POBDONUDS_PARAMETER_CAN_ID_LENGTH	0x30C	obd_msgprotocol	The CAN identifier length (see "obd_msgprotocol" on page 45)

## Detailed Parameters Characteristics

### POBDONUDS\_PARAMETER\_API\_VERSION

**Access:** R

**Description:** This parameter is used to get information about the PCAN-OBDonUDS API implementation version.

**Possible values:** The value is a null-terminated string indicating the version number of the API implementation. The returned text has the following form: x,x,x,x for major, minor, release and build.

**Default value:** NA.

**PCAN-Device:** NA. Any PCAN device can be used, including the PCANTP\_HANDLE\_NONEBUS channel

### POBDONUDS\_PARAMETER\_AVAILABLE\_ECUS

**Access:** R

**Description:** This value is used to get the number of detected OBDonUDS ECUs.

**Possible values:** A positive numeric value (0 to 8 for legislated-OBDonUDS-compliant vehicle).

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).



#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_DIDS\_F4XX

**Access:** R

**Description:** This value is used to retrieve the list of supported DIDs F4XX for Service 22: Request Current Powertrain Diagnostic Data by the connected ECUs.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific DID (i.e. row 0 is DID F400, row 1 is DID F401, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the DID is supported by the ECU.

For instance, if the variable 'array' contains the list of supported identifiers and 'array'[12] = 5, then DID F40C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_DIDS\_F5XX

**Access:** R

**Description:** This value is used to retrieve the list of supported DIDs F5XX for Service 22: Request Current Powertrain Diagnostic Data by the connected ECUs.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific DID (i.e. row 0 is DID F500, row 1 is DID F501, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the DID is supported by the ECU.

For instance, if the variable 'array' contains the list of supported identifiers and 'array'[12] = 5, then DID F50C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_DIDS\_F7XX

**Access:** R

**Description:** This value is used to retrieve the list of supported DIDs F7XX for Service 22: Request Current Powertrain Diagnostic Data by the connected ECUs.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific DID (i.e. row 0 is DID F700, row 1 is DID F701, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the DID is supported by the ECU.

For instance, if the variable 'array' contains the list of supported identifiers and 'array'[12] = 5, then DID F70C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_MIDS

**Access:** R

**Description:** This value is used to retrieve the list of supported MIDs F6XX for Service 22: Request On-board Monitoring Test Results for Specific Monitored Systems.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific MID (i.e. row 0 is MID F600, row 1 is MID F601, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the MID is supported by the ECU.

For instance, if the variable 'array' contains the list of supported identifiers and 'array'[12] = 5, then MID F60C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_RIDS\_E0XX

**Access:** R

**Description:** This value is used to retrieve the list of supported RIDs E0XX for Service 31: Request Control of On-Board System, Test, or Component.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific RID (i.e. row 0 is RID E000, row 1 is RID E001, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the RID is supported by the ECU.

For instance, if the variable 'array' contains the list of supported identifiers and 'array'[12] = 5, then RID E00C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_SUPPORTMASK\_RIDS\_E1XX

**Access:** R

**Description:** This value is used to retrieve the list of supported RIDs E1XX for Service 31: Request Control of On-Board System, Test, or Component.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific RID (i.e. row 0 is RID E100, row 1 is RID E101, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the RID is supported by the ECU.

For instance, if the variable ‘array’ contains the list of supported identifiers and ‘array’[12] = 5, then RID E10C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

POBDONUDS\_PARAMETER\_SUPPORTMASK\_ITIDS

**Access:** R

**Description:** This value is used to retrieve the list of supported InfoTypes(ITIDs) F8XX for Service 22: Request Vehicle Information.

**Possible values:** The list is represented by an array of 256 bytes where each row corresponds to the support of a specific ITID (i.e. row 0 is ITID F800, row 1 is ITID F801, etc.). The values of the array are bit-encoded and each bit corresponds to an ECU where:

- bit #0 corresponds to ECU#1 and bit #7 to ECU#8,
- a bit set to 1 states that the ITID is supported by the ECU.

For instance, if the variable ‘array’ contains the list of supported identifiers and ‘array’[12] = 5, then ITID F80C is supported by ECU #1 and #3.

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

POBDONUDS\_PARAMETER\_DEBUG

**Access:** R/W

**Description:** This parameter is used to control debug mode. If enabled, any received or transmitted CAN frames will be logged in a PCAN-Basic log file (default file name is PCANBasic.log located inside the current directory).

**Possible values:**

Type	Constant	Value	Description
Byte	POBDONUDS_DEBUG_LVL_NONE	0x00	No debug messages are being generated.
Byte	POBDONUDS_DEBUG_LVL_ERROR	0xF1	Enable debug messages (only errors)
Byte	POBDONUDS_DEBUG_LVL_WARNING	0xF2	Enable debug messages (only warnings, errors)
Byte	POBDONUDS_DEBUG_LVL_INFORMATION	0xF3	Enable debug messages (only information, warnings, errors)
Byte	POBDONUDS_DEBUG_LVL_NOTICE	0xF4	Enable debug messages (only notices, information, warnings, errors)
Byte	POBDONUDS_DEBUG_LVL_DEBUG	0xF5	Enable debug messages (only debug, notices, information, warnings, errors)
Byte	POBDONUDS_DEBUG_LVL_TRACE	0xF6	Enable all debug messages

**Default value:** POBDUDS\_DEBUG\_LVL\_NONE.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel)

#### POBDONUDS\_PARAMETER\_BAUDRATE

**Access:** R

**Description:** This value is used to get the initialized or detected baudrate of the PCAN-OBDonUDS channel.

**Possible values:** see "obd\_baudrate" on the next page

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

#### POBDONUDS\_PARAMETER\_CAN\_ID\_LENGTH

**Access:** R

**Description:** This value is used to get the bit length of the CAN identifiers used during the communication with the ECUs.

**Possible values:** see "obd\_msgprotocol" on the next page

**Default value:** NA.

**PCAN-Device:** All PCAN devices (excluding PCANTP\_HANDLE\_NONEBUS channel).

### 3.5.7 obd\_baudrate

Represents the POBDonUDS baud rates.

**Syntax**

**C/C++**

```
typedef enum _obd_baudrate {
    OBD_BAUDRATE_NON_LEGISLATED = 0,
    OBD_BAUDRATE_250K = PCANTP_BAUDRATE_250K,
    OBD_BAUDRATE_500K = PCANTP_BAUDRATE_500K,
    OBD_BAUDRATE_AUTODETECT = 0xFFFFFFFFU
} obd_baudrate;
```

**C#**

```
public enum obd_baudrate : UInt32
{
    OBD_BAUDRATE_NON_LEGISLATED = 0,
    OBD_BAUDRATE_250K = cantp_baudrate.PCANTP_BAUDRATE_250K,
    OBD_BAUDRATE_500K = cantp_baudrate.PCANTP_BAUDRATE_500K,
    OBD_BAUDRATE_AUTODETECT = 0xFFFFFFFFU
}
```

**Values**

Name	Value	Description
OBD_BAUDRATE_NON_LEGISLATED	0	Non legislated-OBDonUDS baudrate (note: used only as a returned value of OBDonUDS_GetValue function with parameter POBDONUDS_PARAMETER_BAUDRATE)
OBD_BAUDRATE_250K	PCANTP_BAUDRATE_250K	Baudrate 250 kBit/s
OBD_BAUDRATE_500K	PCANTP_BAUDRATE_500K	Baudrate 500 kBit/s
OBD_BAUDRATE_AUTODETECT	0xFFFFFFFFU	Autodetect baudrate mechanism (note: used only with the OBDonUDS_Initialize function)

**See also**

"OBDonUDS\_Initialize" on page 137

### 3.5.8 obd\_msgprotocol

Represents the POBDonUDS message protocols.

**Syntax**

**C/C++**

```
typedef enum _obd_msgprotocol {
    OBD_MSGPROTOCOL_UNKNOWN = PCANTP_ISOTP_FORMAT_NONE,
    OBD_MSGPROTOCOL_11BIT = PCANTP_ISOTP_FORMAT_NORMAL,
    OBD_MSGPROTOCOL_29BIT = PCANTP_ISOTP_FORMAT_FIXED_NORMAL,
} obd_msgprotocol;
```

C#

```
public enum obd_msgprotocol : UInt32
{
    OBD_MSGPROTOCOL_UNKNOWN = cantp_isotp_format.PCANTP_ISOTP_FORMAT_NONE,
    OBD_MSGPROTOCOL_11BIT = cantp_isotp_format.PCANTP_ISOTP_FORMAT_NORMAL,
    OBD_MSGPROTOCOL_29BIT = cantp_isotp_format.PCANTP_ISOTP_FORMAT_FIXED_NORMAL,
}
```

Values

Name	Value	Description
OBD_MSGPROTOCOL_UNKNOWN	PCANTP_ISOTP_FORMAT_NONE	Invalid/undefined value
OBD_MSGPROTOCOL_11BIT	PCANTP_ISOTP_FORMAT_NORMAL	11 bit CAN Identifier length
OBD_MSGPROTOCOL_29BIT	PCANTP_ISOTP_FORMAT_FIXED_NORMAL	29 bit CAN Identifier length

3.5.9 obd\_addressing

Represents the POBDonUDS addressing modes.

Syntax

C/C++

```
typedef enum _obd_addressing {
    OBD_ADDRESSING_UNKNOWN = PCANTP_ISOTP_ADDRESSING_UNKNOWN,
    OBD_ADDRESSING_PHYSICAL = PCANTP_ISOTP_ADDRESSING_PHYSICAL,
    OBD_ADDRESSING_FUNCTIONAL = PCANTP_ISOTP_ADDRESSING_FUNCTIONAL
} obd_addressing;
```

C#

```
public enum obd_addressing : UInt32
{
    OBD_ADDRESSING_UNKNOWN = cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_UNKNOWN,
    OBD_ADDRESSING_PHYSICAL = cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL,
    OBD_ADDRESSING_FUNCTIONAL = cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_FUNCTIONAL
}
```

Values

Name	Value	Description
OBD_ADDRESSING_UNKNOWN	PCANTP_ISOTP_ADDRESSING_UNKNOWN	Unknown or invalid addressing mode
OBD_ADDRESSING_PHYSICAL	PCANTP_ISOTP_ADDRESSING_PHYSICAL	Physical addressing mode (point to point)
OBD_ADDRESSING_FUNCTIONAL	PCANTP_ISOTP_ADDRESSING_FUNCTIONAL	Functional addressing mode (broadcast)

See also

" Understanding PCAN-OBDonUDS API" on page 8

### 3.5.10 obd\_datatype

Represents reserved enumeration of internal parsed responses types.

#### Syntax

#### C/C++

```
typedef enum _obd_datatype {
    OBD_DATATYPE_UNKNOWN = 0,
    OBD_DATATYPE_REQUEST_CURRENT_DATA_RESPONSE,
    OBD_DATATYPE_REQUEST_FREEZE_FRAME_DATA_RESPONSE,
    OBD_DATATYPE_REQUEST_CONTROL_OPERATION_RESPONSE,
    OBD_DATATYPE_REQUEST_VEHICLE_INFORMATION_RESPONSE,
    OBD_DATATYPE_REQUEST_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_PERMANENT_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_READINESS_GROUP_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_SUPPORTED_DTC_EXTENDED_RESPONSE,
    OBD_DATATYPE_REQUEST_CLEAR_TROUBLE_CODES_RESPONSE,
    OBD_DATATYPE_REQUEST_TEST_RESULTS_RESPONSE,
    OBD_DATATYPE_REQUEST_DTC_EXTENDED_RESPONSE
} obd_datatype;
```

#### C#

```
public enum obd_datatype : byte
{
    OBD_DATATYPE_UNKNOWN = 0,
    OBD_DATATYPE_REQUEST_CURRENT_DATA_RESPONSE,
    OBD_DATATYPE_REQUEST_FREEZE_FRAME_DATA_RESPONSE,
    OBD_DATATYPE_REQUEST_CONTROL_OPERATION_RESPONSE,
    OBD_DATATYPE_REQUEST_VEHICLE_INFORMATION_RESPONSE,
    OBD_DATATYPE_REQUEST_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_PERMANENT_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_READINESS_GROUP_TROUBLECODES_RESPONSE,
    OBD_DATATYPE_REQUEST_SUPPORTED_DTC_EXTENDED_RESPONSE,
    OBD_DATATYPE_REQUEST_CLEAR_TROUBLE_CODES_RESPONSE,
    OBD_DATATYPE_REQUEST_TEST_RESULTS_RESPONSE,
    OBD_DATATYPE_REQUEST_DTC_EXTENDED_RESPONSE
}
```

#### Values

Name	Value	Description
OBD_DATATYPE_UNKNOWN	0	Reserved
OBD_DATATYPE_REQUEST_CURRENT_DATA_RESPONSE	1	Reserved
OBD_DATATYPE_REQUEST_FREEZE_FRAME_DATA_RESPONSE	2	Reserved
OBD_DATATYPE_REQUEST_CONTROL_OPERATION_RESPONSE	3	Reserved
OBD_DATATYPE_REQUEST_VEHICLE_INFORMATION_RESPONSE	4	Reserved
OBD_DATATYPE_REQUEST_TROUBLECODES_RESPONSE	5	Reserved
OBD_DATATYPE_REQUEST_PERMANENT_TROUBLECODES_RESPONSE	6	Reserved
OBD_DATATYPE_REQUEST_READINESS_GROUP_TROUBLECODES_RESPONSE	7	Reserved
OBD_DATATYPE_REQUEST_SUPPORTED_DTC_EXTENDED_RESPONSE	8	Reserved
OBD_DATATYPE_REQUEST_CLEAR_TROUBLE_CODES_RESPONSE	9	Reserved
OBD_DATATYPE_REQUEST_TEST_RESULTS_RESPONSE	10	Reserved
OBD_DATATYPE_REQUEST_DTC_EXTENDED_RESPONSE	11	Reserved

### 3.5.11 obd\_DID\_t

Represents the DID value on 2 bytes.

#### Syntax

##### C/C++

```
typedef uint16_t obd_DID_t;
```

##### C#

```
using obd_DID_t = UInt16;
```

#### .NET Framework programming languages:

An alias is used to represent an obd\_DID\_t under Microsoft .NET in order to originate a homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.OBDOnUDS Namespace for C#. However, including a namespace does not include the defined aliases.

If it is wished to work with aliases, those must be copied to the working file, right after the inclusion of the Peak.Can.OBDOnUDS Namespace. Otherwise, just use the native type.

##### C#

```
using System;  
using Peak.Can.OBDOnUDS;  
using obd_DID_t = UInt16;
```

### 3.5.12 obd\_DTC\_t

Represents the DTC value on 4 bytes.

#### Syntax

##### C/C++

```
typedef uint32_t obd_DTC_t;
```

##### C#

```
using obd_DTC_t = UInt32;
```

#### .NET Framework programming languages:

An alias is used to represent an obd\_DTC\_t under Microsoft .NET in order to originate a homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.OBDOnUDS Namespace for C#. However, including a namespace does not include the defined aliases.

If it is wished to work with aliases, those must be copied to the working file, right after the inclusion of the Peak.Can.OBDOnUDS Namespace. Otherwise, just use the native type.



## C#

```
using System;
using Peak.Can.OBDOnUDS;
using obd_DTC_t = UInt32;
```

### 3.5.13 obd\_RID\_t

Represents the the RID value on 2 bytes.

#### Syntax

#### C/C++

```
typedef uint16_t obd_RID_t;
```

## C#

```
using obd_RID_t = UInt16;
```

#### .NET Framework programming languages:

An alias is used to represent an obd\_RID\_t under Microsoft .NET in order to originate a homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.OBDOnUDS Namespace for C# However, including a namespace does not include the defined aliases.

If it is wished to work with aliases, those must be copied to the working file, right after the inclusion of the Peak.Can.OBDOnUDS Namespace. Otherwise, just use the native type.

## C#

```
using System;
using Peak.Can.OBDOnUDS;
using obd_RID_t = UInt16;
```

## 3.6 PCAN-UDS and PCAN-ISO-TP Dependencies

PCAN-OBDonUDS API is built on the PCAN-UDS API and PCAN-ISO-TP API, so it has some dependencies. These dependencies are listed below. For more information, please see the user manuals of these APIs.

### Aliases

Name	Description	API	See also
PCANTP_STATUS_OFFSET_xxx	Constants used by obd_status	PCAN-ISO-TP	"obd_status" on page 37
PUDS_ISO_15765_4_ADDR_ECU_xxx	PUDS ISO_15765_4 address definitions for ECUs	PCAN-UDS	"obd_ecu" on page 34
PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT	PUDS ISO_15765_4 address definitions for external test equipment	PCAN-UDS	"obd_netaddrinfo" on page 16
PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL	PUDS ISO_15765_4 address definitions for OBD functional scheme	PCAN-UDS	"obd_addressing" on page 46

### Enumerations

Name	Description	API	See also
cantp_baudrate, PCANTP_BAUDRATE_xxx	Baudrate register for the PCANTP channel	PCAN-ISO-TP	"obd_baudrate" on page 45
cantp_netstatus, PCANTP_NETSTATUS_xxx	Network result of the communication of an ISO-TP message	PCAN-ISO-TP	"obd_msgaccess" on page 17
cantp_handle, PCANTP_HANDLE_xxx	Currently defined and supported PCANTP handle (a.k.a. channels)	PCAN-ISO-TP	"OBDonUDS_Initialize" on page 137
cantp_hwtype, PCANTP_HWTYPE_xxx	Type of PCAN (non plug-n-play) hardware	PCAN-ISO-TP	"OBDonUDS_Initialize" on page 137
cantp_isotp_format, PCANTP_ISOTP_FORMAT_xxx	Addressing format of an ISO-TP message	PCAN-ISO-TP	"obd_msgprotocol" on page 45
cantp_isotp_addressing, PCANTP_ISOTP_ADDRESSING_xxx	Type of target of an ISO-TP message	PCAN-ISO-TP	"obd_addressing" on page 46
uds_status, PUDS_STATUS_xxx	UDS status codes	PCAN-UDS	"obd_status" on page 37
uds_nrc, PUDS_NRC_xxx	UDS negative response codes	PCAN-UDS	"obd_msgaccess" on page 17

### Structures

Name	Description	API	See also
uds_msg	UDS message	PCAN-UDS	"obd_msg" on page 18

## 3.7 Methods

The methods defined for the class `OBDonUDSApi` are divided into 6 groups of functionality.

Note that these methods are static and can be called in the name of the class, without instantiation.

### ■ Connection

Method	Description
Initialize	Initializes a PCAN-OBDonUDS channel
Uninitialize	Uninitializes a PCAN-OBDonUDS channel

### ■ Configuration

Method	Description
SetValue	Sets a configuration or information value within a PCAN-OBDonUDS channel

### ■ Information

Method	Description
GetValue	Retrieves information from a PCAN-OBDonUDS channel
GetStatus	Retrieves the current bus status of a PCAN-OBDonUDS channel
StatusIsOk	Checks if a PCAN-OBDonUDS status matches an expected result (default is <code>POBDONUDS_STATUS_OK</code> )
GetErrorText	Gets a descriptive text for an error code
ParseResponse_RequestCurrentData	Parses a response to <code>RequestCurrentData</code>
ParseResponse_RequestFreezeFrameData	Parses a response to <code>RequestFreezeFrameData</code>
ParseResponse_RequestConfirmedTroubleCodes	Parses a response to <code>RequestConfirmedTroubleCodes</code>
ParseResponse_ClearTroubleCodes	Parses a response to <code>ClearTroubleCodes</code>
ParseResponse_RequestTestResults	Parses a response to <code>RequestTestResults</code>
ParseResponse_RequestPendingTroubleCodes	Parses a response to <code>RequestPendingTroubleCodes</code>
ParseResponse_RequestControlOperation	Parses a response to <code>RequestControlOperation</code>
ParseResponse_RequestVehicleInformation	Parses a response to <code>RequestVehicleInformation</code>
ParseResponse_RequestPermanentTroubleCodes	Parses a response to <code>RequestPermanentTroubleCodes</code>
ParseResponse_RequestSupportedDTCExtended	Parses a response to <code>RequestSupportedDTCExtended</code>
ParseResponse_RequestDTCExtended	Parses a response to <code>RequestDTCExtended</code>
ParseResponse_RequestDTCForAReadinessGroup	Parses a response to <code>RequestDTCForAReadinessGroup</code>

### ■ Communication

Method	Description
Reset	Resets the receive and transmit queues of a PCAN-OBDonUDS channel
FindOBDonEDS	Tests a channel to detect ECUs responding in OBDonEDS-mode
RequestCurrentData	Requests "Current Powertrain Diagnostic Data" using service \$22-DID
RequestFreezeFrameData	Requests "Powertrain Freeze Frame Data" using service \$19 subfunction \$04
RequestConfirmedTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Confirmed Status" using service \$19 - subfunction \$42
ClearTroubleCodes	Clears/Resets "Emission-Related Diagnostic Information" using service \$14

Method	Description
RequestTestResults	Requests "On-Board Monitoring Test Results for Specific Monitored Systems" using service \$22 - MID
RequestPendingTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Pending Status" using service \$19 - subfunction \$42
RequestControlOperation	Requests "Control of On-Board System, Test, or Component" using service \$31
RequestVehicleInformation	Requests "Vehicle Information" using service \$22 - ITID
RequestPermanentTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Permanent Status" using service \$19 - subfunction \$55
RequestSupportedDTCExtended	Requests "Supported DTCExtendedRecord Information" using service \$19 - subfunction \$1A
RequestDTCExtended	Requests "DTCExtendedDataRecord" using service \$19 - subfunction \$06
RequestDTCForAReadinessGroup	Requests "DTCs for a ReadinessGroup" using service \$19 - subfunction \$56
WaitForService	Handles the communication workflow for a PCAN-OBDonUDS request expecting a single response
WaitForServiceFunctional	Handles the communication workflow for a PCAN-OBDonUDS request expecting multiple responses

#### ■ Message Handling

Method	Description
MsgFree	Deallocates a PCAN-OBDonUDS message
ParsedResponseFree	Deallocates a parsed response

#### ■ Helpers (C# specific methods)

Method	Description
GetByte	Get a byte of a IntPtr pointer in a safe way.
GetData	Get values of an OBDonUDSApi structure in a safe way.

### 3.7.1 Initialize

Initializes a PCAN-OBDonUDS channel.

#### Overloads

Method	Description
obd_status Initialize(cantp_handle channel, cantp_baudrate baudrate, cantp_hwtype hw_type, UInt32 io_port, UInt16 interrupt)	Initializes an OBDonUDS client based on a PCAN-ISO-TP channel which represents a Non-Plug and Play PCAN-Device.
obd_status Initialize(cantp_handle channel, cantp_baudrate baudrate)	Initializes an OBDonUDS client based on a PCAN-ISO-TP channel which represents a Plug and Play PCAN-Device.
obd_status Initialize(cantp_handle channel)	Initializes an OBDonUDS client based on a PCAN-ISO-TP channel which represents a Plug and Play PCAN-Device, using auto-detection of baudrate.

#### Plain function version

"OBDonUDS\_Initialize" on page 137

### 3.7.2 Initialize(cantp\_handle channel, cantp\_baudrate baudrate, cantp\_hwtype hw\_type, UInt32 io\_port, UInt16 interrupt)

Initializes a PCAN-OBDonUDS channel which represents a Non-Plug and Play PCAN-Device.

#### Syntax

#### C#

```
public static extern obd_status Initialize(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_baudrate baudrate,
    [MarshalAs(UnmanagedType.U4)]
    cantp_hwtype hw_type,
    UInt32 io_port,
    UInt16 interrupt);
```

#### Parameters

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.
hw_type	NON PLUG&PLAY: The type of hardware and operation mode (see cantp_hwtype at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
io_port	NON PLUG&PLAY: The I/O address for the parallel port
interrupt	NON PLUG&PLAY: Interrupt number of the parallel port

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the desired channel is already in use.
POBDONUDS_STATUS_UNSUPPORTED_ECUS	Indicates that no ECUs were found during the auto-detection mechanism.
PUDS_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Remarks

As indicated by its name, the Initialize method initiates a cantp\_handle channel, preparing it for communication within the CAN bus connected to it. Calls to the other methods will fail if they are used with a channel handle, different than PCANTP\_HANDLE\_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a POBDonUDS channel means:

- to reserve the channel for the calling application/process
- to detect the baudrate of the CAN bus (if requested with OBD\_BAUDRATE\_AUTODETECT value)
- to detect the obd\_msgprotocol or CAN ID length
- to allocate channel resources, like receive and transmit queues

- to forward initialization to PCAN-UDS API, PCAN-ISO-TP API, and PCAN-Basic API, hence registering/connecting the hardware denoted by the channel handle

The initialization process will fail if an application tries to initialize a PCANTP channel handle that has already been initialized within the same process.

Take into consideration that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT are removed.

**Example**

**C#**

```
obd_status status;
// The Plug & Play Channel is initialized using auto-detection of baudrate
status = OBDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_DNGBUS1,
    (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_AUTODETECT,
    cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, 0x378, 7);

if (!OBDonUDSApi.StatusIsOk(status))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-DNG (Ch-1) was initialized");

// All initialized channels are released
OBDonUDSApi.Uninitialize(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

**See also:**

Uninitialize below, "GetValue" on page 63, " Understanding PCAN-OBDonUDS API" on page 8

**Plain function version:**

"OBDonUDS\_Initialize" on page 137

3.7.3 Initialize(cantp\_handle channel, cantp\_baudrate baudrate)

Initializes a PCAN-OBDonUDS channel which represents a Plug and Play PCAN-Device.

**Syntax**

**C#**

```
public static extern obd_status Initialize(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_baudrate baudrate);
```

**Parameters**

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the desired channel is already in use.
POBDONUDS_STATUS_UNSUPPORTED_ECUS	Indicates that no ECUs were found during the auto-detection mechanism.
PUDS_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

## Remarks

As indicated by its name, the Initialize method initiates a cantp\_handle channel, preparing it for communication within the CAN bus connected to it. Calls to the other methods will fail if they are used with a channel handle, different than PCANTP\_HANDLE\_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a POBDonUDS channel means:

- to reserve the channel for the calling application/process
- to detect the baudrate of the CAN bus (if requested with OBD\_BAUDRATE\_AUTODETECT value)
- to detect the obd\_msgprotocol or CAN ID length
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-UDS API, PCAN-ISO-TP API, and PCAN-Basic API, hence registering/connecting the hardware denoted by the channel handle

The initialization process will fail if an application tries to initialize a PCANTP channel handle that has already been initialized within the same process.

Take into consideration that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT are removed.

## Example

### C#

```
obd_status status;
// The Plug & Play Channel is initialized using auto-detection of baudrate
status = OBDDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_USBBUS1, (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_AUTODETECT);

if (!OBDDonUDSApi.StatusIsOk(status))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-USB (Ch-1) was initialized");

// All initialized channels are released
OBDDonUDSApi.Uninitialize(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

### See also:

Uninitialize below, "GetValue" on page 63, "Understanding PCAN-OBDonUDS API" on page 8

### Plain function version:

"OBDDonUDS\_Initialize" on page 137

### 3.7.4 Initialize(cantp\_handle channel)

Initializes a PCAN-OBDonUDS channel which represents a Plug and Play PCAN-Device, using auto-detection of baudrate.

#### Syntax

#### C#

```
public static extern obd_status Initialize(  
    [MarshalAs(UnmanagedType.U4)]  
    cantp_handle channel);
```

#### Parameters

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the desired channel is already in use.
POBDONUDS_STATUS_UNSUPPORTED_ECUS	Indicates that no ECUs were found during the auto-detection mechanism.
PUDS_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Remarks:

As indicated by its name, the Initialize method initiates a cantp\_handle channel, preparing it for communication within the CAN bus connected to it. Calls to the other methods will fail if they are used with a channel handle, different than PCANTP\_HANDLE\_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a POBDonUDS channel means:

- to reserve the channel for the calling application/process
- to detect the baudrate of the CAN bus
- to detect the obd\_msgprotocol or CAN ID length
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-UDS API, PCAN-ISO-TP API, and PCAN-Basic API, hence registering/connecting the hardware denoted by the channel handle

The initialization process will fail if an application tries to initialize a PCANTP channel handle that has already been initialized within the same process.

Take into consideration that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT are removed.



Example

C#

```
obd_status status;
// The Plug & Play Channel is initialized using auto-detection of baudrate
status = OBDOnUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_USBBUS1);
if (!OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-USB (Ch-1) was initialized");
// All initialized channels are released
OBDOnUDSApi.Uninitialize(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

See also:

Uninitialize below, "GetValue" on page 63, " Understanding PCAN-OBDonUDS API" on page 8

Plain function version:

"OBDonUDS\_Initialize" on page 137

3.7.5 Uninitialize

Uninitializes a PCAN-OBDonUDS channel.

Syntax

C#

```
public static extern obd_status Uninitialize(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel);
```

Parameters

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical error in case of failure is:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application
----------------------------------	---

Remarks

A OBDonUDS channel can be released using one of these possibilities:

- Single-Release: Given a handle of a channel initialized before with the method Initialize. If the given channel can not be found then an error is returned
- Multiple-Release: Giving the handle value PCANTP\_HANDLE\_NONEBUS which instructs the API to search for all channels initialized by the calling application and release them all. This option cause no errors if no hardware were uninitialized

Example

C#

```
obd_status status;
// The Plug & Play Channel (PCAN-USB) is initialized
status = OBDOnUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_USBBUS1,
    (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_500K);

if (!OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-USB (Ch-1) was initialized");

// Release channel
status = OBDOnUDSApi.Uninitialize(cantp_handle.PCANTP_HANDLE_USBBUS1);

if (!OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Uninitialization failed");
else
    Console.WriteLine("PCAN-USB (Ch-1) was released");
```

See also:

"Initialize" on page 52

Plain function version:

"OBDOnUDS\_Uninitialize" on page 138

3.7.6 SetValue

Sets a configuration or information value within a PCAN-OBDOnUDS channel.

Overloads

Method	Description
obd_status SetValue(cantp_handle channel, obd_parameter parameter, IntPtr buffer, UInt32 buffer_size);	Sets a configuration or information value within a PCAN-OBDOnUDS channel.
obd_status SetValue(cantp_handle channel, obd_parameter parameter, ref UInt32 buffer, UInt32 buffer_size);	Sets a configuration or information numeric value within a PCAN-OBDOnUDS channel.
obd_status SetValue(cantp_handle channel, obd_parameter parameter, String buffer, UInt32 buffer_size);	Sets a configuration or information string value within a PCAN-OBDOnUDS channel.
obd_status SetValue(cantp_handle channel, obd_parameter parameter, Byte[] buffer, UInt32 buffer_size);	Sets a configuration or information value as an array of bytes within a PCAN-OBDOnUDS channel.

Plain function version:

"OBDOnUDS\_SetValue" on page 140

### 3.7.7 SetValue(cantp\_handle channel, obd\_parameter parameter, IntPtr buffer, UInt32 buffer\_size)

Sets a configuration or information value within a PCAN-OBDonUDS channel.

#### Syntax

#### C#

```
public static extern obd_status SetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    IntPtr buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to set (see "obd_parameter" on page 39)
buffer	Pointer to the parameter value
buffer_size	Size in bytes of the pointed value

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the parameter is not settable.

#### Remarks

Use the method SetValue to set configuration information or environment values of a POBDonUDS channel. Note that any calls with non OBDonUDS parameters (i.e. obd\_parameter) will be forwarded to PCAN-UDS API, PCAN-ISO-TP API, or PCAN-Basic API.

More information about the parameters and values that can be set can be found in "Detailed Parameters Characteristics" on page 40.

#### See also:

"GetValue" on page 63, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

#### Plain function version:

"OBDonUDS\_SetValue" on page 140

3.7.8 SetValue(cantp\_handle channel, obd\_parameter parameter, ref UInt32 buffer, UInt32 buffer\_size)

Sets a configuration or information numeric value within a PCAN-OBDonUDS channel.

Syntax

C#

```
public static extern obd_status SetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    ref UInt32 buffer,
    UInt32 buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to set (see "obd_parameter" on page 39)
buffer	The numeric parameter value
buffer_size	Size in bytes of the buffer

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the parameter is not settable.

Remarks

Use the method SetValue to set configuration information or environment values of a POBDonUDS channel. Note that any calls with non OBDonUDS parameters (i.e. obd\_parameter) will be forwarded to PCAN-UDS API, PCAN-ISO-TP API, or PCAN-Basic API.

More information about the parameters and values that can be set can be found in "Detailed Parameters Characteristics" on page 40.

See also:

"GetValue" on page 63, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

Plain function version:

"OBDonUDS\_SetValue" on page 140

3.7.9 SetValue(cantp\_handle channel, obd\_parameter parameter, String buffer, UInt32 buffer\_size)

Sets a configuration or information string value within a PCAN-OBDonUDS channel.

Syntax

C#

```
public static extern obd_status SetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    [MarshalAs(UnmanagedType.LPStr, SizeParamIndex = 3)]
    String buffer,
    UInt32 buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to set (see "obd_parameter" on page 39)
buffer	The string parameter value
buffer_size	Size in bytes of the buffer

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the parameter is not settable.

Remarks

Use the method SetValue to set configuration information or environment values of a POBDonUDS channel. Note that any calls with non OBDonUDS parameters (i.e. obd\_parameter) will be forwarded to PCAN-UDS API, PCAN-ISO-TP API, or PCAN-Basic API.

More information about the parameters and values that can be set can be found in "Detailed Parameters Characteristics" on page 40.

See also:

"GetValue" on page 63, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

Plain function version:

"OBDonUDS\_SetValue" on page 140

3.7.10 SetValue(cantp\_handle channel, obd\_parameter parameter, byte[] buffer, UInt32 buffer\_size)

Sets a configuration or information value as an array of bytes within a PCAN-OBDonUDS channel.

Syntax

C#

```
public static extern obd_status SetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
    Byte[] buffer,
    UInt32 buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to set (see "obd_parameter" on page 39)
buffer	The array of bytes parameter value
buffer_size	Size in bytes of the buffer

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the parameter is not settable.

Remarks

Use the method SetValue to set configuration information or environment values of a POBDonUDS channel. Note that any calls with non OBDonUDS parameters (i.e. obd\_parameter) will be forwarded to PCAN-UDS API, PCAN-ISO-TP API, or PCAN-Basic API.

More information about the parameters and values that can be set can be found in "Detailed Parameters Characteristics" on page 40.

Example

The following example shows the use of the method SetValue on the channel PCANTP\_HANDLE\_PCIBUS2 to enable debug mode.

Note: It is assumed that the channel was already initialized.

```
obd_status result;
byte[] debugValue = new byte[1] { OBDOnUDSApi.POBDonUDS_DEBUG_LVL_DEBUG };
// Configure logs to get channel debug information
result = OBDOnUDSApi.SetValue(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    obd_parameter.POBDonUDS_PARAMETER_DEBUG, debugValue, 1);
if (!OBDOnUDSApi.StatusIsOk(result))
    Console.WriteLine("Failed to set value");
else
    Console.WriteLine("Value changed successfully");
```

See also:

"GetValue" below, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

Plain function version:

"OBDOnUDS\_SetValue" on page 140

3.7.11 GetValue

Retrieves an OBDOnUDS client parameter value.

Overloads

Method	Description
obd_status GetValue( cantp_handle channel, obd_parameter parameter, IntPtr buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter value
obd_status GetValue( cantp_handle channel, obd_parameter parameter, StringBuilder buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter string value
obd_status GetValue( cantp_handle channel, obd_parameter parameter, out UInt32 buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter numeric value
obd_status GetValue( cantp_handle channel, obd_parameter parameter, out obd_baudrate buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter baud rate value
obd_status GetValue( cantp_handle channel, obd_parameter parameter, out obd_msgprotocol buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter protocol value
obd_status GetValue( cantp_handle channel, obd_parameter parameter, [Out] Byte[] buffer, UInt32 buffer_size)	Retrieves an OBDOnUDS client parameter value as an array of bytes

Plain function version:

"OBDOnUDS\_GetValue" on page 141

3.7.12 GetValue( cantp\_handle channel, obd\_parameter parameter, IntPtr buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter value.

Syntax

C#

```
public static extern obd_status GetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    IntPtr buffer,
    UInt32 buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Pointer to the parameter value
buffer_size	Size in bytes of the pointed value

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

Plain function version:

"OBDOnUDS\_GetValue" on page 141



### 3.7.13 GetValue( cantp\_handle channel, obd\_parameter parameter, StringBuilder buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter string value.

#### Syntax

#### C#

```
public static extern obd_status GetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    StringBuilder buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

#### Example

#### C#

```
// Get API version
uint BUFFER_SIZE = 300;
StringBuilder buffer = new StringBuilder((int)BUFFER_SIZE);
obd_status status = OBDOnUDSApi.GetValue(cantp_handle.PCANTP_HANDLE_NONEBUS,
    obd_parameter.POBDONUDS_PARAMETER_API_VERSION, buffer, BUFFER_SIZE);
if(OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Api Version {0}", buffer);
else
    Console.WriteLine("Failed to get value");
```

#### See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

#### Plain function version:

"OBDOnUDS\_GetValue" on page 141

3.7.14 GetValue( cantp\_handle channel, obd\_parameter parameter, out UInt32 buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter numeric value.

Syntax

C#

```
public static extern obd_status GetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    out UInt32 buffer,
    UInt32 buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

Plain function version:

"OBDOnUDS\_GetValue" on page 141

### 3.7.15 GetValue( cantp\_handle channel, obd\_parameter parameter, obd\_baudrate buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter baud rate value.

#### Syntax

#### C#

```
public static extern obd_status GetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    out obd_baudrate buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical error in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

#### Example

#### C#

```
obd_baudrate baudrate = (obd_baudrate)0;
obd_status status = OBDOnUDSApi.GetValue(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    obd_parameter.POBDONUDS_PARAMETER_BAUDRATE, out baudrate, sizeof(obd_baudrate));
if (OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Baudrate {0}", baudrate);
else
    Console.WriteLine("Failed to get value");
```

#### See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

#### Plain function version:

"OBDOnUDS\_GetValue" on page 141

### 3.7.16 GetValue( cantp\_handle channel, obd\_parameter parameter, out obd\_msgprotocol buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter protocol value.

#### Syntax

#### C#

```
public static extern obd_status GetValue(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    obd_parameter parameter,
    out obd_msgprotocol buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

#### Example

#### C#

```
obd_msgprotocol canIdLen = (obd_msgprotocol)0;
obd_status status = OBDOnUDSApi.GetValue(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    obd_parameter.POBDONUDS_PARAMETER_CAN_ID_LENGTH, out canIdLen, sizeof(obd_msgprotocol));
if (OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Can Id Length {0}", canIdLen);
else
    Console.WriteLine("Failed to get value");
```

#### See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

#### Plain function version:

"OBDOnUDS\_GetValue" on page 141

### 3.7.17 GetValue( cantp\_handle channel, obd\_parameter parameter, [Out] Byte[] buffer, UInt32 buffer\_size)

Retrieves an OBDOnUDS client parameter value as an array of bytes.

#### Syntax

#### C#

```
public static extern obd_status GetValue(
    [MarshalAs (UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs (UnmanagedType.U4)]
    obd_parameter parameter,
    [MarshalAs (UnmanagedType.LPArray)]
    [Out] Byte[] buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

#### Example

#### C#

```
byte[] numberOfECU = new byte[1];
obd_status status = OBDOnUDSApi.GetValue(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    obd_parameter.POBDONUDS_PARAMETER_AVAILABLE_ECUS, numberOfECU, sizeof(byte));
if (OBDOnUDSApi.StatusIsOk(status))
    Console.WriteLine("Number of OBDOnUDS ECU detected {0}", numberOfECU[0]);
else
    Console.WriteLine("Failed to get value");
```

#### See also:

"SetValue" on page 58, "obd\_parameter" on page 39, "Detailed Parameters Characteristics" on page 40

#### Plain function version:

"OBDOnUDS\_GetValue" on page 141

### 3.7.18 GetStatus

Gets information about the internal CAN bus status of a POBDonUDS channel.

#### Syntax

#### C#

```
public static extern obd_status GetStatus(  
    [MarshalAs(UnmanagedType.U4)]  
    cantp_handle channel);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success, meaning that the internal CAN bus is OK. The typical status are:

PUDS_STATUS_FLAG_BUS_LIGHT	Indicates a bus error within the given channel. The hardware is in bus-light status.
PUDS_STATUS_FLAG_BUS_HEAVY	Indicates a bus error within the given channel. The hardware is in bus-heavy status.
PUDS_STATUS_FLAG_BUS_OFF	Indicates a bus error within the given channel. The hardware is in bus-off status.
POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application

#### Remarks

When the hardware status is bus-off, an application cannot communicate anymore. Consider using the PCAN-Basic property PCAN\_BUSOFF\_AUTORESET which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like bus-off, bus-heavy, and bus-light is to uninitialized and initialize again the channel used. This causes a hardware reset.

#### Example

The following example shows the use of the method GetStatus on the channel PCANTP\_HANDLE\_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized.

```
obd_status result;
// Check the status of the PCI channel
result = OBDOnUDSApi.GetStatus(cantp_handle.PCANTP_HANDLE_PCIBUS1);
switch (result)
{
    case (obd_status)uds_status.PUDS_STATUS_FLAG_BUS_LIGHT:
        Console.WriteLine("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status");
        break;
    case (obd_status)uds_status.PUDS_STATUS_FLAG_BUS_HEAVY:
        Console.WriteLine("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status");
        break;
    case (obd_status)uds_status.PUDS_STATUS_FLAG_BUS_OFF:
        Console.WriteLine("PCAN-PCI (Ch-1): Handling a BUS-OFF status");
        break;
    case obd_status.POBDonUDS_STATUS_OK:
        Console.WriteLine("PCAN-PCI (Ch-1): Status is OK");
        break;
    case obd_status.POBDonUDS_STATUS_NOT_INITIALIZED:
        Console.WriteLine("PCAN-PCI (Ch-1): OBDOnUDS channel not initialized");
        break;
    default:
        // An error occurred
        Console.WriteLine("Failed to retrieve status");
        break;
}
```

See also:

"obd\_status" on page 37

Plain function version:

"OBDOnUDS\_GetStatus" on page 142

3.7.19 StatusIsOk

Checks if a PCAN-OBDOnUDS status matches an expected result.

Overloads

Method	Description
bool StatusIsOk( obd_status status, obd_status status_expected, bool strict_mode)	Checks if a PCAN-OBDOnUDS status matches an expected result, in strict mode or not. Strict mode ensures that bus or extra information are the same.
bool StatusIsOk( obd_status status, obd_status status_expected)	Checks if a PCAN-OBDOnUDS status matches an expected result, in non-strict mode
bool StatusIsOk( obd_status status)	Checks if a PCAN-OBDOnUDS status matches POBDOnUDS_STATUS_OK, in non-strict mode

### 3.7.20 StatusIsOk( obd\_status status, obd\_status status\_expected, bool strict\_mode)

Checks if a PCAN-OBDonUDS status matches an expected result, in strict mode or not.

#### Syntax

#### C#

```
public static extern bool StatusIsOk(
    [MarshalAs(UnmanagedType.U4)]
    obd_status status,
    [MarshalAs(UnmanagedType.U4)]
    obd_status status_expected,
    [MarshalAs(UnmanagedType.I1)]
    bool strict_mode);
```

#### Parameters

Parameter	Description
status	The status to be analyzed (see " obd_status" on page 37).
status_expected	The expected status (see " obd_status" on page 37).
strict_mode	Enable strict mode. Strict mode ensures that bus or extra information are the same.

#### Returns

The return value is true if the status matches the expected parameter.

#### Remarks

When checking an obd\_status, it is preferred to use StatusIsOk instead of comparing it with the == operator because StatusIsOk can remove information flag (in non-strict mode).

#### Example

The following example shows the use of the method StatusIsOk after initializing the channel PCANTP\_HANDLE\_PCIBUS2.

#### C#

```
obd_status result;
result = OBDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_500K);
if (!OBDonUDSApi.StatusIsOk(result, obd_status.POBDonUDS_STATUS_OK, false))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-PCI (Ch-2) was initialized");
```

#### See also:

" obd\_status" on page 37

#### Plain function version:

"OBDonUDS\_StatusIsOk" on page 144



### 3.7.21 StatusIsOk( obd\_status status, obd\_status status\_expected)

Checks if a PCAN-OBDonUDS status matches an expected result, in non-strict mode.

#### Syntax

#### C#

```
public static extern bool StatusIsOk(
    [MarshalAs(UnmanagedType.U4)]
    obd_status status,
    [MarshalAs(UnmanagedType.U4)]
    obd_status status_expected);
```

#### Parameters

Parameter	Description
status	The status to be analyzed (see " obd_status" on page 37).
status_expected	The expected status (see " obd_status" on page 37).

#### Returns

The return value is true if the status matches the expected parameter.

#### Remarks

When checking an obd\_status, it is preferred to use StatusIsOk instead of comparing it with the == operator because StatusIsOk can remove information flag (in non-strict mode).

#### Example

The following example shows the use of the method StatusIsOk after initializing the channel PCANTP\_HANDLE\_PCIBUS2.

#### C#

```
obd_status result;
result = OBDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_500K);
if (!OBDonUDSApi.StatusIsOk(result, obd_status.POBDonUDS_STATUS_OK))
    Console.WriteLine("Initialization failed");
else
    Console.WriteLine("PCAN-PCI (Ch-2) was initialized");
```

#### See also:

" obd\_status" on page 37

#### Plain function version:

"OBDonUDS\_StatusIsOk" on page 144

### 3.7.22 StatusIsOk( obd\_status status)

Checks if a PCAN-OBDonUDS status matches POBDONUDS\_STATUS\_OK, in non-strict mode.

#### Syntax

#### C#

```
public static extern bool StatusIsOk(  
    [MarshalAs(UnmanagedType.U4)]  
    obd_status status);
```

#### Parameters

Parameter	Description
status	The status to be analyzed (see " obd_status" on page 37).

#### Returns

The return value is true if the status matches POBDONUDS\_STATUS\_OK.

#### Remarks

When checking an obd\_status, it is preferred to use StatusIsOk instead of comparing it with the == operator because StatusIsOk can remove information flag (in non-strict mode).

#### Example

The following example shows the use of the method StatusIsOk after initializing the channel PCANTP\_HANDLE\_PCIBUS2.

#### C#

```
obd_status result;  
result = OBDonUDSApi.Initialize(cantp_handle.PCANTP_HANDLE_PCIBUS2, (cantp_baudrate)obd_baudrate.OBD_  
BAUDRATE_500K);  
if (!OBDonUDSApi.StatusIsOk(result))  
    Console.WriteLine("Initialization failed");  
else  
    Console.WriteLine("PCAN-PCI (Ch-2) was initialized");
```

#### See also:

" obd\_status" on page 37

#### Plain function version:

"OBDonUDS\_StatusIsOk" on page 144

### 3.7.23 GetErrorText

Gets a descriptive text for an obd\_status error code.

#### Syntax

#### C#

```
public static extern obd_status GetErrorText(
    [MarshalAs(UnmanagedType.U4)]
    obd_status error_code,
    UInt16 language,
    StringBuilder buffer,
    UInt32 buffer_size);
```

#### Parameters

Parameter	Description
error_code	A obd_status error code (see " obd_status" on page 37)
language	The current languages available for translation are: Neutral (0x00), English (0x09), and French (0x0C)
buffer	String buffer
buffer_size	Buffer size in bytes

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success. The typical error in case of failure is:

POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the function are invalid. Check the parameter 'buffer'; it should point to a char array, big enough to allocate the text for the given error code
--------------------------------------	---

#### Remarks

The Primary Language IDs are codes used by Windows OS from Microsoft, to identify a human language.

The API currently supports the following languages:

Language	Primary Language ID
Neutral (system-dependent)	00h (0)
English	09h (9)
French	0Ch (12)

Note: If the buffer is too small for the resulting text, the error 0x80008000 (PUDS\_STATUS\_MASK\_PCAN|PCAN\_ERROR\_ILLPARAMVAL) is returned. Even when only short texts are being currently returned, a text within this function can have a maximum of 255 characters. For this reason, it is recommended to use a buffer with a length of at least 256 bytes.

#### Example

The following example shows the use of the method GetErrorText to get the description of an error.

The language of the description's text will be the same used by the operating system (if its language is supported; otherwise English is used).

Note: It is assumed that the channel was NOT initialized (to generate an error).

C#

```
StringBuilder buffer = new StringBuilder(256);
obd_status result;
obd_status error_result;
error_result = OBDOnUDSApi.Uninitialize(cantp_handle.PCANTP_HANDLE_USBBUS1);
result = OBDOnUDSApi.GetErrorText(error_result, 0x0, buffer, 256);
if (OBDOnUDSApi.StatusIsOk(result) && !OBDOnUDSApi.StatusIsOk(error_result))
    Console.WriteLine("{0}", buffer);
```

See also:

"obd\_status" on page 37

Plain function version:

"OBDOnUDS\_StatusIsOk" on page 144

3.7.24 ParseResponse\_RequestCurrentData

Parses a response to requested service 22-DID (Request Current Powertrain Diagnostic Data).

Syntax

C#

```
public static extern obd_status ParseResponse_RequestCurrentData(
    [In] ref obd_msg msg_response,
    out obd_request_current_data_response out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported DIDs".

## Remarks

The result must be freed afterwards by calling `ParsedResponseFree`.

If the request was to retrieve supported DIDs, the response will not be parsable.

In case of a negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `RequestCurrentData` and `OBDonUDS_WaitForService`.

## C#

```
obd_msg msg_response_to_parse = new obd_msg();
//.....

// Parse response
obd_request_current_data_response parsed_response = new obd_request_current_data_response();
obd_status status = OBDonUDSApi.ParseResponse_RequestCurrentData(ref msg_response_to_parse, out parsed_response);
if (OBDonUDSApi.StatusIsOk(status))
{
    // Print parsed response
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        if (parsed_response.nb_elements != 0)
        {
            obd_did_object[] vals = new obd_did_object[parsed_response.nb_elements];
            if (OBDonUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 i = 0; i < parsed_response.nb_elements; ++i)
                {
                    Console.WriteLine("-> Element #{0} : ", i + 1);
                    print_did_safe(ref vals[i]);
                }
        }
    }
}
else
{
    Console.WriteLine("Failed status: 0x{0:X}\n", status);
}
// Free message
OBDonUDSApi.ParsedResponseFree(ref parsed_response);
```

## See also:

"obd\_msg" on page 18, "obd\_request\_current\_data\_response" on page 20, "RequestCurrentData" on page 102.

## Plain function version:

"OBDonUDS\_ParseResponse\_RequestCurrentData" on page 146

### 3.7.25 ParseResponse\_RequestFreezeFrameData

Parses a response to requested service 19-04 (Request Powertrain Freeze Frame Data).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_RequestFreezeFrameData(
    [In] ref obd_msg msg_response,
    out obd_request_freeze_frame_data_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using RequestFreezeFrameData and WaitForService.

### C#

```
obd_msg msg_response = new obd_msg();
//.....

// Parse response
obd_request_freeze_frame_data_response parsed_response = new obd_request_freeze_frame_data_response();
obd_status status = OBDonUDSApi.ParseResponse_RequestFreezeFrameData(ref msg_response, out parsed_response);
if (OBDonUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
    {
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    }
    else
    {
        Console.WriteLine("Report type 0x{0:X}, DTC 0x{1:X}, DTC status 0x{2:X}, Record number 0x{3:X}",
            parsed_response.report_type, parsed_response.dtc_number, parsed_response.status_of_dtc,
            parsed_response.record_number);
        if (parsed_response.nb_identifiers != 0)
        {
            obd_did_object[] vals = new obd_did_object[parsed_response.nb_identifiers];
            if (OBDonUDSApi.GetData(ref parsed_response, vals, parsed_response.nb_identifiers))
                for (UInt32 id = 0; id < parsed_response.nb_identifiers; ++id)
                {
                    Console.WriteLine("\n -> Identifier {0} : ", id + 1);
                    Console.WriteLine("Data identifier 0x{0:X}", vals[id].data_identifier);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse : ERROR (0x{0:X})", status);
}
// Free message
OBDonUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_freeze\_frame\_data\_response" on page 21, "RequestFreezeFrameData" on page 104.

### Plain function version:

"OBDonUDS\_ParseResponse\_RequestFreezeFrameData" on page 148

### 3.7.26 ParseResponse\_RequestConfirmedTroubleCodes

Parses a response to requested service 19-42 (Request Emission-Related Diagnostic Trouble Codes with Confirmed Status).

#### Syntax

#### C#

```
public static extern obd_status ParseResponse_RequestConfirmedTroubleCodes(  
    [In] ref obd_msg msg_response,  
    out obd_request_trouble_codes_response out_buffer);
```

#### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

#### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

#### Remarks

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.



## Example

This example shows how to parse a response previously obtained using RequestConfirmedTroubleCodes and WaitForService.

### C#

```
obd_msg msg_response = new obd_msg();
// ...

obd_request_trouble_codes_response parsed_response = new obd_request_trouble_codes_response();
obd_status status = OBDonUDSApi.ParseResponse_RequestConfirmedTroubleCodes(ref msg_response, out parsed_response);
if (OBDonUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("-> Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("-> Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("Report type 0x{0:X2}, Functional group identifier 0x{1:X2}, DTC status availability mask 0x{2:X2}, DTC severity mask 0x{3:X2}, DTC format identifier 0x{4:X2}", parsed_response.report_type,
            parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_severity_mask, parsed_response.DTC_format_identifier);
        if (parsed_response.nb_elements != 0)
        {
            obd_severity_trouble_code[] vals = new obd_severity_trouble_code[parsed_response.nb_elements];
            if (OBDonUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
            for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
            {
                Console.WriteLine("\n -> Element {0} : ", ip + 1);
                Console.WriteLine("DTC identifier 0x{0:X}, Status 0x{1:X2}, Severity 0x{2:X2}",
                    vals[ip].DTC_identifier,
                    vals[ip].status_of_dtc, vals[ip].DTC_severity);
            }
        }
    }
}
else
{
    Console.WriteLine("Parse : ERROR (0x{0:X})", status);
}
// Free message
OBDonUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_trouble\_codes\_response" on page 23, "RequestConfirmedTroubleCodes" on page 105.

### Plain function version:

"OBDonUDS\_ParseResponse\_RequestConfirmedTroubleCodes" on page 149

### 3.7.27 ParseResponse\_ClearTroubleCodes

Parses a response to requested service 14 (Clear/reset Emission-related Diagnostic Information).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_ClearTroubleCodes(
    [In] ref obd_msg msg_response,
    out obd_request_clear_trouble_codes_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

Example

This example shows how to parse a response previously obtained using RequestClearTroubleCodes and WaitForService.

C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_clear_trouble_codes_response parsed_response = new obd_request_clear_trouble_codes_response();
obd_status status = OBDOnUDSApi.ParseResponse_ClearTroubleCodes(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
    {
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    }
    else
    {
        Console.WriteLine("Positive response");
    }
}
else
{
    Console.WriteLine("Parse ERROR (0x{0:X})\n", status);
}
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_clear\_trouble\_codes\_response" on page 24, "ClearTroubleCodes" on page 107.

Plain function version:

"OBDOnUDSApi.ParseResponse\_ClearTroubleCodes" on page 151

3.7.28 ParseResponse\_RequestTestResults

Parses a response to requested service 22-MID (Request On-board Monitoring Test Results for Specific Monitored Systems).

Syntax

C#

```
public static extern obd_status ParseResponse_RequestTestResults(
    [In] ref obd_msg msg_response,
    out obd_request_test_results_response out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

Returns

The return value is an obd\_status code.  
POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported MIDs".

## Remarks

The result must be freed afterwards by calling `ParsedResponseFree`.

If the request was to retrieve supported MIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `RequestTestResults` and `WaitForService`.

### C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_test_results_response parsed_response = new obd_request_test_results_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestTestResults(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}\n", parsed_response.nrc);
    else
    {
        Console.WriteLine("MID {0:X} ", parsed_response.data_identifier);
        if (parsed_response.nb_elements != 0)
        {
            obd_test_data_object[] vals = new obd_test_data_object[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("\n -> Element {0} : ", ip + 1);
                    printTestObject(ref vals[ip]);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})\n", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_test\_results\_response" on page 26, "RequestTestResults" on page 108.

### Plain function version:

"OBDOnUDS\_ParseResponse\_RequestTestResults" on page 152

### 3.7.29 ParseResponse\_RequestPendingTroubleCodes

Parses a response to requested service 19-42 (Request Emission-related Diagnostic Trouble Codes with Pending Status).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_RequestPendingTroubleCodes(  
    [In] ref obd_msg msg_response,  
    out obd_request_trouble_codes_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_trouble_codes_response parsed_response = new obd_request_trouble_codes_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestPendingTroubleCodes(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("Report type 0x{0:X2}, Functional group identifier 0x{1:X2}, DTC status availability mask 0x{2:X2}, DTC severity mask 0x{3:X2}, DTC format identifier 0x{4:X2}",
            parsed_response.report_type, parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_severity_mask, parsed_response.DTC_format_identifier);
        if (parsed_response.nb_elements != 0)
        {
            obd_severity_trouble_code[] vals = new obd_severity_trouble_code[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("\n -> Element {0} : ", ip + 1);
                    Console.WriteLine("DTC identifier 0x{0:X}, Status 0x{1:X2}, Severity 0x{2:X2}", vals[ip].DTC_identifier,
                        vals[ip].status_of_dtc, vals[ip].DTC_severity);
                }
            Console.WriteLine("");
        }
    }
}
else
{
    Console.WriteLine("Parse : ERROR (0x{0:X}): ", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

## See also:

"obd\_msg" on page 18, "obd\_request\_trouble\_codes\_response" on page 23, "RequestPendingTroubleCodes" on page 110.

## Plain function version:

"OBDOnUDS\_ParseResponse\_RequestPendingTroubleCodes" on page 154

### 3.7.30 ParseResponse\_RequestControlOperation

Parses a response to requested service 31 (Request Control of On-Board System, Test, or Component).

#### Syntax

#### C#

```
public static extern obd_status ParseResponse_RequestControlOperation(  
    ref obd_msg msg_response,  
    out obd_request_control_operation_response out_buffer);
```

#### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

#### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported RIDs".

#### Remarks

The result must be freed afterwards by calling ParsedResponseFree.

If the request was to retrieve supported RIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `RequestControlOperation` and `WaitForService`.

### C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_control_operation_response parsed_response = new obd_request_control_operation_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestControlOperation(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("Routine Control Type 0x{0:X2} ", parsed_response.routine_control_type);
        Console.WriteLine("RID 0x{0:X} ", parsed_response.routine_identifier);
        Console.WriteLine("Routine Info 0x{0:X2} ", parsed_response.routine_info);
        if (parsed_response.nb_elements != 0)
        {
            byte[] vals = new byte[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("\n -> Routine Status Element {0} : 0x{1:X2}", ip + 1, vals[ip]);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_control\_operation\_response" on page 27, "RequestControlOperation" on page 112.

### Plain function version:

"OBDOnUDS\_ParseResponse\_RequestControlOperation" on page 155



### 3.7.31 ParseResponse\_RequestVehicleInformation

Parses a response to requested service 22 DID (Request Vehicle Information).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_RequestVehicleInformation(
    [In] ref obd_msg msg_response,
    out obd_request_vehicle_information_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported ITIDs".

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

If the request was to retrieve supported ITIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

Example

This example shows how to parse a response previously obtained using RequestVehicleInformation and WaitForService.

C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_vehicle_information_response parsed_response = new obd_request_vehicle_information_response();
obd_status status = OBDonUDSApi.ParseResponse_RequestVehicleInformation(ref msg_response, out parsed_response);
if (OBDonUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("ITID 0x{0:X}, {1} bytes\n\t\t", parsed_response.data_identifier, parsed_response.nb_elements);
        if (parsed_response.nb_elements != 0)
        {
            byte[] vals = new byte[parsed_response.nb_elements];
            if (OBDonUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 j = 0; j < parsed_response.nb_elements; ++j)
                {
                    Console.WriteLine("{0:X2} ", vals[j]);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})", status);
}
// Free message
OBDonUDSApi.ParsedResponseFree(ref parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_vehicle\_information\_response" on page 28, "RequestVehicleInformation" on page 113.

Plain function version:

"OBDonUDS\_ParseResponse\_RequestVehicleInformation" on page 157

3.7.32 ParseResponse\_RequestPermanentTroubleCodes

Parses a response to requested service 19-55 (Request Emission-related Diagnostic Trouble Codes with Permanent Status).

Syntax

C#

```
public static extern obd_status ParseResponse_RequestPermanentTroubleCodes(
    [In] ref obd_msg msg_response,
    out obd_request_permanent_trouble_codes_response out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

## Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

## Remarks

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using RequestPermanentTroubleCodes and WaitForService.

## C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_permanent_trouble_codes_response parsed_response = new obd_request_permanent_trouble_codes_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestPermanentTroubleCodes(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("-> Report type 0x{0:X2}, Functional group identifier 0x{1:X2}, DTC status availability mask 0x{2:X2}, DTC format identifier 0x{3:X2}", parsed_response.report_type, parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask, parsed_response.DTC_format_identifier);
        if (parsed_response.nb_elements != 0)
        {
            obd_trouble_code[] vals = new obd_trouble_code[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
            for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
            {
                Console.WriteLine("\n -> Element {0} : ", ip + 1);
                Console.WriteLine("DTC identifier 0x{0:X}, Status of DTC 0x{1:X2}", vals[ip].DTC_identifier, vals[ip].status_of_dtc);
            }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

## See also:

"obd\_msg" on page 18, "obd\_request\_permanent\_trouble\_codes\_response" on page 29, "RequestPermanentTroubleCodes" on page 115.

**Plain function version:**

"OBDonUDS\_ParseResponse\_RequestPermanentTroubleCodes" on page 158

3.7.33 ParseResponse\_RequestSupportedDTCExtended

Parses a response to requested service 19-1A (Request Supported DTCExtendedRecord Information).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_RequestSupportedDTCExtended(
    [In] ref obd_msg msg_response,
    out obd_request_supported_dtc_extended_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

### Example

This example shows how to parse a response previously obtained using RequestSupportedDTCExtended and WaitForService.

### C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_supported_dtc_extended_response parsed_response = new obd_request_supported_dtc_extended_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestSupportedDTCExtended(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("-> Report type 0x{0:X2}, DTC status availability mask 0x{1:X2},
        extended DTC data record number 0x{2:X2}", parsed_response.report_type,
        parsed_response.DTC_status_availability_mask, parsed_response.DTC_extended_data_record_number);
        if (parsed_response.nb_elements != 0)
        {
            obd_trouble_code[] vals = new obd_trouble_code[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("\n -> Element {0} : ", ip + 1);
                    Console.WriteLine("DTC identifier 0x{0:X}, Status of DTC 0x{1:X2}", vals[ip].DTC_identifier,
                    vals[ip].status_of_dtc);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_supported\_dtc\_extended\_response" on page 31,  
"RequestSupportedDTCExtended" on page 116.

### Plain function version:

"OBDOnUDS\_ParseResponse\_RequestSupportedDTCExtended" on page 160

## 3.7.34 ParseResponse\_RequestDTCExtended

Parses a response to requested service 19-06 (Request DTCExtendedDataRecord).

### Syntax

### C#

```
public static extern obd_status ParseResponse_RequestDTCExtended(
    [In] ref obd_msg msg_response,
    out obd_request_dtc_extended_response out_buffer);
```

### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

## Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

## Remarks

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using RequestDTCExtended and WaitForService.

### C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_dtc_extended_response parsed_response = new obd_request_dtc_extended_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestDTCExtended(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("-> Report type 0x{0:X2}, DTC identifier 0x{1:X}, DTC status 0x{2:X2},\nDTC extended data record number 0x{3:X2}, {4} bytes", parsed_response.report_type,\n        parsed_response.DTC_identifier, parsed_response.status_of_dtc,\n        parsed_response.DTC_extended_data_record_number, parsed_response.nb_elements);
        Console.WriteLine("\t\t\t");
        if (parsed_response.nb_elements != 0)
        {
            byte[] vals = new byte[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("{0:X2} ", vals[ip]);
                }
            Console.WriteLine("");
        }
    }
}
else
{
    Console.WriteLine("Parse : ERROR {0:X}", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_dtc\_extended\_response" on page 32, "RequestDTCExtended" on page 118.

### Plain function version:

"OBDOnUDS\_ParseResponse\_RequestDTCExtended" on page 161

### 3.7.35 ParseResponse\_RequestDTCForAReadinessGroup

Parses a response to requested service 19-56 (Request DTCs for a ReadinessGroup).

**Syntax**

**C#**

```
public static extern obd_status ParseResponse_RequestDTCForAReadinessGroup(  
    [In] ref obd_msg msg_response,  
    out obd_request_dtc_for_a_readiness_group_response out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

Example

This example shows how to parse a response previously obtained using RequestDTCForAReadinessGroup and WaitForService.

C#

```
obd_msg msg_response = new obd_msg();
// ...

// Parse response
obd_request_dtc_for_a_readiness_group_response parsed_response = new obd_request_dtc_for_a_readiness_group_response();
obd_status status = OBDOnUDSApi.ParseResponse_RequestDTCForAReadinessGroup(ref msg_response, out parsed_response);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Print parsed response
    Console.WriteLine("Response from ECU #{0} : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        Console.WriteLine("Negative response code 0x{0:X}", parsed_response.nrc);
    else
    {
        Console.WriteLine("-> Report type 0x{0:X2}, Functional group identifier 0x{1:X2},
        DTC status availability mask 0x{2:X2}, DTC format identifier 0x{3:X2}, readiness group identifier 0x{4:X2}",
        parsed_response.report_type, parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
        parsed_response.DTC_format_identifier, parsed_response.readiness_group_identifier);
        if (parsed_response.nb_elements != 0)
        {
            obd_trouble_code[] vals = new obd_trouble_code[parsed_response.nb_elements];
            if (OBDOnUDSApi.GetData(ref parsed_response, vals, (Int32)parsed_response.nb_elements))
                for (UInt32 ip = 0; ip < parsed_response.nb_elements; ++ip)
                {
                    Console.WriteLine("\n -> Element {0} : ", ip + 1);
                    Console.WriteLine("DTC identifier 0x{0:X}, Status of DTC 0x{1:X2}", vals[ip].DTC_identifier,
                    vals[ip].status_of_dtc);
                }
        }
    }
}
else
{
    Console.WriteLine("Parse: ERROR (0x{0:X})\n", status);
}
// Free message
OBDOnUDSApi.ParsedResponseFree(ref parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_dtc\_for\_a\_readiness\_group\_response" on page 33, "RequestDTCForAReadinessGroup" on page 119.

Plain function version:

"OBDOnUDS\_ParseResponse\_RequestDTCForAReadinessGroup" on page 163

3.7.36 Reset

Resets the receive and transmit queues of a PCAN-OBDOnUDS channel.

Syntax

C#

```
public static extern obd_status Reset(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client



Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical error in case of failure is:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
----------------------------------	---

Remarks

Calling this method ONLY clears the queues of a channel. A reset of the CAN controller doesn't take place by default (see PCAN-Basic parameter PCAN\_HARD\_RESET\_STATUS).

Example

The following example shows the use of the method Reset on the channel PCANTP\_HANDLE\_PCIBUS1.

Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized.

C#

```
obd_status result;
result = OBDOnUDSApi.Reset(cantp_handle.PCANTP_HANDLE_PCIBUS1);
if (!OBDOnUDSApi.StatusIsOk(result))
    Console.WriteLine("An error occurred");
else
    Console.WriteLine("PCAN-PCI (Ch-1) was reset");
```

See also:

"Uninitialize" on page 57

Plain function version:

"OBDOnUDS\_Reset" on page 165

3.7.37 FindOBDOnEDS

Tests a channel to detect ECUs responding in OBDOnEDS-mode.

Overloads

Method	Description
obd_status FindOBDOnEDS(cantp_handle channel, cantp_baudrate baudrate, cantp_hwtype hw_type, UInt32 io_port, UInt16 interrupt);	Tests a channel based on a PCAN-ISO-TP channel which represents a Non-Plug and Play PCAN-Device, to detect ECUs responding in OBDOnEDS-mode.
obd_status FindOBDOnEDS(cantp_handle channel, cantp_baudrate baudrate)	Tests a channel based on a PCAN-ISO-TP channel which represents a Plug and Play PCAN-Device, to detect ECUs responding in OBDOnEDS-mode.
obd_status FindOBDOnEDS(cantp_handle channel)	Tests a channel based on a PCAN-ISO-TP channel which represents a Non-Plug and Play PCAN-Device, using auto-detection of baudrate, to detect ECUs responding in OBDOnEDS-mode.

Plain function version

" OBDOnUDS\_FindOBDOnEDS" on page 166

3.7.38 FindOBDonEDS(cantp\_handle channel, cantp\_baudrate baudrate, cantp\_hwtype hw\_type, UInt32 io\_port, UInt16 interrupt)

Tests a channel based on a PCAN-ISO-TP channel which represents a Non-Plug and Play PCAN-Device, to detect ECUs responding in OBDonEDS-mode.

Syntax

C#

```
public static extern obd_status FindOBDonEDS(cantp_handle channel,
      cantp_baudrate baudrate,
      cantp_hwtype hw_type,
      UInt32 io_port,
      UInt16 interrupt);
```

Parameters

Parameter	Description
channel	The PCAN-ISO-TP channel handle to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.
hw_type	NON PLUG&PLAY: The type of hardware and operation mode (see cantp_hwtype at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
io_port	NON PLUG&PLAY: The I/O address for the parallel port
Interrupt	NON PLUG&PLAY: Interrupt number of the parallel port

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_UNSUPPORTED\_ECUS is returned if no OBDonEDS ECUs were found.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the research could not take place because the channel was already in use.
PUDS_STATUS_FLAG_PCAN_STATUS	Indicates that the research could not take place. This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

Remarks

This function must be called on a closed channel and leaves the channel closed.

Example

This example tries to find OBDonEDS ECUs on the PCANTP\_HANDLE\_DNGBUS1 channel.

It is assumed that the channel has not been previously initialized.

C#

```
obd_status status = OBDonUDSApi.FindOBDonEDS(cantp_handle.PCANTP_HANDLE_DNGBUS1,
    (cantp_baudrate)obd_baudrate.OBD_BAUDRATE_AUTODETECT, cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, 0x378, 7);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Found OBDonEDS (OBDII) ECUs");
}
else
{
    if (!OBDonUDSApi.StatusIsOk(status, obd_status.POBDonUDS_STATUS_UNSUPPORTED_ECUS, false))
    {
        Console.WriteLine("Unattended status");
    }
}
```

See also:

"Initialize" on page 52

Plain function version:

" OBDonUDS\_FindOBDonEDS" on page 166

3.7.39 FindOBDonEDS(cantp\_handle channel, cantp\_baudrate baudrate)

Tests a channel based on a PCAN-ISO-TP channel which represents a Plug and Play PCAN-Device, to detect ECUs responding in OBDonEDS-mode.

Syntax

C#

```
public static extern obd_status FindOBDonEDS(cantp_handle channel,
    cantp_baudrate baudrate);
```

Parameters

Parameter	Description
channel	The PCAN-ISO-TP channel handle to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_UNSUPPORTED\_ECUS is returned if no OBDonEDS ECUs were found.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the research could not take place because the channel was already in use.
PUDS_STATUS_FLAG_PCAN_STATUS	Indicates that the research could not take place. This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

Remarks

This function must be called on a closed channel and leaves the channel closed.

Example

This example tries to find OBDonEDS ECUs on the PCANTP\_HANDLE\_USBBUS1 channel.

It is assumed that the channel has not been previously initialized.

C#

```
obd_status status = OBDonUDSApi.FindOBDonEDS(cantp_handle.PCANTP_HANDLE_USBBUS1,
(cantp_baudrate)obd_baudrate.OBD_BAUDRATE_AUTODETECT, cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, 0x378, 7);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Found OBDonEDS (OBDII) ECUs");
}
else
{
    if (!OBDonUDSApi.StatusIsOk(status, obd_status.POBDONUDS_STATUS_UNSUPPORTED_ECUS, false))
    {
        Console.WriteLine("Unattended status");
    }
}
```

See also:

"Initialize" on page 52

Plain function version:

"OBDonUDS\_FindOBDonEDS" on page 166

3.7.40 FindOBDonEDS(cantp\_handle channel)

Tests a channel based on a PCAN-ISO-TP channel which represents a Plug and Play PCAN-Device, to detect ECUs responding in OBDonEDS-mode, using auto-detection of baudrate.

Syntax

C#

```
public static extern obd_status FindOBDonEDS(cantp_handle channel);
```

Parameters

Parameter	Description
channel	The PCAN-ISO-TP channel handle to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_UNSUPPORTED\_ECUS is returned if no OBDonEDS ECUs were found.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the research could not take place because the channel was already in use.
PUDS_STATUS_FLAG_PCAN_STATUS	Indicates that the research could not take place. This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

## Remarks

This function must be called on a closed channel and leaves the channel closed.

## Example

This example tries to find OBDonEDS ECUs on the PCANTP\_HANDLE\_USBBUS1 channel.

It is assumed that the channel has not been previously initialized.

## C#

```
obd_status status = OBDonUDSApi.FindOBDonEDS(cantp_handle.PCANTP_HANDLE_USBBUS1);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Found OBDonEDS (OBDII) ECUs");
}
else
{
    if (!OBDonUDSApi.StatusIsOk(status, obd_status.POBDONUDS_STATUS_UNSUPPORTED_ECUS, false))
    {
        Console.WriteLine("Unattended status");
    }
}
```

## See also:

"Initialize" on page 52

## Plain function version:

"OBDonUDS\_FindOBDonEDS" on page 166

### 3.7.41 RequestCurrentData

Request "Current Powertrain Diagnostic Data" using service \$22-DID.

#### Syntax

#### C#

```
public static extern obd_status RequestCurrentData(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs(UnmanagedType.LPArray, ArraySubType = UnmanagedType.U2, SizeParamIndex = 4)]
    obd_DID_t[] data_identifier,
    UInt32 data_identifier_length);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see "obd_netaddrinfo" on page 16)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of DIDs) of the "data_identifier" array

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the number and nature of DIDs passed.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

## Example

This example sends a physically-addressed request "Current Powertrain Diagnostic Data" on channel PCANTP\_HANDLE\_USBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
// Request current data values of 3 DID using physical address scheme (point to point)
obd_DID_t[] DIDs = new obd_DID_t[3] { 0xF415, 0xF401, 0xF405 };
obd_ecu myECU = obd_ecu.POB_D_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestCurrentData(cantp_handle.PCANTP_HANDLE_USBUS1, nai, out msg_request, DIDs, 3);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDOnUDS\_RequestCurrentData" on page 167

### 3.7.42 RequestFreezeFrameData

Sends a request to service 19-04 (Request Powertrain Freeze Frame Data).

#### Syntax

#### C#

```
public static extern obd_status RequestFreezeFrameData(
    [MarshalAs (UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs (UnmanagedType.U4)]
    obd_DTC_t DTC_Number,
    byte record_number);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see "obd_netaddrinfo" on page 16)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
DTC_Number	DTC Mask Record (see " obd_DTC_t" on page 48)
record_number	DTCsSnapshotRecordNumber (see OBDONUDS_DTC_SNAPSHOT_RECORD_xxx at "Definitions" on page 194)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the record_number.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.



## Example

This example sends a physically-addressed request "Powertrain Freeze Frame Data" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
obd_DTC_t dtc_number = 0x00014211;
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestFreezeFrameData(cantp_handle.PCANTP_HANDLE_USBBUS1, nai,
    out msg_request, dtc_number, OBDOnUDSApi.OBDONUDS_DTC_SNAPSHOT_RECORD_FIRST);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDOnUDS\_RequestFreezeFrameData" on page 169

## 3.7.43 RequestConfirmedTroubleCodes

Sends a request to service 19-42 (Request Emission-Related Diagnostic Trouble Codes with Confirmed Status).

## Syntax

## C#

```
public static extern obd_status RequestConfirmedTroubleCodes(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte functional_group_identifier,
    byte DTC_status_mask,
    byte DTC_severity_mask);
```

## Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see <code>cantp_handle</code> at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see <code>obd_netaddrinfo</code> )
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with <code>WaitForServiceXXX</code>
functional_group_identifier	FunctionalGroupIdentifier (see <code>OBDonUDS_EMISSION_SYSTEM_GROUP</code> at "Definitions" on page 194)
DTC_status_mask	DTCStatusMask (see <code>OBDonUDS_DTC_STATUS_CONFIRMED</code> at "Definitions" on page 194)
DTC_severity_mask	DTCSeverityMask (see <code>OBDonUDS_DTC_SEVERITY_CLASS_1</code> at "Definitions" on page 194)

## Returns

The return value is an `obd_status` code. `POBDONUDS_STATUS_OK` is returned on success.

The typical errors in case of failure are:

<code>POBDONUDS_STATUS_NOT_INITIALIZED</code>	Indicates that the given channel was not found in the list of reserved channels of the calling application
<code>POBDONUDS_STATUS_PARAM_INVALID_VALUE</code>	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the <code>functional_group_identifier</code> , the <code>DTC_status_mask</code> , the <code>DTC_severity_mask</code> .

## Remarks

Once the request is sent, to get responses, `WaitForServiceXXX` must be called passing `out_msg_request`.

The `out_msg_request` must be freed afterwards by calling `MsgFree`.

## Example

This example sends a physically-addressed request "Emission-Related Diagnostic Trouble Codes with Confirmed Status" on channel `PCANTP_HANDLE_USBBUS1` to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

## C#

```
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestConfirmedTroubleCodes(cantp_handle.PCANTP_HANDLE_USBBUS1, nai,
    out msg_request, OBDonUDSApi.OBDONUDS_EMISSION_SYSTEM_GROUP, OBDonUDSApi.OBDONUDS_DTC_STATUS_
    CONFIRMED,
    OBDonUDSApi.OBDONUDS_DTC_SEVERITY_CLASS_1);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

**See also:**

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

**Plain function version:**

"OBDonUDS\_RequestConfirmedTroubleCodes" on page 171

3.7.44 ClearTroubleCodes

Sends a request to service 14 (Clear/reset Emission-related Diagnostic Information).

**Syntax**

**C#**

```
public static extern obd_status ClearTroubleCodes(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs(UnmanagedType.U4)]
    obd_DTC_t group_of_DTC);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
group_of_DTC	GroupOfDTC (see " obd_DTC_t" on page 48, OBDonUDS_DTC_xxx at "Definitions" on page 194)

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the group_of_DTC.

**Remarks**

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

**Example**

This example sends a functionally-addressed request "Clear/reset Emission-related Diagnostic Information" on channel PCANTP\_HANDLE\_USBBUS1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C#

```
obd_DTC_t emissionSystemGroup = OBDOnUDSApi.OBDONUDS_DTC_EMISSION_SYSTEM_GROUP;
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.ClearTroubleCodes(cantp_handle.PCANTP_HANDLE_USBBUS1,
    OBDOnUDSApi.OBD_NAI_REQUEST_FUNCTIONAL_11B, out msg_request, emissionSystemGroup);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

Plain function version:

"OBDOnUDS\_ClearTroubleCodes" on page 172

3.7.45 RequestTestResults

Sends a request to service 22-MID (Request On-board Monitoring Test Results for Specific Monitored Systems).

Syntax

C#

```
public static extern obd_status RequestTestResults(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs(UnmanagedType.LPArray, ArraySubType = UnmanagedType.U2, SizeParamIndex = 4)]
    obd_DID_t[] data_identifier,
    UInt32 data_identifier_length);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (MIDs) (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of MIDs) of "data_identifier"

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the data_identifier_length, the data identifiers values.

### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

### Example

This example sends a physically-addressed request "On-board Monitoring Test Results for Specific Monitored Systems" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

### C#

```
obd_DID_t[] mids = new obd_DID_t[1] { 0xF601 };
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestTestResults(cantp_handle.PCANTP_HANDLE_USBBUS1, nai, out msg_request, mids, 1);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

### See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

### Plain function version:

"OBDOnUDS\_RequestTestResults" on page 174

### 3.7.46 RequestPendingTroubleCodes

Sends a request to service 19-42 (Request Emission-related Diagnostic Trouble Codes with Pending Status).

#### Syntax

#### C#

```
public static extern obd_status RequestPendingTroubleCodes(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte functional_group_identifier,
    byte DTC_status_mask,
    byte DTC_severity_mask);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
DTC_status_mask	DTCStatusMask (see OBDONUDS_DTC_STATUS_PENDING at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
DTC_severity_mask	DTCSeverityMask (see OBDONUDS_DTC_SEVERITY_CLASS_1 at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier, the DTC_status_mask, the DTC_severity_mask.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

## Example

This example sends a physically-addressed request "Emission-related Diagnostic Trouble Codes with Pending Status" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
obd_ecu myECU = obd_ecu.POBDECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestPendingTroubleCodes(cantp_handle.PCANTP_HANDLE_USBBUS1,
nai, out msg_request, OBDonUDSApi.OBDONUDS_EMISSION_SYSTEM_GROUP,
OBDonUDSApi.OBDONUDS_DTC_STATUS_PENDING, OBDonUDSApi.OBDONUDS_DTC_SEVERITY_CLASS_1);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDonUDS\_RequestPendingTroubleCodes" on page 176

### 3.7.47 RequestControlOperation

Sends a request to service 31 (Request Control of On-Board System, Test, or Component).

#### Syntax

#### C#

```
public static extern obd_status RequestControlOperation(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte routine_control_type,
    obd_RID_t routine_identifier,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 6)]
    byte[] routine_control_options,
    byte routine_control_options_len);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
routine_control_type	Routine Control Type (see OBDONUDS_ROUTINE_START)
routine_identifier	RID (see " obd_RID_t" on page 49)
routine_control_options	RoutineControlOptionRecord, array of "routine_control_options_len" routine Control Option (byte) (may be NULL)
routine_control_options_len	Length of "routine_control_options" (may be 0)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the routine_control_type.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

#### Example

This example sends a physically-addressed request "Control of On-Board System, Test, or Component" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.



C#

```
obd_RID_t rid = 0xE001;
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestControlOperation(cantp_handle.PCANTP_HANDLE_USBBUS1,
nai, out msg_request, OBDonUDSApi.OBDONUDS_ROUTINE_START, rid, null, 0);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

Plain function version:

"OBDonUDS\_RequestControlOperation" on page 178

3.7.48 RequestVehicleInformation

Sends a request to service 22 DID (Request Vehicle Information).

Syntax

C#

```
public static extern obd_status RequestVehicleInformation(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs(UnmanagedType.LPArray, ArraySubType = UnmanagedType.U2, SizeParamIndex = 4)]
    obd_DID_t[] data_identifier,
    UInt32 data_identifier_length);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (ITIDs) (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of ITIDs) of "data_identifier"

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the data_identifier_length, the ITIDs values.

## Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

## Example

This example sends a functionally-addressed request "Vehicle Information" on channel PCANTP\_HANDLE\_USBBUS1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
obd_DID_t[] VINitid = new obd_DID_t[1] { 0xF802 };
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestVehicleInformation(cantp_handle.PCANTP_HANDLE_USBBUS1,
    OBDOnUDSApi.OBD_NAI_REQUEST_FUNCTIONAL_11B, out msg_request, VINitid, 1);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDOnUDS\_RequestVehicleInformation" on page 180

### 3.7.49 RequestPermanentTroubleCodes

Sends a request to service 19-55 (Request Emission-related Diagnostic Trouble Codes with Permanent Status).

#### Syntax

#### C#

```
public static extern obd_status RequestPermanentTroubleCodes(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte functional_group_identifier);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

Example

This example sends a physically-addressed request "Emission-related Diagnostic Trouble Codes with Permanent Status" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C#

```
obd_ecu myECU = obd_ecu.POBd_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestPermanentTroubleCodes(cantp_handle.PCANTP_HANDLE_USBBUS1,
    nai, out msg_request, OBDonUDSApi.OBDONUDS_EMISSION_SYSTEM_GROUP);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

Plain function version:

"OBDonUDS\_RequestPermanentTroubleCodes" on page 181

3.7.50 RequestSupportedDTCExtended

Sends a request to service 19-1A (Request Supported DTCExtendedRecord Information).

Syntax

C#

```
public static extern obd_status RequestSupportedDTCExtended(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte DTC_extended_DTC_data_record);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
DTC_extended_DTC_data_record	DTC Extended DTC Data Record

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the DTC_extended_DTC_data_record.

## Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

## Example

This example sends a physically-addressed request "Supported DTCExtendedRecord Information" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestSupportedDTCExtended(cantp_handle.PCANTP_HANDLE_USBBUS1,
    nai, out msg_request, 0x90);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDonUDS\_RequestSupportedDTCExtended" on page 183

### 3.7.51 RequestDTCExtended

Sends a request to service 19-06 (Request DTCExtendedDataRecord).

#### Syntax

#### C#

```
public static extern obd_status RequestDTCExtended(
    [MarshalAs (UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    [MarshalAs (UnmanagedType.U4)]
    obd_DTC_t DTC_mask,
    byte DTC_extended_data_record_number);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
DTC_mask	DTC Mask Record (see " obd_DTC_t" on page 48)
DTC_extended_data_record_number	DTC Ext Data Record Number

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the DTC_extended_data_record_number.

#### Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

Example

This example sends a physically-addressed request "DTCExtendedDataRecord" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C#

```
obd_DTC_t mask = 0x00013001;
obd_ecu myECU = obd_ecu.POBDD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestDTCExtended(cantp_handle.PCANTP_HANDLE_USBBUS1,
    nai, out msg_request, mask, 0x92);
if (OBDOnUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

See also:

"WaitForService" on page 121, "WaitForServiceFunctional" on page 123

Plain function version:

"OBDOnUDS\_RequestDTCExtended" on page 185

3.7.52 RequestDTCForAReadinessGroup

Sends a request to service 19-56 (Request DTCs for a ReadinessGroup).

Syntax

C#

```
public static extern obd_status RequestDTCForAReadinessGroup(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    obd_netaddrinfo nai,
    out obd_msg out_msg_request,
    byte functional_group_identifier,
    byte readiness_group_identifier);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP)
readiness_group_identifier	ReadinessGroupIdentifier

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier, the readiness_group_identifier.

## Remarks

Once the request is sent, to get responses, WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling MsgFree.

## Example

This example sends a physically-addressed request "DTCs for a ReadinessGroup" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C#

```
obd_ecu myECU = obd_ecu.POBD_ECU_1;
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestDTCForAReadinessGroup(cantp_handle.PCANTP_HANDLE_USBBUS1,
    nai, out msg_request, OBDonUDSApi.OBDONUDS_EMISSION_SYSTEM_GROUP, 0x02);
if (OBDonUDSApi.StatusIsOk(status))
{
    Console.WriteLine("Request sent with success");
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

## See also:

"WaitForService" on the next page, "WaitForServiceFunctional" on page 123

## Plain function version:

"OBDonUDS\_RequestDTCForAReadinessGroup" on page 186



### 3.7.53 WaitForService

Handles the communication workflow for a PCAN-OBDonUDS request expecting a single response.

#### Syntax

#### C#

```
public static extern obd_status WaitForService(  
    [MarshalAs(UnmanagedType.U4)]  
    cantp_handle channel,  
    [In] ref obd_msg msg_request,  
    out obd_msg out_msg_response,  
    out obd_msg out_msg_request_confirmation);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
msg_request	A sent obd_msg message used as a reference to manage the OBDonUDS service
out_msg_response	An obd_msg structure buffer to store the resulting POBDonUDS response
out_msg_request_confirmation	An obd_msg structure buffer to store the resulting POBDonUDS request confirmation

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers.
POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING	Indicates that the msg_request addressing mode is not physical.
(obd_status)PUDS_STATUS_SERVICE_TIMEOUT_CONFIRMATION	Timeout while waiting for request confirmation (request message loopback).
(obd_status)PUDS_STATUS_SERVICE_TIMEOUT_RESPONSE	Timeout while waiting for response message.
(obd_status)PUDS_STATUS_NO_MESSAGE	Indicates that no matching message was received in the given time.
(obd_status)PUDS_STATUS_NETWORK_ERROR	A network error occurred.
(obd_status)PUDS_STATUS_MESSAGE_BUFFER_ALREADY_USED	The given message buffer is already allocated, user must release the buffer before reusing it.

#### Remarks

The out\_msg\_response and out\_msg\_request\_confirmation must be freed afterwards by calling MsgFree.

## Example

This example sends a physically-addressed request "Current Powertrain Diagnostic Data" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1, then it waits for responses by calling `OBDonUDS_WaitForService`.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

## C#

```
obd_ecu myECU = obd_ecu.POBDD_ECU_1;
obd_DID_t[] DIDs = new obd_DID_t[3] { 0xF415, 0xF401, 0xF405 };
obd_netaddrinfo nai = new obd_netaddrinfo();
nai.protocol = obd_msgprotocol.OBD_MSGPROTOCOL_11BIT;
nai.target_type = obd_addressing.OBD_ADDRESSING_PHYSICAL;
nai.source_addr = (UInt16)uds_address.PUDS_ADDRESS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = (UInt16)myECU;
obd_msg msg_request = new obd_msg();
obd_status status = OBDonUDSApi.RequestCurrentData(cantp_handle.PCANTP_HANDLE_USBBUS1,
    nai, out msg_request, DIDs, 3);
if (OBDonUDSApi.StatusIsOk(status))
{
    // Wait for responses
    obd_msg msg_response = new obd_msg();
    obd_msg msg_request_confirmation = new obd_msg();
    status = OBDonUDSApi.WaitForService(cantp_handle.PCANTP_HANDLE_USBBUS1,
        ref msg_request, out msg_response, out msg_request_confirmation);
    if (OBDonUDSApi.StatusIsOk(status))
    {
        Console.WriteLine("Response received with success");
    }
    else
    {
        Console.WriteLine("An error occurred");
    }
    // Free messages
    OBDonUDSApi.MsgFree(ref msg_response);
    OBDonUDSApi.MsgFree(ref msg_request_confirmation);
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDonUDSApi.MsgFree(ref msg_request);
```

## Plain function version:

"`OBDonUDS_WaitForService`" on page 188

### 3.7.54 WaitForServiceFunctional

Handles the communication workflow for a PCAN-OBDonUDS request expecting multiple responses.

#### Syntax

#### C#

```
public static extern obd_status WaitForServiceFunctional(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [In] ref obd_msg msg_request,
    UInt32 max_msg_count,
    [MarshalAs(UnmanagedType.U1)]
    bool wait_until_timeout,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
    [Out] obd_msg[] out_msg_responses,
    out UInt32 out_msg_count,
    out obd_msg out_msg_request_confirmation);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
msg_request	A sent obd_msg message used as a reference to manage the OBDonUDS service
max_msg_count	Length of the buffer array "out_msg_responses" (max. messages that can be received)
wait_until_timeout	if FALSE, the function is interrupted if "out_msg_count" reaches "max_msg_count".
out_msg_responses	A buffer to store the resulting responses. It must be an array of "max_msg_count" entries (it must have at least a size of max_msg_count * sizeof(obd_msg) bytes)
out_msg_count	Resulting actual number of messages read
out_msg_request_confirmation	An obd_msg structure buffer to store the resulting POBDONUDS request confirmation

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

PUDS\_STATUS\_OVERFLOW indicates success but output responses buffer is too small to hold all responses.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the max msg count.
POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING	Indicates that the msg_request addressing mode is not functional.
PUDS_STATUS_SERVICE_TX_ERROR	Indicates that an error occurred while transmitting the request.
PUDS_STATUS_SERVICE_TIMEOUT_CONFIRMATION	Indicates that timeout was reached while waiting for request confirmation.
PUDS_STATUS_NO_MESSAGE	Indicates that no matching message was received (in the given time).
PUDS_STATUS_NETWORK_ERROR	Indicates that a network error occurred.
PUDS_STATUS_SERVICE_RX_OVERFLOW	Indicates that service received more messages than input buffer expected.
PUDS_STATUS_MESSAGE_BUFFER_ALREADY_USED	Indicates that the given message buffer is already allocated, user must release the buffer before reusing it.

**Remarks**

Each message of out\_msg\_responses and the out\_msg\_request\_confirmation must be freed afterwards by calling MsgFree.

**Example**

This example sends a functionally-addressed request "Vehicle Information" on channel PCANTP\_HANDLE\_USBBUS1, then it waits for responses by calling WaitForServiceFunctional.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

**C#**

```
// Request vehicle identification number (VIN) using functional addressing scheme
obd_DID_t[] VINitid = new obd_DID_t[1] { 0xF802 };
obd_msg msg_request = new obd_msg();
obd_status status = OBDOnUDSApi.RequestVehicleInformation(cantp_hndle.PCANTP_HANDLE_USBBUS1,
    OBDOnUDSApi.OBD_NAI_REQUEST_FUNCTIONAL_11B, out msg_request, VINitid, 1);
if (OBDOnUDSApi.StatusIsOk(status))
{
    // Wait for responses
    obd_msg[] msg_responses = new obd_msg[256];
    UInt32 nb_responses = 0;
    obd_msg msg_request_confirmation = new obd_msg();
    status = OBDOnUDSApi.WaitForServiceFunctional(cantp_handle.PCANTP_HANDLE_USBBUS1,
        ref msg_request, 256, false, msg_responses, out nb_responses, out msg_request_confirmation);
    if (OBDOnUDSApi.StatusIsOk(status))
    {
        Console.WriteLine("{0} Responses received with success", nb_responses);
    }
    else
    {
        Console.WriteLine("An error occurred");
    }
    // Free messages
    OBDOnUDSApi.MsgFree(ref msg_request_confirmation);
    for (UInt32 ir = 0; ir < nb_responses; ++ir)
        OBDOnUDSApi.MsgFree(ref msg_responses[ir]);
}
else
{
    Console.WriteLine("An error occurred");
}
// Free message
OBDOnUDSApi.MsgFree(ref msg_request);
```

**Plain function version:**

"OBDOnUDS\_WaitForServiceFunctional" on page 190

**3.7.55 MsgFree**

Deallocates a PCAN-OBDonUDS message.

**Syntax**

**C#**

```
public static extern obd_status MsgFree(
    ref obd_msg msg_buffer);
```

**Parameters**

Parameter	Description
msg_buffer	An allocated obd_msg structure buffer (see "obd_msg" on page 18)

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the msg_buffer or one of its fields is NULL
(obd_status)PUDS_STATUS_CAUTION_BUFFER_IN_USE	Indicates that the message structure is currently in use, it cannot be deleted

Example

C#

```
obd_status status = OBDOnUDSApi.MsgFree(ref msg_request);
```

Plain function version:

"OBDOnUDS\_MsgFree" on page 192

3.7.56 ParsedResponseFree

Deallocates a parsed response.

Overloads

Method Syntax C#	
public static extern obd_status ParsedResponseFree( ref obd_response_generic response);	
public static extern obd_status ParsedResponseFree( ref obd_request_vehicle_information_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_clear_trouble_codes_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_control_operation_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_current_data_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_dtc_extended_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_dtc_for_a_readiness_group_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_freeze_frame_data_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_permanent_trouble_codes_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_supported_dtc_extended_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_test_results_response response);	
public static extern obd_status ParsedResponseFree( ref obd_request_trouble_codes_response response);	

Parameters

Parameter	Description
response	The parsed response to freed

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success. The typical errors in case of failure are:

POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the msg_buffer or one of its fields is NULL or incorrect.
--------------------------------------	--

## Remarks

For all services, this function is to be used after calls to methods ParseXXXX, to free the structures obd\_request\_XXX\_response.

## Example

### C#

```
obd_request_current_data_response parsed_response;  
// ...  
OBDonUDSApi.ParsedResponseFree(ref parsed_response);
```

### Plain function version:

"OBDonUDS\_ParsedResponseFree" on page 192

## 3.7.57 GetData

C# specific helper methods, examples of how to use the structures IntPtr fields in safe mode, with Marshaling operations.

### Overloads

Method	Description
GetData(ref obd_request_vehicle_information_response parsed_response, byte[] vals, Int32 nb)	Get byte values of an obd_request_vehicle_information_response object
GetData(ref obd_request_current_data_response parsed_response, obd_did_object[] vals, Int32 nb)	Get obd_did_object values of an obd_request_current_data_response object
GetData(ref obd_did_object didObject, byte[] vals, Int32 nb)	Get byte values of an obd_did_object object
GetData(ref obd_request_trouble_codes_response parsed_response, obd_severity_trouble_code[] vals, Int32 nb)	Get obd_severity_trouble_code values of an obd_request_trouble_codes_response object
GetData(ref obd_request_freeze_frame_data_response parsed_response, obd_did_object[] vals, Int32 nb)	Get obd_did_object values of an obd_request_freeze_frame_data_response object
GetData(ref obd_request_test_results_response parsed_response, obd_test_data_object[] vals, Int32 nb)	Get obd_test_data_object values of an obd_request_test_results_response object
GetData(ref obd_request_control_operation_response parsed_response, byte[] vals, Int32 nb)	Get byte values of an obd_request_control_operation_response object
GetData(ref obd_request_permanent_trouble_codes_response parsed_response, obd_trouble_code[] vals, Int32 nb)	Get obd_trouble_code values of an obd_request_permanent_trouble_codes_response object
GetData(ref obd_request_supported_dtc_extended_response parsed_response, obd_trouble_code[] vals, Int32 nb)	Get obd_trouble_code values of an obd_request_supported_dtc_extended_response object
GetData(ref obd_request_dtc_extended_response parsed_response, byte[] vals, Int32 nb)	Get byte values of an obd_request_dtc_extended_response object
GetData(ref obd_request_dtc_for_a_readiness_group_response parsed_response, obd_trouble_code[] vals, Int32 nb)	Get obd_trouble_code values of an obd_request_dtc_for_a_readiness_group_response object

3.7.58 GetData(ref obd\_request\_vehicle\_information\_response parsed\_response, byte[] vals, Int32 nb)

C# specific helper method, example of how to get the byte values of an obd\_request\_vehicle\_information\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_vehicle_information_response parsed_response,
    byte[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 90

3.7.59 GetData(ref obd\_request\_current\_data\_response parsed\_response, obd\_did\_object[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_did\_object values of an obd\_request\_current\_data\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_current_data_response parsed_response,
    obd_did_object[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 77

3.7.60 GetData(ref obd\_did\_object didObject, byte[] vals, Int32 nb)

C# specific helper method, example of how to get the byte values of an obd\_did\_object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_did_object didObject, byte[] vals, Int32 nb)
```

Parameters

Parameter	Description
didObject	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

```
obd_did_object didObject = new obd_did_object();
//...

Console.WriteLine("DID 0x{0:X}, {1} bytes: ", didObject.data_identifier, didObject.size);
if (didObject.size != 0)
{
    byte[] vals = new byte[didObject.size];
    if(OBDOnUDSApi.GetData(ref didObject, vals, (Int32)didObject.size))
        for (UInt32 i = 0; i < didObject.size; ++i)
            Console.Write("{0:x2} ", vals[i]);
}
```



3.7.61 GetData(ref obd\_request\_trouble\_codes\_response parsed\_response, obd\_severity\_trouble\_code[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_severity\_trouble\_code values of an obd\_request\_trouble\_codes\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_trouble_codes_response parsed_response,
    obd_severity_trouble_code[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 86

3.7.62 GetData(ref obd\_request\_freeze\_frame\_data\_response parsed\_response, obd\_did\_object[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_did\_object values of an obd\_request\_freeze\_frame\_data\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_freeze_frame_data_response parsed_response,
    obd_did_object[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 79

3.7.63 GetData(ref obd\_request\_test\_results\_response parsed\_response, obd\_test\_data\_object[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_test\_data\_object values of an obd\_request\_test\_results\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_test_results_response parsed_response,
    obd_test_data_object[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 84

3.7.64 GetData(ref obd\_request\_control\_operation\_response parsed\_response, byte[] vals, Int32 nb)

C# specific helper method, example of how to get the byte values of an obd\_request\_control\_operation\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_control_operation_response parsed_response,
    byte[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

**Returns**

true if ok, false if not ok.

**Example**

**C#**

See "Example" on page 88

3.7.65 GetData(ref obd\_request\_permanent\_trouble\_codes\_response parsed\_response, obd\_trouble\_code[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_trouble\_code values of an obd\_request\_permanent\_trouble\_codes\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

**Syntax**

**C#**

```
public static bool GetData(ref obd_request_permanent_trouble_codes_response parsed_response,
    obd_trouble_code[] vals, Int32 nb)
```

**Parameters**

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

**Returns**

true if ok, false if not ok.

**Example**

**C#**

See "Example" on page 91

3.7.66 GetData(ref obd\_request\_supported\_dtc\_extended\_response parsed\_response, obd\_trouble\_code[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_trouble\_code values of an obd\_request\_supported\_dtc\_extended\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

**Syntax**

**C#**

```
public static bool GetData(ref obd_request_supported_dtc_extended_response parsed_response,
    obd_trouble_code[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 93

3.7.67 GetData(ref obd\_request\_dtc\_extended\_response parsed\_response, byte[] vals, Int32 nb)

C# specific helper method, example of how to get the byte values of an obd\_request\_dtc\_extended\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_dtc_extended_response parsed_response,
    byte[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok.

Example

C#

See "Example" on page 94

3.7.68 GetData(ref obd\_request\_dtc\_for\_a\_readiness\_group\_response parsed\_response, obd\_trouble\_code[] vals, Int32 nb)

C# specific helper method, example of how to get the obd\_trouble\_code values of an obd\_request\_dtc\_for\_a\_readiness\_group\_response object pointed by IntPtr field in safe mode, with Marshaling operations.

Syntax

C#

```
public static bool GetData(ref obd_request_dtc_for_a_readiness_group_response parsed_response,
    obd_trouble_code[] vals, Int32 nb)
```

Parameters

Parameter	Description
parsed_response	object to retrieve values from
vals	allocated buffer where to store values
nb	number of values to retrieve

Returns

true if ok, false if not ok

Example

C#

See "Example" on page 96

### 3.7.69 GetByte

C# specific helper method, example of how to get the byte pointed by the structures IntPtr fields in safe mode, with Marshaling operations.

#### Syntax

##### C#

```
public static Byte GetByte(IntPtr ptr);
```

#### Parameters

Parameter	Description
ptr	The IntPtr pointer to get byte value from

#### Returns

The byte value, or 0 if ptr is null.

#### Example

##### C#

```
obd_msg[] msg_responses = new obd_msg[256];
int i = 0;
// ...
Console.WriteLine("Response number {0} from ECU {1} : service 0x{2:X} NRC 0x{3:X}",
    i + 1,
    msg_responses[i].nai.source_addr,
    OBDOnUDSApi.GetByte(msg_responses[i].links.service_id),
    OBDOnUDSApi.GetByte(msg_responses[i].links.nrc));
```

## 3.8 Functions

The functions of the PCAN-OBDonUDS API are divided in 5 groups of functionalities.

### ■ Connection

Function	Description
OBDonUDS_Initialize	Initializes a PCAN-OBDonUDS channel
OBDonUDS_Uninitialize	Uninitializes a PCAN-OBDonUDS channel

### ■ Configuration

Function	Description
OBDonUDS_SetValue	Sets a configuration or information value within a PCAN-OBDonUDS channel

### ■ Information

Function	Description
OBDonUDS_GetValue	Retrieves information from a PCAN-OBDonUDS channel
OBDonUDS_GetStatus	Retrieves the current bus status of a PCAN-OBDonUDS channel
OBDonUDS_StatusIsOk	Checks if a PCAN-OBDonUDS status matches an expected result (default is POBDONUDS_STATUS_OK)
OBDonUDS_GetErrorText	Gets a descriptive text for an error code
OBDonUDS_ParseResponse_RequestCurrentData	Parses a response to OBDonUDS_RequestCurrentData
OBDonUDS_ParseResponse_RequestFreezeFrameData	Parses a response to OBDonUDS_RequestFreezeFrameData
OBDonUDS_ParseResponse_RequestConfirmedTroubleCodes	Parses a response to OBDonUDS_RequestConfirmedTroubleCodes
OBDonUDS_ParseResponse_ClearTroubleCodes	Parses a response to OBDonUDS_ClearTroubleCodes
OBDonUDS_ParseResponse_RequestTestResults	Parses a response to OBDonUDS_RequestTestResults
OBDonUDS_ParseResponse_RequestPendingTroubleCodes	Parses a response to OBDonUDS_RequestPendingTroubleCodes
OBDonUDS_ParseResponse_RequestControlOperation	Parses a response to OBDonUDS_RequestControlOperation
OBDonUDS_ParseResponse_RequestVehicleInformation	Parses a response to OBDonUDS_RequestVehicleInformation
OBDonUDS_ParseResponse_RequestPermanentTroubleCodes	Parses a response to OBDonUDS_RequestPermanentTroubleCodes
OBDonUDS_ParseResponse_RequestSupportedDTCExtended	Parses a response to OBDonUDS_RequestSupportedDTCExtended
OBDonUDS_ParseResponse_RequestDTCExtended	Parses a response to OBDonUDS_RequestDTCExtended
OBDonUDS_ParseResponse_RequestDTCForAReadinessGroup	Parses a response to OBDonUDS_RequestDTCForAReadinessGroup

### ■ Communication

Function	Description
OBDonUDS_Reset	Resets the receive and transmit queues of a PCAN-OBDonUDS channel
OBDonUDS_FindOBDonEDS	Tests a channel to detect ECUs responding in OBDonEDS-mode
OBDonUDS_RequestCurrentData	Requests "Current Powertrain Diagnostic Data" using service \$22-DID
OBDonUDS_RequestFreezeFrameData	Requests "Powertrain Freeze Frame Data" using service \$19 subfunction \$04
OBDonUDS_RequestConfirmedTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Confirmed Status" using service \$19 - subfunction \$42
OBDonUDS_ClearTroubleCodes	Clears/Resets "Emission-Related Diagnostic Information" using service \$14
OBDonUDS_RequestTestResults	Requests "On-Board Monitoring Test Results for Specific Monitored Systems" using service \$22 - MID

Function	Description
OBDonUDS_RequestPendingTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Pending Status" using service \$19 - subfunction \$42
OBDonUDS_RequestControlOperation	Requests "Control of On-Board System, Test, or Component" using service \$31
OBDonUDS_RequestVehicleInformation	Requests "Vehicle Information" using service \$22 - ITID
OBDonUDS_RequestPermanentTroubleCodes	Requests "Emission-Related Diagnostic Trouble Codes with Permanent Status" using service \$19 - subfunction \$55
OBDonUDS_RequestSupportedDTCEXtended	Requests "Supported DTCEXtendedRecord Information" using service \$19 - subfunction \$1A
OBDonUDS_RequestDTCEXtended	Requests "DTCEXtendedDataRecord" using service \$19 - subfunction \$06
OBDonUDS_RequestDTCForAReadinessGroup	Requests DTCs for a ReadinessGroup Service \$19 - Subfunction \$56
OBDonUDS_WaitForService	Handles the communication workflow for a PCAN-OBDonUDS request expecting a single response
OBDonUDS_WaitForServiceFunctional	Handles the communication workflow for a PCAN-OBDonUDS request expecting multiple responses

## ■ Message Handling

Function	Description
OBDonUDS_MsgFree	Deallocates a PCAN-OBDonUDS message
OBDonUDS_ParsedResponseFree	Deallocates a parsed response



### 3.8.1 OBDonUDS\_Initialize

Initializes a PCAN-OBDonUDS channel.

#### Syntax

C

```
obd_status OBDonUDS_Initialize(  
    cantp_handle channel,  
    cantp_baudrate baudrate,  
    cantp_hwtype hw_type,  
    uint32_t io_port,  
    uint16_t interrupt);
```

C++

```
obd_status OBDonUDS_Initialize(  
    cantp_handle channel,  
    cantp_baudrate baudrate = static_cast<cantp_baudrate>(OBD_BAUDRATE_AUTODETECT),  
    cantp_hwtype hw_type = static_cast<cantp_hwtype>(0),  
    uint32_t io_port = 0,  
    uint16_t interrupt = 0);
```

#### Parameters

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN Hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.
hw_type	NON PLUG&PLAY: The type of hardware and operation mode (see cantp_hwtype at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
io_port	NON PLUG&PLAY: The I/O address for the parallel port
interrupt	NON PLUG&PLAY: Interrupt number of the parallel port

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the desired channel is already in use.
POBDONUDS_STATUS_UNSUPPORTED_ECUS	Indicates that no ECUs were found during the auto-detection mechanism.
PUDS_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Remarks

As indicated by its name, the OBDonUDS\_Initialize function initiates a cantp\_handle channel, preparing it for communication within the CAN bus connected to it. Calls to the other functions will fail if they are used with a channel handle, different than PCANTP\_HANDLE\_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a POBDonUDS channel means:

- to reserve the channel for the calling application/process
- to detect the baudrate of the CAN bus (if requested with OBD\_BAUDRATE\_AUTODETECT value)
- to detect the obd\_msgprotocol or CAN ID length
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-UDS API, PCAN-ISO-TP API, and PCAN-Basic API, hence registering/connecting the hardware denoted by the channel handle

The initialization process will fail if an application tries to initialize a PCANTP channel handle that has already been initialized within the same process.

Take into consideration that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT are removed

The PCAN-OBDonUDS API uses the same function for initializations of both, Plug-And-Play and Not-Plug-And-Play hardware.

The OBDonUDS\_Initialize function has three additional parameters that are only for the connection of Non-Plug-And-Play hardware. With Plug-And-Play hardware, however, only two parameters are to be supplied. The remaining three are not evaluated.

### Example

#### C/C++

```
obd_status status;
// The Plug & Play Channel (PCAN-UDS) is initialized using auto-detection of baudrate
status = OBDonUDS_Initialize(PCANTP_HANDLE_USBBUS1, (cantp_baudrate)OBD_BAUDRATE_AUTODETECT, (cantp_
hwtype) (0), 0, 0);
if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Initialization failed\n");
else
    printf("PCAN-USB (Ch-1) was initialized\n");
// All initialized channels are released
OBDonUDS_Uninitialize(PCANTP_HANDLE_NONEBUS);
```

#### See also:

OBDonUDS\_Uninitialize below, "OBDonUDS\_GetValue" on page 141, " Understanding PCAN-OBDonUDS API" on page 8

#### Class-method version:

"Initialize" on page 52

## 3.8.2 OBDonUDS\_Uninitialize

Uninitializes a PCAN-OBDonUDS channel.

### Syntax

#### C/C++

```
obd_status OBDonUDS_Uninitialize(
    cantp_handle channel);
```

## Parameters

Parameter	Description
channel	The handle of the PCAN-ISO-TP channel (see <code>cantp_handle</code> at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

## Returns

The return value is an `obd_status` code. `POBDONUDS_STATUS_OK` is returned on success.

The typical error in case of failure is:

<code>POBDONUDS_STATUS_NOT_INITIALIZED</code>	Indicates that the given channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application
---	---

## Remarks

A `OBDonUDS` channel can be released using one of these possibilities:

- Single-Release: Given a handle of a channel initialized before with the function `OBDonUDS_Initialize`. If the given channel can not be found then an error is returned
- Multiple-Release: Giving the handle value `PCANTP_HANDLE_NONEBUS` which instructs the API to search for all channels initialized by the calling application and release them all. This option cause no errors if no hardware were uninitialized

## Example

### C/C++

```
obd_status status;
// The Plug & Play Channel (PCAN-USB) is initialized
status = OBDonUDS_Initialize(PCANTP_HANDLE_USBBUS1, (cantp_baudrate)OBD_BAUDRATE_500K,
                             (cantp_hwtype)(0), 0, 0);

if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Initialization failed\n");
else
    printf("PCAN-USB (Ch-1) was initialized\n");

// Release channel
status = OBDonUDS_Uninitialize(PCANTP_HANDLE_USBBUS1);
if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Uninitialization failed\n");
else
    printf("PCAN-USB (Ch-1) was released\n");
```

## See also:

"`OBDonUDS_Initialize`" on page 137

## Class-method version:

"`Uninitialize`" on page 57

### 3.8.3 OBDonUDS\_SetValue

Sets a configuration or information value within a PCAN-OBDonUDS channel.

**Syntax**

**C/C++**

```
obd_status OBDonUDS_SetValue(
    cantp_handle channel,
    obd_parameter parameter,
    void* buffer,
    uint32_t buffer_size);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to set (see "obd_parameter" on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical error in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the parameter is not settable.

**Remarks**

Use the function OBDonUDS\_SetValue to set configuration information or environment values of a POBDonUDS channel. Note that any calls with non OBDonUDS parameters (i.e. obd\_parameter) will be forwarded to PCAN-UDS API, PCAN-ISO-TP API, or PCAN-Basic API.

More information about the parameters and values that can be set can be found in "Detailed Parameters Characteristics" on page 40.

Example

The following example shows the use of the function `OBDonUDS_SetValue` on the channel `PCANTP_HANDLE_PCIBUS2` to enable debug mode.

Note: It is assumed that the channel was already initialized.

C/C++

```
obd_status status;
uint8_t buffer = POBDONUDS_DEBUG_LVL_DEBUG;
// Configure logs to get channel debug information
status = OBDonUDS_SetValue(PCANTP_HANDLE_PCIBUS2, POBDONUDS_PARAMETER_DEBUG, &buffer,
    sizeof(uint8_t));
if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Failed to set value\n");
else
    printf("Value changed successfully\n");
```

See also:

"`OBDonUDS_GetValue`" below, "`obd_parameter`" on page 39, "Detailed Parameters Characteristics" on page 40

Class-method version:

"`SetValue`" on page 58

3.8.4 OBDonUDS\_GetValue

Retrieves information from a PCAN-OBDonUDS channel.

Syntax

C/C++

```
obd_status OBDonUDS_GetValue(
    cantp_handle channel,
    obd_parameter parameter,
    void* buffer,
    uint32_t buffer_size);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see <code>cantp_handle</code> at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
parameter	The parameter to get (see " <code>obd_parameter</code> " on page 39)
buffer	Buffer for the parameter value
buffer_size	Size in bytes of the buffer

Returns

The return value is an `obd_status` code. `POBDONUDS_STATUS_OK` is returned on success.

The typical error in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
POBDONUDS_STATUS_HANDLE_INVALID	Indicates that the given channel is not compatible with the parameter.
POBDONUDS_STATUS_PARAM_INVALID_TYPE	Indicates that the given buffer or its size is not correct.
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameter is not correct.

Example

The following example shows the use of the function `OBDonUDS_GetValue` on the channel `PCANTP_HANDLE_PCIBUS2` to retrieve the number of connected and responding ECUs on the CAN bus. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized.

C/C++

```
uint8_t numberOfECU = 0;
obd_status status = OBDonUDS_GetValue(PCANTP_HANDLE_PCIBUS2, POBDONUDS_PARAMETER_AVAILABLE_ECUS,
&numberOfECU, sizeof(numberOfECU));
if(OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Number of OBDonUDS ECU detected %d\n", numberOfECU);
else
    printf("Failed to get value\n");
```

See also:

"`OBDonUDS_SetValue`" on page 140, "`obd_parameter`" on page 39, "Detailed Parameters Characteristics" on page 40

Class-method version:

"`GetValue`" on page 63

3.8.5 OBDonUDS\_GetStatus

Gets information about the internal CAN bus status of a `POBDonUDS` channel.

Syntax

C/C++

```
obd_status OBDonUDS_GetStatus(
    cantp_handle channel);
```

Parameters

Parameter	Description
channel	A <code>PCANTP</code> channel handle representing an <code>OBDonUDS</code> client (see <code>cantp_handle</code> at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

Returns

The return value is an `obd_status` code. `POBDONUDS_STATUS_OK` is returned on success, meaning that the internal CAN bus is OK.

The typical status are:

PUDS_STATUS_FLAG_BUS_LIGHT	Indicates a bus error within the given channel. The hardware is in bus-light status.
PUDS_STATUS_FLAG_BUS_HEAVY	Indicates a bus error within the given channel. The hardware is in bus-heavy status.
PUDS_STATUS_FLAG_BUS_OFF	Indicates a bus error within the given channel. The hardware is in bus-off status.
POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application

### Remarks

When the hardware status is bus-off, an application cannot communicate anymore. Consider using the PCAN-Basic property `PCAN_BUSOFF_AUTORESET` which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like bus-off, bus-heavy, and bus-light is to uninitialized and initialize again the channel used. This causes a hardware reset.

### Example

The following example shows the use of the function `OBDonUDS_GetStatus` on the channel `PCANTP_HANDLE_PCIBUS1`. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized.

### C/C++

```
obd_status status;
// Check the status of the PCI channel
status = OBDonUDS_GetStatus(PCANTP_HANDLE_PCIBUS1);
switch (status)
{
    case (obd_status)PUDS_STATUS_FLAG_BUS_LIGHT:
        printf("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status\n");
        break;
    case (obd_status)PUDS_STATUS_FLAG_BUS_HEAVY:
        printf("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status\n");
        break;
    case (obd_status)PUDS_STATUS_FLAG_BUS_OFF:
        printf("PCAN-PCI (Ch-1): Handling a BUS-OFF status\n");
        break;
    case POBDONUDS_STATUS_OK:
        printf("PCAN-PCI (Ch-1): Status is OK\n");
        break;
    case POBDONUDS_STATUS_NOT_INITIALIZED:
        printf("PCAN-PCI (Ch-1): OBDonUDS channel not initialized\n");
        break;
    default:
        // An error occurred
        printf("Failed to retrieve status\n");
        break;
}
```

### See also:

"obd\_status" on page 37

### Class-method version:

"GetStatus" on page 70

### 3.8.6 OBDonUDS\_StatusIsOk

Checks if a PCAN-OBDonUDS status matches an expected result (default is POBDONUDS\_STATUS\_OK).

#### Syntax

##### C

```
bool OBDonUDS_StatusIsOk(
    const obd_status status,
    const obd_status status_expected,
    bool strict_mode);
```

##### C++

```
bool OBDonUDS_StatusIsOk(
    const obd_status status,
    const obd_status status_expected = POBDONUDS_STATUS_OK,
    bool strict_mode =0);
```

#### Parameters

Parameter	Description
status	The status to be analyzed (see " obd_status" on page 37).
status_expected	The expected status (see " obd_status" on page 37). The default is POBDONUDS_STATUS_OK.
strict_mode	Enable strict mode (default is false). Strict mode ensures that bus or extra information are the same.

#### Returns

The return value is true if the status matches the expected parameter.

#### Remarks

When checking an obd\_status, it is preferred to use OBDonUDS\_StatusIsOk instead of comparing it with the == operator because OBDonUDS\_StatusIsOk can remove information flag (in non-strict mode).

#### Example

The following example shows the use of the function OBDonUDS\_StatusIsOk after initializing the channel PCANTP\_HANDLE\_PCIBUS2.

##### C/C++

```
obd_status status;
status = OBDonUDS_Initialize(PCANTP_HANDLE_PCIBUS2, (cantp_baudrate)OBD_BAUDRATE_500K,
    (cantp_hwtype)0, 0, 0);
if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("Initialization failed\n");
else
    printf("PCAN-PCI (Ch-2) was initialized\n");
```

#### See also:

" obd\_status" on page 37

#### Class-method version:

"StatusIsOk" on page 71



### 3.8.7 OBDOnUDS\_GetErrorText

Gets a descriptive text for an obd\_status error code.

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_GetErrorText(  
    obd_status error_code,  
    uint16_t language,  
    char* buffer,  
    uint32_t buffer_size);
```

#### Parameters

Parameter	Description
error_code	A obd_status error code (see " obd_status" on page 37)
language	The current languages available for translation are: Neutral (0x00), English (0x09), and French (0x0C)
buffer	Buffer for a null terminated char array
buffer_size	Buffer size in bytes

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success. The typical error in case of failure is:

POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the function are invalid. Check the parameter 'buffer'; it should point to a char array, big enough to allocate the text for the given error code
--------------------------------------	---

#### Remarks

The Primary Language IDs are codes used by Windows OS from Microsoft, to identify a human language.

The API currently supports the following languages:

Language	Primary Language ID
Neutral (system-dependent)	00h (0)
English	09h (9)
French	0Ch (12)

Note: If the buffer is too small for the resulting text, the error 0x80008000 (PUDS\_STATUS\_MASK\_PCAN|PCAN\_ERROR\_ILLPARAMVAL) is returned. Even when only short texts are being currently returned, a text within this function can have a maximum of 255 characters. For this reason, it is recommended to use a buffer with a length of at least 256 bytes.

#### Example

The following example shows the use of the function OBDOnUDS\_GetErrorText to get the description of an error.

The language of the description's text will be the same used by the operating system (if its language is supported; otherwise English is used).

Note: It is assumed that the channel was NOT initialized (to generate an error).

```
char str_msg[256];
obd_status status;
obd_status error_result;
error_result = OBDOnUDS_Uninitialize(PCANTP_HANDLE_USBBUS1);
status = OBDOnUDS_GetErrorText(error_result, 0x0, str_msg, 256);
if (OBDOnUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false) &&
    !OBDOnUDS_StatusIsOk(error_result, POBDONUDS_STATUS_OK, false))
    printf("%s\n", str_msg);
```

See also:

"obd\_status" on page 37

Class-method version:

"GetErrorText" on page 75

3.8.8 OBDOnUDS\_ParseResponse\_RequestCurrentData

Parses a response to requested service 22-DID (Request Current Powertrain Diagnostic Data).

Syntax

C/C++

```
obd_status OBDOnUDS_ParseResponse_RequestCurrentData(
    obd_msg* msg_response,
    obd_request_current_data_response* out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported DIDs".

## Remarks

The result must be freed afterwards by calling `OBDonUDS_ParsedResponseFree`.

If the request was to retrieve supported DIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestCurrentData` and `OBDonUDS_WaitForService`.

## C/C++

```
obd_msg msg_response_to_parse;
//...

// Parse response
obd_request_current_data_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestCurrentData(&msg_response_to_parse, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        for (uint32_t i = 0; i < parsed_response.nb_elements; ++i)
        {
            printf("Element #d : DID 0x%04X, %02d bytes: [", i + 1,
                parsed_response.elements[i].data_identifier, parsed_response.elements[i].size);
            for (uint32_t j = 0; j < parsed_response.elements[i].size; ++j)
            {
                printf("%02x ", parsed_response.elements[i].data[j]);
            }
            printf("]\n");
        }
    }
}
else
{
    printf("Failed status: 0x%08X\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

## See also:

"obd\_msg" on page 18, "obd\_request\_current\_data\_response" on page 20, "OBDonUDS\_RequestCurrentData" on page 167.

## Class-method version:

"ParseResponse\_RequestCurrentData" on page 76

### 3.8.9 OBDonUDS\_ParseResponse\_RequestFreezeFrameData

Parses a response to requested service 19-04 (Request Powertrain Freeze Frame Data).

#### Syntax

#### C/C++

```
obd_status OBDonUDS_ParseResponse_RequestFreezeFrameData(  
    obd_msg* msg_response,  
    obd_request_freeze_frame_data_response* out_buffer);
```

#### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

#### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

#### Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestFreezeFrameData` and `OBDonUDS_WaitForService`.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_freeze_frame_data_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestFreezeFrameData(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
    {
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    }
    else
    {
        printf("Report type 0x%02X, DTC 0x%06X, DTC status 0x%02X, Record number 0x%02X",
            parsed_response.report_type, parsed_response.dtc_number, parsed_response.status_of_dtc,
            parsed_response.record_number);
        for (uint32_t id = 0; id < parsed_response.nb_identifiers; ++id)
        {
            printf("\n Identifier %d : ", id + 1);
            printf("Data identifier 0x%04X", parsed_response.identifiers[id].data_identifier);
            for (uint32_t idata = 0; idata < parsed_response.identifiers[id].size; ++idata)
            {
                printf(" %02X", parsed_response.identifiers[id].data[idata]);
            }
            printf("\n");
        }
    }
}
else
{
    printf("Parse : ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)<parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_freeze\_frame\_data\_response" on page 21, "OBDonUDS\_RequestFreezeFrameData" on page 169.

### Class-method version:

"ParseResponse\_RequestFreezeFrameData" on page 78

## 3.8.10 OBDonUDS\_ParseResponse\_RequestConfirmedTroubleCodes

Parses a response to requested service 19-42 (Request Emission-Related Diagnostic Trouble Codes with Confirmed Status).

### Syntax

### C/C++

```
obd_status OBDonUDS_ParseResponse_RequestConfirmedTroubleCodes(
    obd_msg* msg_response,
    obd_request_trouble_codes_response* out_buffer);
```

### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

### Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

### Example

This example shows how to parse a response previously obtained using OBDonUDS\_RequestConfirmedTroubleCodes and OBDonUDS\_WaitForService.

### C/C++

```
obd_msg msg_response;
//...

obd_request_trouble_codes_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestConfirmedTroubleCodes(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, Functional group identifier 0x%02X, DTC status availability mask 0x%02X,
            DTC severity mask 0x%02X, DTC format identifier 0x%02X",
            parsed_response.report_type, parsed_response.functional_group_identifier,
            parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_severity_mask, parsed_response.DTC_format_identifier);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("DTC identifier 0x%06X, Status 0x%02X, Severity 0x%02X",
                parsed_response.elements[ip].DTC_identifier,
                parsed_response.elements[ip].status_of_dtc, parsed_response.elements[ip].DTC_severity);
        }
        printf("\n");
    }
}
else
{
    printf("Parse : ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

**See also:**

"obd\_msg" on page 18, "obd\_request\_trouble\_codes\_response" on page 23, "OBDonUDS\_RequestConfirmedTroubleCodes" on page 171.

**Class-method version:**

"ParseResponse\_RequestConfirmedTroubleCodes" on page 80

3.8.11 OBDonUDS\_ParseResponse\_ClearTroubleCodes

Parses a response to requested service 14 (Clear/reset Emission-related Diagnostic Information).

**Syntax**

**C/C++**

```
obd_status  OBDonUDS_ParseResponse_ClearTroubleCodes(
    obd_msg* msg_response,
    obd_request_clear_trouble_codes_response* out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestClearTroubleCodes` and `OBDonUDS_WaitForService`.

C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_clear_trouble_codes_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_ClearTroubleCodes(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
    {
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    }
    else
    {
        printf("Positive response\n");
    }
}
else
{
    printf("Parse ERROR (0x%X)\n", status);
}
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_clear\_trouble\_codes\_response" on page 24, "OBDonUDS\_ClearTroubleCodes" on page 172.

Class-method version:

"ParseResponse\_ClearTroubleCodes" on page 82

3.8.12 OBDonUDS\_ParseResponse\_RequestTestResults

Parses a response to requested service 22-MID (Request On-board Monitoring Test Results for Specific Monitored Systems).

Syntax

C/C++

```
obd_status OBDonUDS_ParseResponse_RequestTestResults(
    obd_msg* msg_response,
    obd_request_test_results_response* out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result



### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported MIDs".

### Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

If the request was to retrieve supported MIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

### Example

This example shows how to parse a response previously obtained using OBDonUDS\_RequestTestResults and OBDonUDS\_WaitForService.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_test_results_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestTestResults(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("MID 0x%X ", parsed_response.data_identifier);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("TID 0x%02X Unit/Scaling 0x%02X Test Val. 0x%02X 0x%02X Min.Limit 0x%02X 0x%02X Max.Limit 0x%02X 0x%02X ",
                parsed_response.elements[ip].test_identifier,
                parsed_response.elements[ip].unit_and_scaling,
                parsed_response.elements[ip].test_value[0], parsed_response.elements[ip].test_value[1],
                parsed_response.elements[ip].min_test_limit[0], parsed_response.elements[ip].min_test_limit[1],
                parsed_response.elements[ip].max_test_limit[0], parsed_response.elements[ip].max_test_limit[1]);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

**See also:**

"obd\_msg" on page 18, "obd\_request\_test\_results\_response" on page 26, "OBDonUDS\_ParseResponse\_RequestTestResults" on page 152.

**Class-method version:**

"ParseResponse\_RequestTestResults" on page 83

3.8.13 OBDonUDS\_ParseResponse\_RequestPendingTroubleCodes

Parses a response to requested service 19-42 (Request Emission-related Diagnostic Trouble Codes with Pending Status).

**Syntax**

**C/C++**

```
obd_status  OBDonUDS_ParseResponse_RequestPendingTroubleCodes(  
    obd_msg* msg_response,  
    obd_request_trouble_codes_response* out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

Example

```
obd_msg msg_response;
//...

// Parse response
obd_request_trouble_codes_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDOnUDS_ParseResponse_RequestPendingTroubleCodes(&msg_response, &parsed_response);
if (OBDOnUDS_StatusIsOk(status, POBDOnUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, Functional group identifier 0x%02X, DTC status availability mask 0x%02X,
            DTC severity mask 0x%02X, DTC format identifier 0x%02X",
            parsed_response.report_type, parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_severity_mask, parsed_response.DTC_format_identifier);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("DTC identifier 0x%06X, Status 0x%02X, Severity 0x%02X", parsed_response.elements[ip].DTC_identifier,
                parsed_response.elements[ip].status_of_dtc, parsed_response.elements[ip].DTC_severity);
        }
        printf("\n");
    }
}
else
{
    printf("Parse : ERROR (0x%X): ", status);
}

// Free message
OBDOnUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_trouble\_codes\_response" on page 23, "OBDOnUDS\_ParseResponse\_RequestPendingTroubleCodes" on the previous page.

Class-method version:

"ParseResponse\_RequestPendingTroubleCodes" on page 85

3.8.14 OBDOnUDS\_ParseResponse\_RequestControlOperation

Parses a response to requested service 31 (Request Control of On-Board System, Test, or Component).

Syntax

C/C++

```
obd_status OBDOnUDS_ParseResponse_RequestControlOperation(
    obd_msg* msg_response,
    obd_request_control_operation_response* out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported RIDs".

## Remarks

The result must be freed afterwards by calling `OBDonUDS_ParsedResponseFree`.

If the request was to retrieve supported RIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestControlOperation` and `OBDonUDS_WaitForService`.

## C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_control_operation_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestControlOperation(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Routine Control Type 0x%02X ", parsed_response.routine_control_type);
        printf("RID 0x%X ", parsed_response.routine_identifier);
        printf("Routine Info 0x%02X ", parsed_response.routine_info);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Routine Status Element %d : 0x%02X", ip + 1, parsed_response.elements[ip]);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

## See also:

"obd\_msg" on page 18, "obd\_request\_control\_operation\_response" on page 27, "OBDonUDS\_RequestControlOperation" on page 178.

**Class-method version:**

"ParseResponse\_RequestControlOperation" on page 87

**3.8.15 OBDONUDS\_ParseResponse\_RequestVehicleInformation**

Parses a response to requested service 22 DID (Request Vehicle Information).

**Syntax**

**C/C++**

```
obd_status  OBDONUDS_ParseResponse_RequestVehicleInformation(
    obd_msg* msg_response,
    obd_request_vehicle_information_response* out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)
POBDONUDS_STATUS_NOT_PARSABLE	Indicates that the response is not parsable because it matches a request of type "get supported ITIDs".

**Remarks**

The result must be freed afterwards by calling OBDONUDS\_ParsedResponseFree.

If the request was to retrieve supported ITIDs, the response will not be parsable.

In case of negative response, its NRC is available as a part of the parsed response.

Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestVehicleInformation` and `OBDonUDS_WaitForService`.

C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_vehicle_information_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestVehicleInformation(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf(" ITID 0x%X, %d bytes\n\t\t", parsed_response.data_identifier, parsed_response.nb_elements);
        for (uint32_t j = 0; j < parsed_response.nb_elements; ++j)
        {
            printf("%02X ", parsed_response.elements[j]);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}

// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

See also:

"obd\_msg" on page 18, "obd\_request\_vehicle\_information\_response" on page 28, "OBDonUDS\_RequestVehicleInformation" on page 180.

Class-method version:

"ParseResponse\_RequestVehicleInformation" on page 89

3.8.16 OBDonUDS\_ParseResponse\_RequestPermanentTroubleCodes

Parses a response to requested service 19-55 (Request Emission-related Diagnostic Trouble Codes with Permanent Status).

Syntax

C/C++

```
obd_status OBDonUDS_ParseResponse_RequestPermanentTroubleCodes(
    obd_msg* msg_response,
    obd_request_permanent_trouble_codes_response* out_buffer);
```

Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

### Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

### Example

This example shows how to parse a response previously obtained using OBDonUDS\_RequestPermanentTroubleCodes and OBDonUDS\_WaitForService.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_permanent_trouble_codes_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestPermanentTroubleCodes(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, Functional group identifier 0x%02X, DTC status availability mask 0x%02X,\n",
            parsed_response.report_type, parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_format_identifier);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("DTC identifier 0x%06X, Status of DTC 0x%02X", parsed_response.elements[ip].DTC_identifier,
                parsed_response.elements[ip].status_of_dtc);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_permanent\_trouble\_codes\_response" on page 29, "OBDonUDS\_RequestPermanentTroubleCodes" on page 181.

### Class-method version:

"ParseResponse\_RequestPermanentTroubleCodes" on page 90

### 3.8.17 OBDOnUDS\_ParseResponse\_RequestSupportedDTCExtended

Parses a response to requested service 19-1A (Request Supported DTCExtendedRecord Information).

**Syntax**

**C/C++**

```
obd_status OBDOnUDS_ParseResponse_RequestSupportedDTCExtended(
    obd_msg* msg_response,
    obd_request_supported_dtc_extended_response* out_buffer);
```

**Parameters**

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

**Returns**

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

**Remarks**

The result must be freed afterwards by calling OBDOnUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.



## Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestSupportedDTCEExtended` and `OBDonUDS_WaitForService`.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_supported_dtc_extended_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestSupportedDTCEExtended(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, DTC status availability mask 0x%02X, extended DTC data record number 0x%02X",
            parsed_response.report_type, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_extended_data_record_number);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("DTC identifier 0x%06X, Status of DTC 0x%02X",
                parsed_response.elements[ip].DTC_identifier, parsed_response.elements[ip].status_of_dtc);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_supported\_dtc\_extended\_response" on page 31, "OBDonUDS\_RequestSupportedDTCEExtended" on page 183.

### Class-method version:

"ParseResponse\_RequestSupportedDTCEExtended" on page 92

## 3.8.18 OBDonUDS\_ParseResponse\_RequestDTCEExtended

Parses a response to requested service 19-06 (Request DTCEExtendedDataRecord).

### Syntax

#### C/C++

```
obd_status OBDonUDS_ParseResponse_RequestDTCEExtended(
    obd_msg* msg_response,
    obd_request_dtc_extended_response* out_buffer);
```

## Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

## Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

## Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

## Example

This example shows how to parse a response previously obtained using `OBDonUDS_RequestDTCExtended` and `OBDonUDS_WaitForService`.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_dtc_extended_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestDTCExtended(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, DTC identifier 0x%06X, DTC status 0x%02X,
              DTC extended data record number 0x%02X, %d bytes\n\t\t",
              parsed_response.report_type, parsed_response.DTC_identifier, parsed_response.status_of_dtc,
              parsed_response.DTC_extended_data_record_number, parsed_response.nb_elements);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("02X ", parsed_response.elements[ip]);
        }
        printf("\n");
    }
}
else
{
    printf("Parse : ERROR (0x%X)\n", status);
}
// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

### See also:

"obd\_msg" on page 18, "obd\_request\_dtc\_extended\_response" on page 32, "OBDonUDS\_RequestDTCExtended" on page 185.

### Class-method version:

"ParseResponse\_RequestDTCExtended" on page 93

## 3.8.19 OBDonUDS\_ParseResponse\_RequestDTCForAReadinessGroup

Parses a response to requested service 19-56 (Request DTCs for a ReadinessGroup).

### Syntax

### C/C++

```
obd_status OBDonUDS_ParseResponse_RequestDTCForAReadinessGroup(
    obd_msg* msg_response,
    obd_request_dtc_for_a_readiness_group_response* out_buffer);
```

### Parameters

Parameter	Description
msg_response	Response to be parsed
out_buffer	Parsed result

### Returns

The return value is an obd\_status code.

POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_CAUTION\_NRC\_NOT\_NULL is returned as a caution when the response parsed is a negative response.

The typical errors in case of failure are:

POBDONUDS_STATUS_INVALID_SI	Indicates that the Service ID found while parsing is invalid (null pointer or wrong value)
POBDONUDS_STATUS_INVALID_DATA_LENGTH	Indicates that the length found while parsing is invalid
POBDONUDS_STATUS_NETWORK_ERROR	Indicates that the Network result found while parsing is invalid (null pointer or value indicating an error)

### Remarks

The result must be freed afterwards by calling OBDonUDS\_ParsedResponseFree.

In case of negative response, its NRC is available as a part of the parsed response.

### Example

This example shows how to parse a response previously obtained using OBDonUDS\_RequestDTCForAReadinessGroup and OBDonUDS\_WaitForService.

### C/C++

```
obd_msg msg_response;
//...

// Parse response
obd_request_dtc_for_a_readiness_group_response parsed_response;
memset(&parsed_response, 0, sizeof(parsed_response));
obd_status status = OBDonUDS_ParseResponse_RequestDTCForAReadinessGroup(&msg_response, &parsed_response);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Print parsed response
    printf("Response from ECU #d : ", parsed_response.ecu_address);
    if (parsed_response.nrc != 0)
        printf("Negative response code 0x%02X\n", parsed_response.nrc);
    else
    {
        printf("Report type 0x%02X, Functional group identifier 0x%02X, DTC status availability mask 0x%02X,
            DTC format identifier 0x%02X, readiness group identifier 0x%02X", parsed_response.report_type,
            parsed_response.functional_group_identifier, parsed_response.DTC_status_availability_mask,
            parsed_response.DTC_format_identifier, parsed_response.readiness_group_identifier);
        for (uint32_t ip = 0; ip < parsed_response.nb_elements; ++ip)
        {
            printf("\n Element %d : ", ip + 1);
            printf("DTC identifier 0x%06X, Status of DTC 0x%02X", parsed_response.elements[ip].DTC_identifier,
                parsed_response.elements[ip].status_of_dtc);
        }
        printf("\n");
    }
}
else
{
    printf("Parse: ERROR (0x%X)\n", status);
}

// Free message
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

**See also:**

"obd\_msg" on page 18, "obd\_request\_dtc\_for\_a\_readiness\_group\_response" on page 33, "OBDonUDS\_RequestDTCForAReadinessGroup" on page 186.

**Class-method version:**

"ParseResponse\_RequestDTCForAReadinessGroup" on page 95

3.8.20 OBDonUDS\_Reset

Resets the receive and transmit queues of a PCAN-OBDonUDS channel.

**Syntax**

**C/C++**

```
obd_status OBDonUDS_Reset(
    cantp_handle channel);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical error in case of failure is:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application.
----------------------------------	---

**Remarks**

Calling this function ONLY clears the queues of a channel. A reset of the CAN controller doesn't take place by default (see PCAN-Basic parameter PCAN\_HARD\_RESET\_STATUS).

**Example**

The following example shows the use of the function OBDonUDS\_Reset on the channel PCANTP\_HANDLE\_PCIBUS1.

Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized.

**C/C++**

```
obd_status status;
status = OBDonUDS_Reset(PCANTP_HANDLE_PCIBUS1);
if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    printf("An error occurred\n");
else
    printf("PCAN-PCI (Ch-1) was reset\n");
```

**See also:**

"OBDonUDS\_Uninitialize" on page 138

**Class-method version:**

"Reset" on page 96

3.8.21 OBDonUDS\_FindOBDonEDS

Tests a channel to detect ECUs responding in OBDonEDS-mode.

**Syntax**

**C**

```
obd_status OBDonUDS_FindOBDonEDS(cantp_handle channel,
    cantp_baudrate baudrate,
    cantp_hwtype hw_type,
    uint32_t io_port,
    uint16_t interrupt);
```

**C++**

```
obd_status OBDonUDS_FindOBDonEDS(cantp_handle channel,
    cantp_baudrate baudrate =static_cast<cantp_baudrate>(OBD_BAUDRATE_AUTODETECT),
    cantp_hwtype hw_type =static_cast<cantp_hwtype>(0),
    uint32_t io_port =0,
    uint16_t interrupt =0);
```

**Parameters**

Parameter	Description
channel	The PCAN-ISO-TP channel handle to be used as OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
baudrate	The CAN hardware baudrate (see cantp_baudrate at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50, "obd_baudrate" on page 45). The baudrate is limited to legislated-OBD baudrate or the auto-detection value.
hw_type	NON PLUG&PLAY: The type of hardware and operation mode (see cantp_hwtype at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
io_port	NON PLUG&PLAY: The I/O address for the parallel port
Interrupt	NON PLUG&PLAY: Interrupt number of the parallel port

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

POBDONUDS\_STATUS\_UNSUPPORTED\_ECUS is returned if no OBDonEDS ECUs where found.

The typical errors in case of failure are:

POBDONUDS_STATUS_ALREADY_INITIALIZED	Indicates that the research could not take place because the channel was already in use.
PUDS_STATUS_FLAG_PCAN_STATUS	Indicates that the research could not take place. This error flag states that the error is composed of a more precise PCAN-Basic error (see uds_status at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

**Remarks**

This function must be called on a closed channel and leaves the channel closed.

Example

This example tries to find OBDonEDS ECUs on the PCANTP\_HANDLE\_USBBUS1 channel.

It is assumed that the channel has not been previously initialized.

C/C++

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
obd_status status = OBDonUDS_FindOBDonEDS(channel, (cantp_baudrate)OBD_BAUDRATE_AUTODETECT, (cantp_hwtype)(0), 0, 0);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Found OBDonEDS (OBDII) ECUs\n");
}
else
{
    if (!OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_UNSUPPORTED_ECUS, false))
    {
        printf("Unattended status\n");
    }
}
```

See also:

"OBDonUDS\_Initialize" on page 137

Class-method version:

"FindOBDonEDS" on page 97

3.8.22 OBDonUDS\_RequestCurrentData

Request "Current Powertrain Diagnostic Data" using service \$22-DID.

Syntax

C/C++

```
obd_status OBDonUDS_RequestCurrentData(
    cantp_handle channel,
    obd_netaddrinfo nai,
    obd_msg* out_msg_request,
    obd_DID_t* data_identifier,
    uint32_t data_identifier_length);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see "obd_netaddrinfo" on page 16)
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDonUDS_WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of DIDs) of the "data_identifier" array

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the number and nature of DIDs passed.

### Remarks

Once the request is sent, to get responses, `OBDonUDS_WaitForServiceXXX` must be called passing `out_msg_request`.

The `out_msg_request` must be freed afterwards by calling `OBDonUDS_MsgFree`.

### Example

This example sends a physically-addressed request "Current Powertrain Diagnostic Data" on channel `PCANTP_HANDLE_USBBUS1` to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

### C/C++

```
// Request current data values of 3 DID using physical address scheme (point to point)
obd_DID_t DIDs[3] = { 0xF415, 0xF401, 0xF405 };
obd_ecu myECU = POBD_ECU_1;
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = myECU;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestCurrentData(PCANTP_HANDLE_USBBUS1, nai, &msg_request, DIDs, 3);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

### See also:

"`OBDonUDS_WaitForService`" on page 188, "`OBDonUDS_WaitForServiceFunctional`" on page 190

### Class-method version:

"`RequestCurrentData`" on page 102



### 3.8.23 OBDOnUDS\_RequestFreezeFrameData

Sends a request to service 19-04 (Request Powertrain Freeze Frame Data).

**Syntax**

**C/C++**

```
obd_status OBDOnUDS_RequestFreezeFrameData(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    obd_DTC_t DTC_Number,  
    uint8_t record_number);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see "obd_netaddrinfo" on page 16))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
DTC_Number	DTC Mask Record (see " obd_DTC_t" on page 48)
record_number	DTCSnapshotRecordNumber (see OBDONUDS_DTC_SNAPSHOT_RECORD_xxx at "Definitions" on page 194)

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the record_number.

**Remarks**

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.

## Example

This example sends a physically-addressed request "Powertrain Freeze Frame Data" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_DTC_t dtc_number = 0x00014211;
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestFreezeFrameData(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, dtc_number, OBDONUDS_DTC_SNAPSHOT_RECORD_FIRST);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestFreezeFrameData" on page 104

### 3.8.24 OBDOnUDS\_RequestConfirmedTroubleCodes

Sends a request to service 19-42 (Request Emission-Related Diagnostic Trouble Codes with Confirmed Status).

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_RequestConfirmedTroubleCodes(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    uint8_t functional_group_identifier,  
    uint8_t DTC_status_mask,  
    uint8_t DTC_severity_mask);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP at "Definitions" on page 194)
DTC_status_mask	DTCStatusMask (see OBDONUDS_DTC_STATUS_CONFIRMED at "Definitions" on page 194)
DTC_severity_mask	DTCSeverityMask (see OBDONUDS_DTC_SEVERITY_CLASS_1 at "Definitions" on page 194)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier, the DTC_status_mask, the DTC_severity_mask.

#### Remarks

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.

Example

This example sends a physically-addressed request "Emission-Related Diagnostic Trouble Codes with Confirmed Status" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C/C++

```
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestConfirmedTroubleCodes(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, OBDONUDS_EMISSION_SYSTEM_GROUP, OBDONUDS_DTC_STATUS_CONFIRMED,
    OBDONUDS_DTC_SEVERITY_CLASS_1);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

Class-method version:

"RequestConfirmedTroubleCodes" on page 105

3.8.25 OBDonUDS\_ClearTroubleCodes

Sends a request to service 14 (Clear/reset Emission-related Diagnostic Information).

Syntax

C/C++

```
obd_status OBDonUDS_ClearTroubleCodes(
    cantp_handle channel,
    obd_netaddrinfo nai,
    obd_msg* out_msg_request,
    obd_DTC_t group_of_DTC);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDonUDS_WaitForServiceXXX
group_of_DTC	GroupOfDTC (see " obd_DTC_t" on page 48, OBDONUDS_DTC_xxx at "Definitions" on page 194)

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the group_of_DTC.

## Remarks

Once the request is sent, to get responses, OBDonUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDonUDS\_MsgFree.

## Example

This example sends a functionally-addressed request "Clear/reset Emission-related Diagnostic Information" on channel PCANTP\_HANDLE\_USBBUS1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_DTC_t emissionSystemGroup = OBDonUDS_DTC_EMISSION_SYSTEM_GROUP;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_ClearTroubleCodes(PCANTP_HANDLE_USBBUS1,
        OBD_NAI_REQUEST_FUNCTIONAL_11B, &msg_request, emissionSystemGroup);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"ClearTroubleCodes" on page 107

### 3.8.26 OBDOnUDS\_RequestTestResults

Sends a request to service 22-MID (Request On-board Monitoring Test Results for Specific Monitored Systems).

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_RequestTestResults(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    obd_DID_t* data_identifier,  
    uint32_t data_identifier_length);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (MIDs) (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of MIDs) of "data_identifier"

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the data_identifier_length, the data identifiers values.

#### Remarks

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.

## Example

This example sends a physically-addressed request "On-board Monitoring Test Results for Specific Monitored Systems" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_DID_t mid = 0xF601;
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestTestResults(PCANTP_HANDLE_USBBUS1, nai, &msg_request, &mid, 1);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestTestResults" on page 108

### 3.8.27 OBDOnUDS\_RequestPendingTroubleCodes

Sends a request to service 19-42 (Request Emission-related Diagnostic Trouble Codes with Pending Status).

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_RequestPendingTroubleCodes(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    uint8_t functional_group_identifier,  
    uint8_t DTC_status_mask,  
    uint8_t DTC_severity_mask);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
DTC_status_mask	DTCStatusMask (see OBDONUDS_DTC_STATUS_PENDING at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
DTC_severity_mask	DTCSeverityMask (see OBDONUDS_DTC_SEVERITY_CLASS_1 at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier, the DTC_status_mask, the DTC_severity_mask.

#### Remarks

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.



## Example

This example sends a physically-addressed request "Emission-related Diagnostic Trouble Codes with Pending Status" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestPendingTroubleCodes(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, OBDONUDS_EMISSION_SYSTEM_GROUP, OBDONUDS_DTC_STATUS_PENDING,
    OBDONUDS_DTC_SEVERITY_CLASS_1);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestPendingTroubleCodes" on page 110

### 3.8.28 OBDonUDS\_RequestControlOperation

Sends a request to service 31 (Request Control of On-Board System, Test, or Component).

#### Syntax

#### C/C++

```
obd_status OBDonUDS_RequestControlOperation(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    uint8_t routine_control_type,  
    obd_RID_t routine_identifier,  
    uint8_t* routine_control_options,  
    uint8_t routine_control_options_len);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDonUDS_WaitForServiceXXX
routine_control_type	Routine Control Type (see OBDONUDS_ROUTINE_START)
routine_identifier	RID (see " obd_RID_t" on page 49)
routine_control_options	RoutineControlOptionRecord, array of "routine_control_options_len" routine Control Option (byte) (may be NULL)
routine_control_options_len	Length of "routine_control_options" (may be 0)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the routine_control_type.

#### Remarks

Once the request is sent, to get responses, OBDonUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDonUDS\_MsgFree.

## Example

This example sends a physically-addressed request "Control of On-Board System, Test, or Component" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_RID_t rid = 0xE001;
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestControlOperation(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, OBDonUDS_ROUTINE_START, rid, NULL, 0);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestControlOperation" on page 112

### 3.8.29 OBDOnUDS\_RequestVehicleInformation

Sends a request to service 22 DID (Request Vehicle Information).

**Syntax**

**C/C++**

```
obd_status OBDOnUDS_RequestVehicleInformation(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    obd_DID_t* data_identifier,  
    uint32_t data_identifier_length);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
data_identifier	Array of "data_identifier_length" DIDs (ITIDs) (see "obd_DID_t" on page 48)
data_identifier_length	Length (number of ITIDs) of "data_identifier"

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the data_identifier_length, the ITIDs values.

**Remarks**

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.

Example

This example sends a functionally-addressed request "Vehicle Information" on channel PCANTP\_HANDLE\_USBBUS1.  
Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C/C++

```
obd_DID_t VInitid = 0xF802;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDOnUDS_RequestVehicleInformation(PCANTP_HANDLE_USBBUS1,
    OBD_NAI_REQUEST_FUNCTIONAL_11B, &msg_request, &VInitid, 1);
if (OBDOnUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDOnUDS_MsgFree(&msg_request);
```

See also:

"OBDOnUDS\_WaitForService" on page 188, "OBDOnUDS\_WaitForServiceFunctional" on page 190

Class-method version:

"RequestVehicleInformation" on page 113

3.8.30 OBDOnUDS\_RequestPermanentTroubleCodes

Sends a request to service 19-55 (Request Emission-related Diagnostic Trouble Codes with Permanent Status).

Syntax

C/C++

```
obd_status OBDOnUDS_RequestPermanentTroubleCodes(
    cantp_handle channel,
    obd_netaddrinfo nai,
    obd_msg* out_msg_request,
    uint8_t functional_group_identifier);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP)

Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier.

### Remarks

Once the request is sent, to get responses, `OBDonUDS_WaitForServiceXXX` must be called passing `out_msg_request`.

The `out_msg_request` must be freed afterwards by calling `OBDonUDS_MsgFree`.

### Example

This example sends a physically-addressed request "Emission-related Diagnostic Trouble Codes with Permanent Status" on channel `PCANTP_HANDLE_USBBUS1` to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

### C/C++

```
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestPermanentTroubleCodes(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, OBDONUDS_EMISSION_SYSTEM_GROUP);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

### See also:

"`OBDonUDS_WaitForService`" on page 188, "`OBDonUDS_WaitForServiceFunctional`" on page 190

### Class-method version:

"`RequestPermanentTroubleCodes`" on page 115

### 3.8.31 OBDOnUDS\_RequestSupportedDTCExtended

Sends a request to service 19-1A (Request Supported DTCExtendedRecord Information).

**Syntax**

**C/C++**

```
obd_status OBDOnUDS_RequestSupportedDTCExtended(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    uint8_t DTC_extended_DTC_data_record);
```

**Parameters**

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDOnUDS_WaitForServiceXXX
DTC_extended_DTC_data_record	DTC Extended DTC Data Record

**Returns**

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the DTC_extended_DTC_data_record.

**Remarks**

Once the request is sent, to get responses, OBDOnUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDOnUDS\_MsgFree.

## Example

This example sends a physically-addressed request "Supported DTCEXtendedRecord Information" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestSupportedDTCEXtended(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, 0x90);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestSupportedDTCEXtended" on page 116



### 3.8.32 OBDonUDS\_RequestDTCEExtended

Sends a request to service 19-06 (Request DTCEExtendedDataRecord).

#### Syntax

#### C/C++

```
obd_status OBDonUDS_RequestDTCEExtended(  
    cantp_handle channel,  
    obd_netaddrinfo nai,  
    obd_msg* out_msg_request,  
    obd_DTC_t DTC_mask,  
    uint8_t DTC_extended_data_record_number);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDonUDS_WaitForServiceXXX
DTC_mask	DTC Mask Record (see " obd_DTC_t" on page 48)
DTC_extended_data_record_number	DTC Ext Data Record Number

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the DTC_extended_data_record_number.

#### Remarks

Once the request is sent, to get responses, OBDonUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDonUDS\_MsgFree.

Example

This example sends a physically-addressed request "DTCEXtendedDataRecord" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

C/C++

```
obd_DTC_t mask = 0x00013001;
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestDTCEXtended(PCANTP_HANDLE_USBBUS1, nai, &msg_request, mask, 0x92);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

See also:

"OBDonUDS\_WaitForService" on page 188, "OBDonUDS\_WaitForServiceFunctional" on page 190

Class-method version:

"RequestDTCEXtended" on page 118

3.8.33 OBDonUDS\_RequestDTCForAReadinessGroup

Sends a request to service 19-56 (Request DTCs for a ReadinessGroup).

Syntax

C/C++

```
obd_status OBDonUDS_RequestDTCForAReadinessGroup(
    cantp_handle channel,
    obd_netaddrinfo nai,
    obd_msg* out_msg_request,
    uint8_t functional_group_identifier,
    uint8_t readiness_group_identifier);
```

Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDonUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
nai	Network Addressing Information (see obd_netaddrinfo))
out_msg_request	Resulting request added to the Tx Queue (see "obd_msg" on page 18), to be later used with OBDonUDS_WaitForServiceXXX
functional_group_identifier	FunctionalGroupIdentifier (see OBDONUDS_EMISSION_SYSTEM_GROUP)
readiness_group_identifier	ReadinessGroupIdentifier

## Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the functional_group_identifier, the readiness_group_identifier.

## Remarks

Once the request is sent, to get responses, OBDonUDS\_WaitForServiceXXX must be called passing out\_msg\_request.

The out\_msg\_request must be freed afterwards by calling OBDonUDS\_MsgFree.

## Example

This example sends a physically-addressed request "DTCs for a ReadinessGroup" on channel PCANTP\_HANDLE\_USBBUS1 to ECU 1.

Note: It is assumed that the channel was already initialized and the protocol OBD\_MSGPROTOCOL\_11BIT was retrieved.

## C/C++

```
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestDTCForAReadinessGroup(PCANTP_HANDLE_USBBUS1, nai,
    &msg_request, OBDONUDS_EMISSION_SYSTEM_GROUP, 0x02);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    printf("Request sent with success\n");
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

"OBDonUDS\_WaitForService" on the next page, "OBDonUDS\_WaitForServiceFunctional" on page 190

## Class-method version:

"RequestDTCForAReadinessGroup" on page 119

### 3.8.34 OBDOnUDS\_WaitForService

Handles the communication workflow for a PCAN-OBDOnUDS request expecting a single response.

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_WaitForService(  
    cantp_handle channel,  
    obd_msg* msg_request,  
    obd_msg* out_msg_response,  
    obd_msg* out_msg_request_confirmation);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
msg_request	A sent obd_msg message used as a reference to manage the OBDOnUDS service
out_msg_response	An obd_msg structure buffer to store the resulting POBDOnUDS response
out_msg_request_confirmation	An obd_msg structure buffer to store the resulting POBDOnUDS request confirmation

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers.
POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING	Indicates that the msg_request addressing mode is not physical.
(obd_status)PUDS_STATUS_SERVICE_TIMEOUT_CONFIRMATION	Timeout while waiting for request confirmation (request message loopback).
(obd_status)PUDS_STATUS_SERVICE_TIMEOUT_RESPONSE	Timeout while waiting for response message.
(obd_status)PUDS_STATUS_NO_MESSAGE	Indicates that no matching message was received in the given time.
(obd_status)PUDS_STATUS_NETWORK_ERROR	A network error occurred.
(obd_status)PUDS_STATUS_MESSAGE_BUFFER_ALREADY_USED	The given message buffer is already allocated, user must release the buffer before reusing it.

#### Remarks

The out\_msg\_response and out\_msg\_request\_confirmation must be freed afterwards by calling OBDOnUDS\_MsgFree.

## Example

This example sends a physically-addressed request "Current Powertrain Diagnostic Data" on channel PCANTP\_HANDLE\_USBUS1 to ECU 1, then it waits for responses by calling `OBDonUDS_WaitForService`.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

## C/C++

```
obd_DID_t DIDs[3] = { 0xF415, 0xF401, 0xF405 };
obd_netaddrinfo nai;
nai.protocol = OBD_MSGPROTOCOL_11BIT;
nai.source_addr = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
nai.target_addr = POBD_ECU_1;
nai.target_type = OBD_ADDRESSING_PHYSICAL;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestCurrentData(PCANTP_HANDLE_USBUS1, nai, &msg_request, DIDs, 3);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Wait for responses
    obd_msg msg_response;
    memset(&msg_response, 0, sizeof(msg_response));
    obd_msg msg_request_confirmation;
    memset(&msg_request_confirmation, 0, sizeof(msg_request_confirmation));
    status = OBDonUDS_WaitForService(PCANTP_HANDLE_USBUS1, &msg_request, &msg_response,
                                    &msg_request_confirmation);
    if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    {
        printf("Response received with success\n");
    }
    else
    {
        printf("An error occurred\n");
    }
    // Free messages
    OBDonUDS_MsgFree(&msg_response);
    OBDonUDS_MsgFree(&msg_request_confirmation);
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## See also:

### Class-method version:

"WaitForService" on page 121

### 3.8.35 OBDOnUDS\_WaitForServiceFunctional

Handles the communication workflow for a PCAN-OBDOnUDS request expecting multiple responses.

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_WaitForServiceFunctional(  
    cantp_handle channel,  
    obd_msg* msg_request,  
    uint32_t max_msg_count,  
    bool wait_until_timeout,  
    obd_msg* out_msg_responses,  
    uint32_t* out_msg_count,  
    obd_msg* out_msg_request_confirmation);
```

#### Parameters

Parameter	Description
channel	A PCANTP channel handle representing an OBDOnUDS client (see cantp_handle at "PCAN-UDS and PCAN-ISO-TP Dependencies" on page 50)
msg_request	A sent obd_msg message used as a reference to manage the OBDOnUDS service
max_msg_count	Length of the buffer array "out_msg_responses" (max. messages that can be received)
wait_until_timeout	if FALSE, the function is interrupted if "out_msg_count" reaches "max_msg_count".
out_msg_responses	A buffer to store the resulting responses. It must be an array of "max_msg_count" entries (it must have at least a size of max_msg_count * sizeof(obd_msg) bytes)
out_msg_count	Resulting actual number of messages read
out_msg_request_confirmation	An obd_msg structure buffer to store the resulting POBDOnUDS request confirmation

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

PUDS\_STATUS\_OVERFLOW indicates success but output responses buffer is too small to hold all responses.

The typical errors in case of failure are:

POBDONUDS_STATUS_NOT_INITIALIZED	Indicates that the given channel was not found in the list of reserved channels of the calling application
POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that one of the parameters passed to the function is invalid. Check the pointers, the max msg count.
POBDONUDS_STATUS_INVALID_REQUEST_ADDRESSING	Indicates that the msg_request addressing mode is not functional.
PUDS_STATUS_SERVICE_TX_ERROR	Indicates that an error occurred while transmitting the request.
PUDS_STATUS_SERVICE_TIMEOUT_CONFIRMATION	Indicates that timeout was reached while waiting for request confirmation.
PUDS_STATUS_NO_MESSAGE	Indicates that no matching message was received (in the given time).
PUDS_STATUS_NETWORK_ERROR	Indicates that a network error occurred.
PUDS_STATUS_SERVICE_RX_OVERFLOW	Indicates that service received more messages than input buffer expected.
PUDS_STATUS_MESSAGE_BUFFER_ALREADY_USED	Indicates that the given message buffer is already allocated, user must release the buffer before reusing it.

#### Remarks

Each message of out\_msg\_responses and the out\_msg\_request\_confirmation must be freed afterwards by calling OBDOnUDS\_MsgFree.

## Example

This example sends a functionally-addressed request "Vehicle Information" on channel PCANTP\_HANDLE\_USBBUS1, then it waits for responses by calling `OBDonUDS_WaitForServiceFunctional`.

Note: It is assumed that the channel was already initialized and the protocol `OBD_MSGPROTOCOL_11BIT` was retrieved.

## C/C++

```
// Request vehicle identification number (VIN) using functional addressing scheme
obd_DID_t VINitid = 0xF802;
obd_msg msg_request;
memset(&msg_request, 0, sizeof(msg_request));
obd_status status = OBDonUDS_RequestVehicleInformation(PCANTP_HANDLE_USBBUS1, OBD_NAI_REQUEST_FUNCTIONAL_11B,
    &msg_request, &VINitid, 1);
if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
{
    // Wait for responses
    obd_msg msg_responses[256];
    memset(&msg_responses, 0, 256*sizeof(obd_msg));
    uint32_t nb_responses = 0;
    obd_msg msg_request_confirmation;
    memset(&msg_request_confirmation, 0, sizeof(msg_request_confirmation));
    obd_status status = OBDonUDS_WaitForServiceFunctional(PCANTP_HANDLE_USBBUS1, &msg_request,
        256, false, msg_responses, &nb_responses, &msg_request_confirmation);
    if (OBDonUDS_StatusIsOk(status, POBDONUDS_STATUS_OK, false))
    {
        printf("%d Responses received with success\n", nb_responses);
    }
    else
    {
        printf("An error occurred\n");
    }
    // Free messages
    OBDonUDS_MsgFree(&msg_request_confirmation);
    for (uint32_t ir = 0; ir < nb_responses; ++ir)
        OBDonUDS_MsgFree(&(msg_responses[ir]));
}
else
{
    printf("An error occurred\n");
}
// Free message
OBDonUDS_MsgFree(&msg_request);
```

## Class-method version:

"WaitForServiceFunctional" on page 123

### 3.8.36 OBDOnUDS\_MsgFree

Deallocates a PCAN-OBDOnUDS message.

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_MsgFree(  
    obd_msg* msg_buffer);
```

#### Parameters

Parameter	Description
msg_buffer	An allocated obd_msg structure buffer (see "obd_msg" on page 18)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.

The typical errors in case of failure are:

POBDONUDS_STATUS_PARAM_INVALID_VALUE	Indicates that the msg_buffer or one of its fields is NULL
(obd_status)POBDONUDS_STATUS_CAUTION_BUFFER_IN_USE	Indicates that the message structure is currently in use, it cannot be deleted

#### Example

#### C/C++

```
obd_msg msg_request;  
//...  
  
obd_status status = OBDOnUDS_MsgFree(&msg_request);
```

#### Class-method version:

"MsgFree" on page 124

### 3.8.37 OBDOnUDS\_ParsedResponseFree

Deallocates a parsed response

#### Syntax

#### C/C++

```
obd_status OBDOnUDS_ParsedResponseFree(obd_response_generic* response);
```

#### Parameters

Parameter	Description
response	The parsed response to be freed (see "obd_response_generic" on page 18)

#### Returns

The return value is an obd\_status code. POBDONUDS\_STATUS\_OK is returned on success.



The typical error in case of failure is:

POBDONUDS\_STATUS\_PARAM\_INVALID\_VALUE    Indicates that the msg\_buffer or one of its fields is NULL or incorrect.

---

### Remarks

For all services, this function is to be used after calls to functions OBDonUDS\_ParseXXXX, to free the structures obd\_request\_XXX\_response, that must be casted to the generic structure obd\_response\_generic.

### Example

#### C/C++

```
obd_request_current_data_response parsed_response;  
// ...  
OBDonUDS_ParsedResponseFree((obd_response_generic*)&parsed_response);
```

### Class-method version:

"ParsedResponseFree" on page 125

## 3.9 Definitions

### 3.9.1 Miscellaneous

Constant	Value	Description
OBDONUDS_EMISSION_SYSTEM_GROUP	0x33	Functional Group Identifier: Emission System Group
OBDONUDS_DTC_SEVERITY_CLASS_1	0x02	DTC Severity Mask: DTC Severity Class 1
OBDONUDS_DTC_STATUS_CONFIRMED	0x08	DTC Status Mask: Confirmed Trouble Codes
OBDONUDS_DTC_STATUS_PENDING	0x04	DTC Status Mask: Pending Trouble Codes
OBDONUDS_ROUTINE_START	0x01	Routine Control Type: Start Routine
OBDONUDS_DTC_SNAPSHOT_RECORD_FIRST	0x00	DTC Snapshot Record Number: First
OBDONUDS_DTC_SNAPSHOT_RECORD_LAST	0xF0	DTC Snapshot Record Number: Last
OBDONUDS_DTC_EMISSION_SYSTEM_GROUP	0xFFFF33	Group of DTC: Emission System Group
OBDONUDS_DTC_ALL_GROUPS	0FFFFFFF	Group of DTC: All Groups

### 3.9.2 POBDONUDS parameter value definitions

See "POBDONUDS\_PARAMETER\_DEBUG" on page 43

## 4 Additional information

PCAN is the platform for PCAN-OBDonUDS, PCAN-UDS, PCAN-ISO-TP, and PCAN-Basic. In the following topics there is an overview of PCAN and the fundamental practice with the interface DLL CanApi2 (PCAN-API).

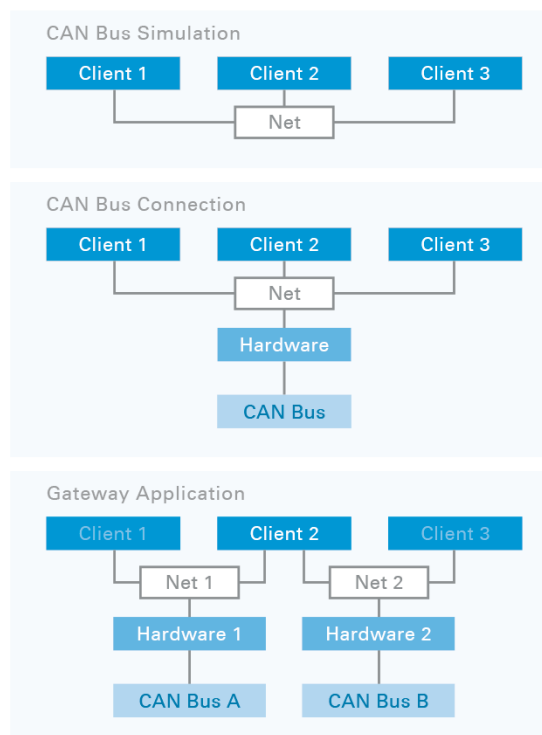
- PCAN Fundamentals: This section contains an introduction to PCAN
- PCAN-Basic: This section contains general information about the PCAN-Basic API
- OBDonUDS, UDS, and ISO-TP Network Addressing Information: This section contains general information about the ISO-TP network addressing format

### 4.1 PCAN Fundamentals

PCAN stands for PEAK CAN and is a flexible system for planning, developing, and using Controller Area Networks (CAN). It is a powerful product for both the developer and the end-user.

The PCAN system consists of a collection of Windows device drivers which allow the connection of Windows applications to all CAN buses physically connected via a PEAK CAN interface hardware. The interface to the user and the manager of a CAN-equipped installation are the so-called PCAN clients. With their help, process factors are controlled and visualized. The drivers permit the connection of several clients, which are then able to communicate via the CAN buses. Furthermore, several hardware components are supported, which are based on the CAN controller Philips SJA1000.

So-called nets are available. These specify virtual CAN buses that are extended into the PC. Several clients can connect to a virtual CAN bus. The connection to the outside world (a physical CAN bus) is possible with a hardware interface, e.g. the PCAN-USB or the PCAN-PCI Express card. The following figures give an overview of possible configurations.



The following rules apply to PCAN clients, nets, and hardware:

- One or multiple clients can be connected with a net
- A client can be connected with multiple nets

- A net is connected to no or exactly one active hardware
- Multiple connections to different nets can be defined for a hardware
- A maximum of one defined connection to a net can be activated for a hardware
- When a client transmits, the message is passed to all other clients connected to the net and via the hardware to the external CAN bus
- If a message is received by the hardware, it is received by all clients connected to the net

Users of free software and APIs like PCAN-View do not have to define and manage nets. If PCAN-View is instructed to connect directly to a PCAN hardware, the application automatically creates a net for the selected hardware and automatically establishes a connection with this net.

## 4.2 PCAN-Basic

PCAN-Basic is a programming application interface (API) for CAN communication using the PCAN system of the company PEAK-System Technik GmbH. It comprises of a collection of Windows device drivers which allow the connection of Windows applications to all CAN buses physically connected via a PEAK CAN interface hardware. PCAN-Basic supports connecting several CAN channels simultaneously, from the same or different hardware types.

The following table shows the number of channels that can be connected per hardware type:

	PCAN-ISA	PCAN-Dongle	PCAN-PCI	PCAN-USB	PCAN-PC-Card	PCAN-LAN
Number of channels	8	1	16	16	2	16

### Using the PCAN-Basic API

PCAN-Basic offers the possibility to use several PCAN-Channels within the same application easily. The communication process is divided in 3 phases: initialization, interaction, and finalization of a PCAN-Channel.

- **Initialization:** In order to do CAN communication, it is necessary to initialize a CAN channel at first. This is done by calling the function `CAN_Initialize` (class-method: `Initialize`) or `CAN_InitializeFD` (class-method: `InitializeFD`) in case CAN FD communication is desired.
- **Interaction:** After a successful initialization, the channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions `CAN_Read` and `CAN_Write` (class method versions: `Read` and `Write`) can then be used to read and write CAN messages. If the used channel is CAN FD capable and it was initialized using `CAN_InitializeFD`, the functions to be used are `CAN_ReadFD` and `CAN_WriteFD` (class method versions: `ReadFD` and `WriteFD`). If desired, additional configuration can be made to improve the communication session, like changing the message filter to target specific messages.
- **Finalization:** When the communication is finished, the function `CAN_Uninitialize` (class-method: `Uninitialize`) should be called in order to release the Channel and the resources allocated for it. With this, the channel is marked as "Free" and can be used by other applications.

## Hardware and Drivers

Overview of the current PCAN hardware and device drivers:

Hardware	Plug-and-Play Hardware	Driver
PCAN-Dongle	no	Pcan_dng.sys
PCAN-ISA	no	Pcan_isa.sys
PCAN-PC/104	no	Pcan_isa.sys
PCAN-PCI	yes	Pcan_pci.sys
PCAN-PCI Express	yes	Pcan_pci.sys
PCAN-PCI Express FD	yes	Pcan_pci.sys
PCAN-cPCI	yes	Pcan_pci.sys
PCAN-miniPCI	yes	Pcan_pci.sys
PCAN-miniPCle	yes	Pcan_pci.sys
PCAN-miniPCle FD	yes	Pcan_pci.sys
PCAN-M.2	yes	Pcan_pci.sys
PCAN-Chip PCle FD	yes	Pcan_pci.sys
PCAN-PC/104-Plus	yes	Pcan_pci.sys
PCAN-PC/104-Plus Quad	yes	Pcan_pci.sys
PCAN-PC/104-Express	yes	Pcan_pci.sys
PCAN-PC/104-Express FD	yes	Pcan_pci.sys
PCAN-ExpressCard	yes	Pcan_pci.sys
PCAN-ExpressCard 34	yes	Pcan_pci.sys
PCAN-USB	yes	Pcan_usb.sys
PCAN-USB FD	yes	Pcan_usb.sys
PCAN-USB Pro	yes	Pcan_usb.sys
PCAN-USB Pro FD	yes	Pcan_usb.sys
PCAN-USB Hub	yes	Pcan_usb.sys
PCAN-USB X6	yes	Pcan_usb.sys
PCAN-Chip USB	yes	Pcan_usb.sys
PCAN-PC Card	yes	Pcan_pcc.sys
PCAN-Ethernet Gateway DR	yes	Pcan_lan.sys
PCAN-Ethernet Gateway FD DR	yes	Pcan_lan.sys
PCAN-Wireless Gateway DR	yes	Pcan_lan.sys
PCAN-Wireless Gateway	yes	Pcan_lan.sys
PCAN-Wireless Automotive Gateway	yes	Pcan_lan.sys

## 4.3 OBDOnUDS, UDS, and ISO-TP Network Addressing Information

The PCAN-OBDonUDS API is built on top of the UDS and ISO-TP APIs. The following configuration is automatically set when the API is loaded in order to do legislated OBD-communication:

- Only the normal addressing format is used in the case of 11 bit CAN identifiers
- Only the normal fixed addressing format is used in the case of 29 bit CAN identifiers
- ISO-TP Blocksize parameter is defined to 0
- ISO-TP SeparationTime parameter is defined to 0

- ISO-TP WaitForTransmission parameter is defined to 0
- UDS max time in milliseconds to wait to receive the request loopback is defined to 250 ms
- UDS max time in milliseconds to wait to receive the message response indication is defined to 250 ms

Since the UDS API is already configured to allow legislated-OBDon communication, no extra configuration is done (see "UDS and ISO-TP Network Addressing Information" below).

#### 4.3.1 UDS and ISO-TP Network Addressing Information

The UDS API makes use of the PCAN-ISO-TP API to receive and transmit UDS messages. When a PCAN-OBDonUDS channel is initialized, the ISO-TP API is configured to allow the following communications:

- Functional request using 11 bits CAN identifier and normal addressing, from External Test Equipment address (PUDS\_ISO\_15765\_4\_ADDR\_TEST\_EQUIPMENT) to OBD functional address (PUDS\_ISO\_15765\_4\_ADDR\_OBD\_FUNCTIONAL)
- Physical requests and responses using 11 bits CAN identifier and normal addressing, between the External Test Equipment address (PUDS\_ISO\_15765\_4\_ADDR\_TEST\_EQUIPMENT) and standard ECU addresses (ECU #1 to #8)
- Communications with 29 bits CAN identifier and FIXED NORMAL addressing format
- Communications with 29 bits CAN identifier and MIXED addressing format

If an application requires other communication settings, it will have to be set through the PCAN-ISO-TP API. Once a PCAN-UDS channel is initialized, PCAN-ISO-TP specific functions (like CANTP\_AddMapping) can be called with this PCAN-UDS channel.